ADAPTIVE TASK PLANNING AND MOTION PLANNING FOR ROBOTS IN DYNAMIC ENVIRONMENTS

CUEBONG WONG

Design, Manufacturing & Engineering Management
University of Strathclyde, Glasgow
April 2020

A thesis presented in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy

CUEBONG WONG

Adaptive Task Planning and Motion Planning for Robots in Dynamic Environments

University of Strathclyde

Department of Design, Manufacturing & Engineering Management

James Weir Building

75 Montrose Street

Glasgow, United Kingdom

G1 1XJ

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Copyright Statement

Signed: Cuebong Wong

Date: 02/05/2020

To my family, who lovingly raised me with such care and compassion,
to whom I owe everything to.

"A good plan violently executed now is better than a perfect plan executed next week."

General George S. Patton, Jr.

Abstract

Emerging applications involving a high degree of *uncertainty, dynamics, variability* and *unpredictability* have begun to impart greater complexity to tasks performed by robots. In these kinds of environments, one of the most common causes for robot failures has been linked to the inability of the underlying planner to adapt to changing conditions in the environment. While an extensive range of methods have been developed to solve static planning problems in the robotics domain, until now solving the dynamic counterparts of these problems remain mostly elusive.

Motivated by these challenges, this thesis presents developments that advance the state-of-the-art in *optimal* task and motion planning to address the *dynamic* variant of common robotic planning problems. Studies into adaptive planning problems are conducted to investigate the challenges that arise when extending planning methods from offline planning to online planning. In particular, this research seeks to characterise the interactions between plan quality and computational efficiency when solving dynamic planning problems and to identify the practical considerations for implementing adaptive planning algorithms in physical systems. The contributions of this thesis are a number of fast yet practical planning techniques and methods that provide and maintain near-optimal, collision-free solutions to complex planning problems involving dynamic environments.

In this thesis I first describe a case study that examines the challenges unique to dynamic motion planning through a robotic pick and place task. The observations derived from this case study inspired the development of two new methods for solving complex task planning problems. The first of these is an adaptive task and path planning framework that addresses the optimal task planning problem for mobile robots under dynamic conditions. This framework integrates a sampling-based multi-goal path planning algorithm with symbolic task planning

to incrementally find high-quality task plans. Crucially, the framework supports *anytime*-like planning and dynamic re-planning of both tasks and low-level motions to enable fast and adaptive computation of optimal solutions. To support this, a tree pruning technique is proposed for multi-goal planning problems to substantially reduce the time and memory complexity of the planner.

In the second half of this thesis, I present a highly competitive clustering-based algorithm for robotic task sequencing problems (RTSPs). Unlike existing methods, the algorithm is capable of finding near-optimal solutions for complex tasks involving hard spatial constraints. With a view towards dynamic robotic task sequencing, I go on to introduce two new concepts to the RTSP. The first is partial planning, which adopts the idea of planning-during-execution to reduce the pre-execution planning time of an algorithm for online applications. The second is the concept of *dynamic* RTSPs, a new sub-class of RTSPs that involve dynamically-changing problem variables. I subsequently present an adaptive algorithm for online tracking of near-optimal RTSP solutions under dynamic influences. As a pioneering work within the scope of dynamic task sequencing, I provide a quantitative evaluation of the algorithm for the purpose of benchmarking in future developments.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor Erfu Yang for his continual support across all aspects of my life throughout my period of study. His supervision, care and undivided attention has been invaluable in my journey to pursue a PhD. I am thankful for the many opportunities he has provided me with to expand my horizon in the world of research, and for all the time he has made available to me at my times of need.

I am grateful to Carmelo Mineo for our stimulating discussions that have shaped my work in numerous ways, for providing training that has helped develop my proficiency in working with KUKA robots, for his support with conducting experiments in my research, for his perpetual energy and sense of humour that has kept me motivated to the very end, and most of all for the countless suggestions he has offered me over the course of my study that have taught me how to become a more effective researcher.

I would like to thank Xiu-Tian Yan and Dongbing Gu for their advice and useful discussions that has helped steer the direction of my PhD. The precious time they have spent to review the manuscripts I produced as part of my research has helped substantially improve my general academic writing skills.

I would like to acknowledge Jaime Zabalza, Zixiang Fei and Yijun Yan for their involvement in the case study reported in Chapter 3 of this thesis. Without their contributions in the area of machine vision it would not have been possible to develop a working physical system. I would also like to thank the Advanced Forming Research Centre for the provision of research funds through the Route to Impact scheme, which has supported research activities that include the above case study.

I am grateful to Quang-Cuong Pham and Francisco Suárez-Ruiz for their guidance on best practices in robotics research, for their provision of learning material to help build up my proficiency and confidence in developing robotic software,

and for hosting my visit to their research group at Nanyang Technological University, Singapore. Observing their disciplined approach to research has inspired me to stretch beyond my comfort zone.

I would like to thank Sen Wang for organising and facilitating my visit to Heriot-Watt University, where I was able to access necessary equipment to conduct important experiments for my research.

I gratefully acknowledge the funding received towards my PhD from the Engineering and Physical Sciences Research Council under its Doctoral Training Partnership Programme (DTP 2016-2017 University of Strathclyde, Glasgow, UK). Without their support I would not have had the opportunity to pursue this research.

Last but not least, I would like to express my eternal gratitude to my friends and family for their unconditional love and support during a period where I have deprived them of my attention. I would like to give special thanks to Chi Wai Cho for her patience and encouragement when I needed them most. Without these people I would not be where I am today. Thank you.

Cuebong Wong
April 2020

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

APF Artificial Potential Field

C-space Configuration space

CTMP Combined Task and Motion Planning

CTSP Clustered Travelling Salesman Problem

DLL Dynamic Link Library

DoF Degrees of Freedom

DRM Dynamic Roadmap

DRRT Dynamic Rapidly-exploring Random Tree

DRTSP Dynamic Robotic Task Sequencing Problem

DTSP Dynamic Task Sequencing Problem

EA Evolutionary Algorithm

EHC Enforced Hill Climbing

ERRT Execution extended Rapidly-exploring Random Tree

FF FastForward

GA Genetic Algorithm

GTSP Generalised Travelling Salesman Problem

IK Inverse Kinematics

IPC International Planning Competition

ITPP Integrated Task and Path Planning

ITRA Interfacing Toolbox for Robotic Arms

KRL KUKA Robot Language

LIN Linear path motion

LPG Local Planning Graphs

MP-RRT Multipartite Rapidly-exploring Random Tree

MWR Mobile Wheeled Robot

P3DX Pioneer 3-DX differential drive mobile robot

PRM Probabilistic Roadmap

PTP Point-To-Point

ROS Robot Operating System

ROV Remotely-Operated Vehicle

RRT Rapidly-exploring Random Tree

RSI Robot Sensor Interface

SCP Set Covering Problem

SCTSP Set Covering Travelling Salesman Problem

SO Shared Object

T-RRT Transition-based Rapidly-exploring Random Tree

TSP Travelling Salesman Problem

TSPN Travelling Salesman Problem with Neighbourhoods

UAV Unmanned Aerial Vehicle

Publications Arising from this Thesis

Articles

[J1] C. Wong, C. Mineo, E. Yang, X.T. Yan, D. Gu, 'A fast, near-optimal clustering-based algorithm for solving spatially-constrained robotic task sequencing problems', IEEE/ASME Trans. on Mechatronics. (Initial manuscript submitted on 13th March 2020.)

– This paper presents the work reported in Chapter 6. My contributions as the first author are as follows: I developed the presented algorithm, prepared the manuscript drafts and conducted all experiments.

[J2] C. Mineo, M. Vasilev, B. Cowan, C. N. MacLeod, S. Gareth, C. Wong, E. Yang, R. Fuentes, E. J. Cross, 'Enabling robotic adaptive behaviour capabilities for new industry 4.0 automated quality inspection paradigms', in Insight - Non-Destructive Testing and Condition Monitoring. (Accepted on 14th February 2020.)

– This paper presents the developments of the ITRA toolbox, which was used for the experiments presented in Chapters 3 and 6. My contributions as a co-author are as follows: I supported the development of the external control capabilities of the ITRA toolbox and edited the manuscript drafts.

[J3] J. Zabalza, Z. Fei, C. Wong, C. Mineo, E. Yang, T. Rodden, J. Mehnen, Q.C. Pham, J. Ren, 'Smart sensing and adaptive reasoning for enabling industrial robots with interactive human-robot capabilities in dynamic environments – a case study', in Sensors, 19(6), 1354, 2019.

– This paper reports the findings of the case study described in Chapter 3 with a primary focus on machine vision and integration. My contributions as a co-author are as follows: I developed the software for motion planning, supported the development of the user-interfacing software, conducted all experiments, wrote Section 4 of the paper and edited the manuscript drafts.

[J4] C. Wong, E. Yang, X.T. Yan, D. Gu, 'Autonomous robots for harsh environments: A holistic overview of current solutions and ongoing challenges', in Systems Science and Control Engineering, vol. 6, no. 1, pp 213-219, 2018.

– This paper presents a detailed survey on the advancement of autonomous robots for harsh environments as summarised in Chapter 1. My contributions as the first author are as follows: I conducted all literature review activities and prepared the manuscript drafts.

Conference proceedings

[C1] C. Wong, E. Yang, X.T. Yan, D. Gu, 'An optimal approach to anytime task and path planning for autonomous mobile robots in dynamic environments', in Towards Autonomous Robotic Systems: 20th Annual Conference, London, UK, 2019. (**Nominated** for the *Best Poster Award*.)

– This paper publishes the work reported in Chapter 5. My contributions as the first author is as follows: I developed the presented planning algorithm, conducted all experiments and prepared the manuscript drafts.

[C2] C. Wong, E. Yang, X.T. Yan, D. Gu, 'Dynamic anytime task and path planning for mobile robots', in UK-RAS19 Conference on Embedded Intelligence, Loughborough, UK, 2019. (**Awarded** with the *Best Paper Award*.)

– This paper provides a preliminary report of the work in Chapter 5 and has been extended into publication [C1]. My contributions as the first author is as follows: I developed the presented planning algorithm, conducted all experiments and prepared the manuscript drafts.

[C3] J. Zabalza, Z. Fei, C. Wong, C. Mineo, E. Yang, T. Rodden, J. Mehnen, Q.C. Pham, J. Ren, 'Making industrial robots smarter with adaptive reasoning

and autonomous thinking for real-time tasks in dynamic environments: a case study', in Int. Conf. on Brain Inspired Cognitive Systems, Xi'an, China, 2018.

– This paper gives a preliminary report of the case study described in Chapter 3 with a primary focus on machine vision and integration. This has been extended into the journal publication [J3]. My contributions as a co-author are as follows: I developed the software for motion planning, supported the development of the user-interfacing software, conducted all experiments, wrote Section 2 of the paper and edited the manuscript drafts.

[C4] C. Mineo, M. Vasilev, C.N. MacLeod, R. Su, S.G. Pierce, C. Wong, E. Yang, R. Fuentes, and E.J. Cross, 'Enabling robotic adaptive behaviour capabilities for new industry 4.0 automated quality inspection paradigms', in 57th Annual British Conference on Non-Destructive Testing, Nottingham, UK, 2018.

– This paper describes the use of the ITRA toolbox for industrial applications and is a reduced conference version of the article published as [J2]. My contributions as a co-author are as follows: I supported the development of the external control capabilities of the ITRA toolbox and edited the manuscript drafts.

[C5] C. Wong, E. Yang, X.T. Yan, D. Gu, 'Optimal path planning based on a multi-tree T-RRT* approach for robotic planning in continuous cost spaces', in 2018 12th France-Japan and 10th Europe-Asia Congress on Mechatronics, Tsu, Japan.

– This paper presents the work contained in Chapter 4. My contributions as the first author are as follows: I developed the presented algorithm, performed all experiments and prepared the manuscript drafts.

[C6] C. Wong, E. Yang, X.T. Yan, D. Gu, 'An overview of robotics and autonomous systems for harsh environments', in 2017 23rd IEEE Int. Conf. on Automation and Computing, Huddersfield, UK.

– This paper presents an overview of the advancement of autonomous robots for harsh environments as discussed in Chapter 1 and is a reduced conference version of the journal publication [J4]. My contributions as the first author are as follows: I conducted all literature review activities and prepared the manuscript drafts.

[C7] C. Wong, E. Yang, X.T. Yan, D. Gu, 'Adaptive and intelligent navigation of autonomous planetary rovers - a survey', in 2017 NASA/ESA Conf. on Adaptive Hardware and Systems, Pasadena, CA, USA.

– This paper presents a survey of adaptive navigation methods for autonomous planetary rovers, which is summarised in Chapters 1 and 2. My contributions as the first author are as follows: I conducted all literature review activities and prepared the manuscript drafts.

Technical Reports

[T1] C. Mineo, C. Wong, M. Vasilev, B. Cowan, C.N. MacLeod, R. Su, S.G. Pierce and E. Yang, 'Interfacing toolbox for robotic arms with real-time adaptive behaviour capabilities', tech. rep., University of Strathclyde, Oct. 2019.

– This technical report describes the ITRA toolbox, which was used for the experiments presented in Chapters 3 and 6. Unlike publications [J2] and [C4], this technical report provides a more detailed account of the benchmark results for the individual functions of ITRA. My contributions as a co-author are as follows: I supported the development of the external control capabilities of the ITRA toolbox and edited the manuscript drafts.

Chapter 1

Introduction

1.1 Motivation

Planning is a fundamental concept in robotics. For all autonomous agents, including biological beings (such as humans) and non-biological entities (e.g. autonomous robots), the paradigm of "sense, think, and act" forms the core of the agent's deliberation process in making informed decisions to act within its environment. Humans have learnt to develop this capability since birth, and are often able to achieve many basic actions without *explicit planning*. Through the process of continual learning, we have developed pre-stored plans for everyday skills such as walking, writing, throwing, lifting and so forth. We generally are not aware of deliberate planning until we are faced with new, complex tasks that are unfamiliar to us.

While humans have made the execution of many tasks appear easy, this is unfortunately not the case when translated to robotics. Even a seemingly simple problem of grasping an object is rather complex to replicate, as it involves a high Degree-of-Freedom (DoF) system, complex robot kinematics, object localisation and geometric perception, grasp positioning, and force control. In actual fact humans are able to achieve these tasks with ease due to the complex arrangement of billions of neurons that form highly specialised networks, which underlie the cognitive functions of the human brain. To date, researchers have only begun to understand the inner workings of this complex system, and while the field of artificial intelligence (AI) has sought to emulate these functions in non-biological systems, robots still lack the capability to reliably *learn* primitive behaviours and apply them in new tasks. This means even the simple act of moving between two

points (before we begin to consider collisions) involves explicit planning to ensure constraints such as joint limits and nonholonomic constraints are satisfied.

In the field of robotics, several forms of planning exist. The major classifications include motion planning, perception planning, manipulation planning, task planning and mission planning.

Motion planning considers the general class of problems that deal with low-level motion between a starting pose and a goal pose in space. By taking into account geometric knowledge of the environment, and the robot's kinematic and dynamic constraints, motion planning seeks to find a collision-free geometric **path** and the corresponding **trajectory** (a *time-parameterisation* of the path) to realise the motion.

Perception planning addresses the problem of sensor placements to optimally observe the entire set of manifolds for a particular scene of interest. One example of its application lies in remote visual mapping of large components such as wind turbine blades [1] and oil & gas tanks, where perception planning is used to minimise the number of observation points for a drone while guaranteeing a bounded level of coverage. In emerging assembly applications based on mobile sensor and mobile robot platforms, the success of precise part insertion in unstructured environments have relied upon sensor placement planning to obtain an optimal view of the insertion features [2].

Manipulation planning is concerned with problems involving the interaction of objects via contacts (e.g. using a robotic manipulator equipped with a gripper). It consists of planning stable grasp positions, handling of forces, as well as re-grasping for complex assembly tasks. Sensory capabilities are generally required to deliberate about the geometries of objects being manipulated.

In **task planning**, the objective is to find a valid sequence of actions to achieve a set of given objectives. Robotic task planning problems appear in many different forms. For example, a simple routing task can involve finding an optimal sequence to visit a set of goal locations in no particular order, while an exploration task may involve numerous actions such as sampling soil, docking, recharging battery, and taking images at points of interest. For more complex problems such as robotic assembly tasks, *task precedence* may dictate certain actions are performed before others. For example, obstacles may have to be moved to give access to a target object, or a drilling action must be performed before a dowel insertion action can be executed. Due to the highly abstracted

form of many actions, the validity of a plan generally involves additional motion and manipulation planning to determine the feasibility of individual actions.

Mission planning addresses planning problems of a high abstraction level and is more loosely defined. In some contexts involving single robot systems, the term mission planning have been used interchangeably with the concept of task planning. Conversely in the context of multi-robot systems (e.g. fleets and swarms), mission planning is generally defined as the planning of task allocation for distribution to the team of robotic agents. This takes into account parameters such as robot availability, battery level, robot capabilities and location. One common example can be found in the logistics environment, where mobile robot fleets operating in warehouses are used to provide transport operations as efficiently as possible. In these scenarios, mission planning precedes task planning, where the objectives given to an individual robot's task planner is provided by the mission planner. [3]

This thesis is specifically concerned with the areas of *motion planning* and *task planning*. Both of these areas have been studied extensively as independent disciplines, with large bodies of literature dedicated to extending existing methods and presenting new algorithms that seek to improve planning performance in terms of *completeness*, *convergence*, *generality* and *complexity*. However, in recent years, numerous authors have begun to investigate the concept of *combined task and motion planning* (CTMP), which has, by and large, focused on high-complexity planning problems involving manipulation tasks. CTMP fundamentally consists of leveraging the geometric and spatial relationships of the robot, the objects and the environment, derived from the continuous state space in motion planning, to guide the search for a feasible solution to a task planning problem.

So far, the majority of developments in CTMP has primarily focused on static planning problems, with few considerations for **optimal planning** in this domain. Yet the capability to develop optimal plans is important across many applications as it directly affects aspects such as safety, productivity, and energy-efficiency. Furthermore, while existing research in the area of motion planning has dedicated some attention to the problem of *dynamic re-planning*, this has been much less prominent in general task planning and indeed CTMP literature. Nevertheless, re-planning capabilities are crucial in the real world, particularly in modern applications that involve aspects of *uncertainty*, *variability*, *unpre-*

dictability and *dynamics*. This necessitates further development in the area of **adaptive planning**, which seeks to enable online re-planning capabilities to cope with these challenging environments.

To this end, **this thesis presents developments that extend the state-of-the-art in optimal task and motion planning towards dynamic re-planning**.

1.1.1 The Progression of Robotics and Autonomy

Modern robotics have become a cornerstone of technological advancement in almost all aspects of human life. From medical and care assistance to manufacturing and decommissioning, from disaster recovery to agriculture, and even in food production and pizza making, robots have undoubtedly cemented their importance in the 21st century.

While the exact origin of the term *robots* have long been debated due to its vague early definition and differences in its interpretation, one thing is for certain. The emergence of the first industrial robot in 1961 was the beginning of an *automation* era that would give rise to the third industrial revolution, where large-scale automation in production became widespread. Industrial robots became popular for reasons commonly coined as the 4 *Ds* - to replace work that were Dull, Dirty, Dangerous and Difficult.[1] As a result, industrial robots were widely known for their use in highly-repetitive, low-skilled tasks in rather harmful environments to human workers. They provided economic benefits too, as the efficiency of these systems often provided quick return on investment for mass production processes, and were much more reliable in extended runs than a human worker who would have otherwise been subjected to significant strain over long hours.

However, these traditional applications of robotics had their limitations, which confined the uptake of these technologies to large-scale industrial processes. Traditional automated robotic systems were generally purpose-built for a particular process, involving custom-designed fixtures caged within a safety enclosure for the safety of workers in its vicinity. Software to operate these systems were pre-

[1] In some contexts the 4^{th} D have been replaced by other reasons for the adoption of robots such as Dear (expensive manual labour), Dexterous (precision that is unachievable with human hands) and Domestic (fulfilling emotionally-sensitive tasks such as cleaning toilets, though without making a fuss!).

programmed by a skilled programmer with a sequence of precisely-defined tasks and actions, providing little to no tolerance for variation in the processes. This meant fixtures had to be carefully designed to reduce positional and geometric variability in parts and materials being handled as far as possible. Any changes to the process required considerable offline programming effort and, at times, the redesigning of fixtures, which is a costly and time-consuming procedure. Consequently, industrial robots have traditionally been limited to *fixed, structured* environments with minimal reliance on sensing technologies.

Since then, the concept of *autonomy* emerged, bringing with it a transformation in the landscape of robotics. Unlike traditional robotic systems that operated as a "pre-programmed machine", autonomy introduced the capability to compensate for variation and uncertainty internally without human intervention. This was a game-changer. Through the progressive developments in advanced control theory, sensing technologies and AI, robotics and autonomous systems had captured the interest of both the research community and industry leaders across the world for their potential to transform the way machines interacted with objects and environments. Robots began to possess basic planning, reasoning and decision-making skills that allowed them to "think" about the tasks they were required to perform, while having the tools they need to "see".

Providers of robotic systems began developing more sophisticated software packages that made it easier to program and re-program robots without extensive training. Universal Robots, for example, have now made it possible for operators to program a Universal Robot arm with just a few high-level commands to detect, grasp and arrange small components randomly scattered on a table. The task itself involves complex robot kinematics, machine vision and force-torque sensing, yet at no point is the operator required to access and manipulate the low-level code for simple behaviours. This provision of *primitive* skills have made robots much more accessible to SMEs, who previously did not possess the in-house specialists to program robots nor justify the large investments for costly production cells to minimize variation. This is reflected in the growth of industrial robots being used across the world, with a recent forecast of 3.1 million industrial robots to be deployed worldwide in 2020 [4].

Robots have now evolved beyond the conventional 6-DoF industrial robot and can be found in a variety of forms across many more challenging environments. Mobile ground vehicles have attracted the agriculture community for their po-

tential to replace human labour in the harvesting stage amidst concerns over the decreasing amount of labour in the sector [5]. Six-wheeled rovers [6] with flexible suspension systems have been deployed to Mars for the purpose of planetary exploration where no man has gone. In the field of disaster recovery, legged robot systems such as crawler robots [7] and humanoids [8] are being developed to navigate effectively through rugged terrain and interact with manipulatable objects and obstacles. Unmanned aerial vehicles (UAVs) have found significant importance in the fields of inspection [9, 10] and search and rescue [11] as they are agile and low-cost, and possess greater freedom to navigate their environments. Remotely-operated underwater vehicles (ROVs) have had notable success in deep-sea applications [12], while preliminary observations of humanoid robots with modified bases have proven their feasibility for manipulating objects in extreme environments such as the ocean [13] and in space [14].

There are indeed vast opportunities and possibilities for smarter, more intelligent robots to support all aspects of human life, but while major milestones have been achieved in the field of robotics to date, there are still long ways to go before robots can *truly* operate autonomously in the tasks that are now being demanded of them.

1.1.2 Limitations and Opportunities

Perhaps unsurprisingly, the aforementioned application areas in Section 1.1.1 introduce new challenges as robots are deployed beyond the stable environment of mass production cells that remain static over time. The harsher conditions of operating in the outside world require robots to possess the capability to adapt to environments that are *unstructured*, *uncertain*, *unpredictable* and *dynamic*. Unfortunately, cases of robotic failure is not uncommon [15, 16]. Commonly cited reasons for failure include the inability of a robot to handle small variations in a task (a problem that arose when tests were no longer performed in a controlled laboratory environment), limitations in algorithms to cope with new environments and challenges to adapt to changing environmental conditions (e.g. lighting variations and dynamically-moving objects).

These limitations are well recognised across various sectors as human input is still heavily relied upon for the safe operation of robots. In those highly dynamic applications where substantial offline programming proves to be too costly, tele-

operation and semi-autonomous operation modes are still the preferred method for planning and executing actions. This particularly applies to sensitive, high-risk applications where the *consequences* of failure is severe. Consider for example the task of sorting and segregating legacy nuclear waste. Damaged or corroded containers must be cut open so that internal legacy waste can be examined and sorted according to their contamination levels. Particularly highly contaminated waste must be identified, retrieved and packed into safer storage containers. These tasks all involve complex manipulation and grasping actions. To add to the complexity, the environment is dirty and conditions change progressively as operations take place, creating a very dynamic environment. This problem poses many challenges to autonomous robotics as decisions at several planning levels must be made. To date, this is still achieved with a high reliance on operators in a control room due to the strict requirements for reliability and robustness [17].

Nevertheless, increased levels of autonomy is highly sought after, if not essential, for these activities. Many of these scenarios involve remote environments that are either too far or too dangerous for humans to enter. This can often mean communications between operators in a control room and the robot is severely hindered. Past planetary rovers on Mars are particularly clear examples, where plans generated by human operators were limited to being sent once a *sol* (i.e. Martian day) due to delays in transmission [6]. Additional challenges are introduced in human-robot interactions within tele-operation due to differences in how data is perceived and processed by humans and robots, which can often lead to poor efficiency and the risk of human-error [15]. The operation of these robots also required lengthy training to ensure proper and effective control by operators, which, from an economic perspective, increases costs.

This creates opportunities for furthering the advancement of autonomous robotic technologies in areas of perception, planning and control to address aspects of *efficiency*, *optimality*, *flexibility* and *adaptiveness*. Motivated by these challenges and opportunities, this thesis presents a series of developments in task and motion planning to enable adaptive robotic behaviours that can adequately cope with changing observations obtained through perception. The planning algorithms seek to provide collision-free task plans that carry the necessary low-level motion commands required by a control system for safe execution on a robot. The methods described in this thesis are industrially relevant, seeking to pro-

vide near-optimal solutions while possessing efficient re-planning capabilities to circumvent modes of failure due to environmental variation.

1.2 Research Aim and Hypothesis

1.2.1 Research Aim

This research investigates the problem of optimal and adaptive planning for robots in dynamic environments. Specifically, I study problems that comprise of two aspects of planning common to almost all robotic planning applications: task planning and motion planning.

Task planning consists of finding a high-level sequence of discrete commands that enables a robot to achieve a set of goals when actions are performed in the given order from the initial world state. However, the high-level commands that require driving the actuators of the robot in some way do not provide the low-level control actions required to physically execute such commands. Motion planning resolves this problem by translating a high-level command to continuous low-level instructions to actuate the robot within its physical kinematic limits while avoiding collision with itself and the environment. Accordingly, solving complex task planning problems generally require the use of motion planning to first verify the feasibility of high-level commands (e.g. ensure that the robot can execute the action without collision) and subsequently provide the low-level instructions to execute them. In addition to this, motion planning can provide the cost of low-level motions required for solving **optimal** task planning problems, where each command incurs an action-specific cost (such as the duration required to perform the action).

Conventional planning problems assume that the environment is static and solutions obtained offline remain valid when sent for execution. In these scenarios, extensive planning time can generally be allocated to obtain high-quality solutions (relative to a particular optimisation criteria such as the time required by the robot to perform all actions in the plan). Indeed for complex planning problems, achieving a high-quality task plan has required extensive planning time ranging from minutes to hours. However, when addressing the dynamic variant of these problems, planners must be capable of finding plans within seconds to be of practical use in real-world applications. Solving dynamic planning prob-

lems become particularly challenging when the objective of the problem is to find high-quality (i.e. optimal) solutions online. While some research on adaptive task and motion planning do exist in literature, solving problems involving the combination of both optimal and dynamic considerations remains mostly elusive to this day.

The aim of this research is to identify common features of general dynamic task planning problems and to provide adaptive planning algorithms and techniques that enable the computation of low-cost solutions while achieving high planning efficiency for applications involving dynamically-changing environments.

1.2.2 Research Hypothesis

In this research, a number of developments are presented to advance the state-of-the-art in robotic planning. Some of these developments are applicable stand-alone for static planning problems, but these have been further extended into adaptive algorithms that can specifically cope with new observations of the environment. For each adaptive task and motion planning algorithm presented in this thesis, I test the following research hypothesis through experimental comparisons and empirical findings:

"The adaptive algorithm obtains near-optimal, collision-free solutions faster than an offline planning algorithm for a dynamic planning problem involving changes in the observation of the environment."

1.2.3 Research Questions

To develop a solution to each of the dynamic planning problems studied in this thesis, the following research questions were identified during the initial phase of literature review. Each of these questions proved difficult to address when applied to relevant existing work in literature that provides an offline solution to the static variant of the planning problems investigated. By addressing these research questions during the development of solutions to dynamic planning problems, I show that the presented adaptive algorithms successfully achieve the aim of this research while extending the state-of-the-art in robotic task and motion planning.

1. What are the necessary considerations for planning in dynamic environments?

2. What are the interactions between the goals of minimising plan cost and maximising planning efficiency when solving a task and motion planning problem?

3. How can problems that have conventionally been solved offline as a static planning problem be addressed more efficiently to enable online planning?

4. What are the practical considerations for implementing adaptive planning algorithms in physical robots?

1.2.4 Research Objectives

To achieve the research aim described in this Chapter, the following objectives have been identified taking into consideration the research questions that must be addressed.

1. Conduct a detailed literature review on the state-of-the-art for robotic task and motion planning, covering both optimal planning and adaptive/dynamic planning methods, and identify the limitations of these work in relation to the optimal and dynamic planning problems studied in this research.

2. Conduct a study on the challenges of solving the dynamic motion planning problem to identify the considerations that must be accounted for when extending an offline planning problem to online planning.

3. Identify planning problems to serve as use cases for the study of optimal and dynamic task planning and develop the methods for evaluation of algorithms according to solution quality and planning efficiency.

4. Develop techniques that advance the state-of-the-art in task and motion planning towards online planning by reducing the computation time required to obtain a high-quality solution to the static variant of the planning problems identified in Objective 3.

5. Extend the techniques developed in Objective 4 into adaptive algorithms that adequately cope with dynamic planning problems involving changes in the observation of the environment.

6. Evaluate the adaptive algorithms developed in Objective 5 to test the research hypothesis introduced above and assess the deployability of algorithms on physical robots.

1.3 Research Methodology

The methodology adopted in this research comprises of investigating two integrated task and motion planning problems in dynamic environments. Since different types of robots often introduce unique considerations during planning due to the particular characteristics of the robot and problem, this research widens its considerations to two different robotic planning domains with the aim of identifying more common features of dynamic planning problems as well as general techniques that can be applied across different domains to solve such problems. The two planning problems identified for this research include task planning for mobile wheeled robots and task sequencing for industrial robotic manipulators. These were chosen based on the differences in the number of robot DoFs, the differences in the typical number of goals in the planning problems and, from a practicality point of view, the availability of physical robots that can be used for experimental evaluations.

These investigations have been broken down into a number of smaller studies that individually address one or more of the research questions introduced in Section 1.2.3, as described below.

Finally, I note that different computing machines are used throughout this research for the experimental evaluation of algorithms, as reported throughout the chapters of this thesis. Where results are compared across different methods, the same machine is used to ensure a fair comparison. These comparisons are self-contained within each chapter, while results reported across different chapters are not intended for cross-comparison. Thus the use of different machines do not influence the validity of results and the findings thereof.

1.3.1 Literature Review

The literature review seeks to provide a critical analysis of the state-of-the-art in robotic task and motion planning. Though the research in these areas are extensive, this literature review narrows down on existing work that particularly

addresses the problems of optimal planning and adaptive/dynamic planning. I include literature that solely addresses task or motion planning as an independent problem as well as literature that study these as an integrated planning problem (i.e. a CTMP problem). Furthermore, the literature review explores the relative advantages between search-based algorithms and learning-based approaches.

The purpose of this study is to summarise the limitations of existing state-of-the-art methods for solving the problem of optimal and adaptive task planning in dynamic environments and to identify the key knowledge gaps in literature, which have subsequently been addressed in this research.

1.3.2 Case Study on Dynamic Motion Planning

A case study is conducted to explore the challenges and key considerations for optimal planning in dynamic environments. Given that optimal and dynamic motion planning is a comparatively less complex problem compared to optimal and dynamic task planning (which remains a largely unsolved problem), I choose to approach this case study from the perspective of motion planning. A dynamic pick-and-place task for an industrial manipulator is chosen as an application example for the problem of dynamic motion planning as it provides the opportunity to explore the complexity of non-trivial state spaces due to the robot's high number of DoFs. I stress, however, that pick-and-place is not one of the primary planning problems examined in the rest of this thesis.

This case study will particularly seek to address Research Question 1 (see Section 1.2.3) and findings will be discussed in the wider context of general robot domains. As we will see, the development of algorithms for solving each of the two integrated task and path planning problems studied in this thesis are driven by the findings in this study.

1.3.3 Task Planning for Mobile Wheeled Robots

The first integrated task and motion planning problem studied in this thesis addresses a general task planning problem for a mobile wheeled robot, where the robot's state space is defined by its x and y position in the world (i.e. a 2-dimensional configuration space).

The attention given to this problem is two-fold. In the first instance, the problem is treated as a static optimisation problem were dynamic changes are

not considered. Drawing from the findings of the case study on dynamic motion planning, this first part of the investigation addresses the planning problem from the direction of advancing the capabilities of motion planning. Consequently, a new algorithm for motion planning between multiple goals is developed to efficiently compute the cost of low-level motions necessary for optimal task planning. I explore the benefits of this algorithm for solving optimal task planning problems by integrating it with off-the-shelf task planners and comparing its performance with alternative methods as a baseline. To this end, this segment of work sets out to partially address Research Questions 2 and 3 by examining how the efficiency of motion planning can be improved and the effect this has on solution quality. I also assess how decisions made at the task planning layer can affect the performance of the integrated planner in terms of solution quality and planning efficiency.

The second part of this investigation extends my considerations to the dynamic variant of the planning problem, where we turn our attention to the higher level task and motion planning architecture. A number of techniques are explored to determine how the integrated planner can be extended into a framework for online planning. A key objective is to enable the online adaptation of both low-level motions and high-level task plans to maintain optimality and avoid collisions when new obstacles in the environment are observed. Since mobile robots must generally rely upon on-board computing power to run the entire system, particular attention is devoted here to ensure that the developed algorithms can be deployed efficiently to robots with limited memory capacity. Thus this latter part of the investigation addresses Research Questions 2-4 for mobile wheeled robots. The framework is evaluated on a physical differential drive mobile robot using a comparatively light-weight computing machine to assess the performance of the algorithm for a typical task planning problem.

1.3.4 Task Sequencing for Robotic Manipulators

The second integrated planning problem studied in this thesis comprises of a task sequencing problem specific to high DoF serial-link manipulators. This problem is notably more complex than the problem of task planning for mobile wheeled robots for two reasons: Firstly, the high DoF nature of these robots introduce the phenomena of *kinematic redundancy*, where a target position in the Euclidean

space maps to a **set** of points in the robot's configuration space. Secondly, the number of goals in a typical task sequencing problem is often in the order of hundreds, which is manifolds greater than the number of goals commonly encountered in the mobile wheeled robot domain.

These two features can lead to a combinatorial explosion for a large number of goals and high kinematic redundancy when solved using brute force methods. Thus it is impractical to adopt the previous bottom-up strategy of solving the motion planning problem for all possible actions and subsequently computing the optimal task-level plan as this can quickly become intractable. Instead, the problem is addressed using a top-down approach, where motion planning queries are performed **after** a task sequence is determined.

Like before, this problem is investigated in two parts. The first part of this work is devoted to the static optimisation problem where the poor planning performance of existing methods is addressed for particularly challenging task sequencing problems involving hard spatial constraints. New techniques are proposed to advance the state-of-the-art such that complex problems involving spatial constraints can be solved more efficiently while providing high-quality solutions. I go on to identify the necessary considerations for the deployment of the algorithm to physical systems, taking into account different application requirements. Thus the developments described here specifically address Research Questions 2-4 from a static planning perspective for robotic manipulators. The resulting algorithm is benchmarked against existing methods to quantify and assess the performance of the proposed method.

The latter half of this study investigates the dynamic variant of the task sequencing problem and seeks to extend the previously developed algorithm to enable adaptive planning. More specifically, this segment of work examines how the computation time required to generate a set of executable actions can be drastically reduced to meet the stringent requirements of online planning while satisfying the goals of the sequencing problem. I go on to identify techniques that enable fast re-planning of task sequences and individual motions to preserve high-quality solutions and avoid collision within a dynamically-changing environment. The outputs of this work are two new variants of the algorithm previously developed for static planning. Their performances are evaluated through simulation-based experiments and compared against the benchmarked performance of the original algorithm. Findings are discussed in the wider context of

adaptive planning, providing additional insight into the practical considerations for implementation on physical robots. Collectively this work addresses Research Questions 1-4 from a dynamic planning perspective for robotic manipulators.

1.4 Thesis Organisation

This thesis is composed of eight chapters, including this introductory chapter.

Chapter 2 is dedicated to the literature review and covers the state-of-the-art in task and motion planning within the context of optimal planning and adaptive planning. The limitations of the current state-of-the-art are identified and the knowledge gaps are discussed within the scope of the contributions of this thesis.

Chapter 3 presents the case study on the dynamic motion planning implemented on a large-scale industrial robot for a dynamic pick-and-place task. I discuss the challenges of enabling adaptive robot behaviour in low-level motion planning and its general implications to re-planning in robotics. This chapter also touches on the integration between the areas of planning, perception and control and their requirements for flexible online adaptation.

The next four chapters provide the main contribution of this thesis. **Chapter 4** introduces the multi-tree-based motion planning algorithm that efficiently computes all optimal paths between multiple goals. I show that the algorithm can be applied to general cost spaces and demonstrate the use of the algorithm for solving task planning problems in the mobile robot domain.

Chapter 5, addresses the problem of high- and low-level re-planning in mobile robots for partially-known and dynamic environments. I present an adaptive task and path planning framework that adopts and extends the algorithm presented in Chapter 4 to efficiently cope with new environmental observations, and show that the framework can support algorithmic *anytime*-like behaviour.

Chapter 6 shifts the focus to fixed-based manipulators, where I address robotic task sequencing problems (RTSP) for spatially-constrained applications. I present a novel clustering-based method that exploits the *kinematic redundancy* of serial-chain manipulators to quickly find higher quality solutions compared to existing state-of-the-art approaches.

Chapter 7 describes a preliminary study on the problem of dynamic task sequencing, where two new variants of the algorithm introduced in Chapter 6 are presented. In particular, this chapter introduces the concepts of partial planning

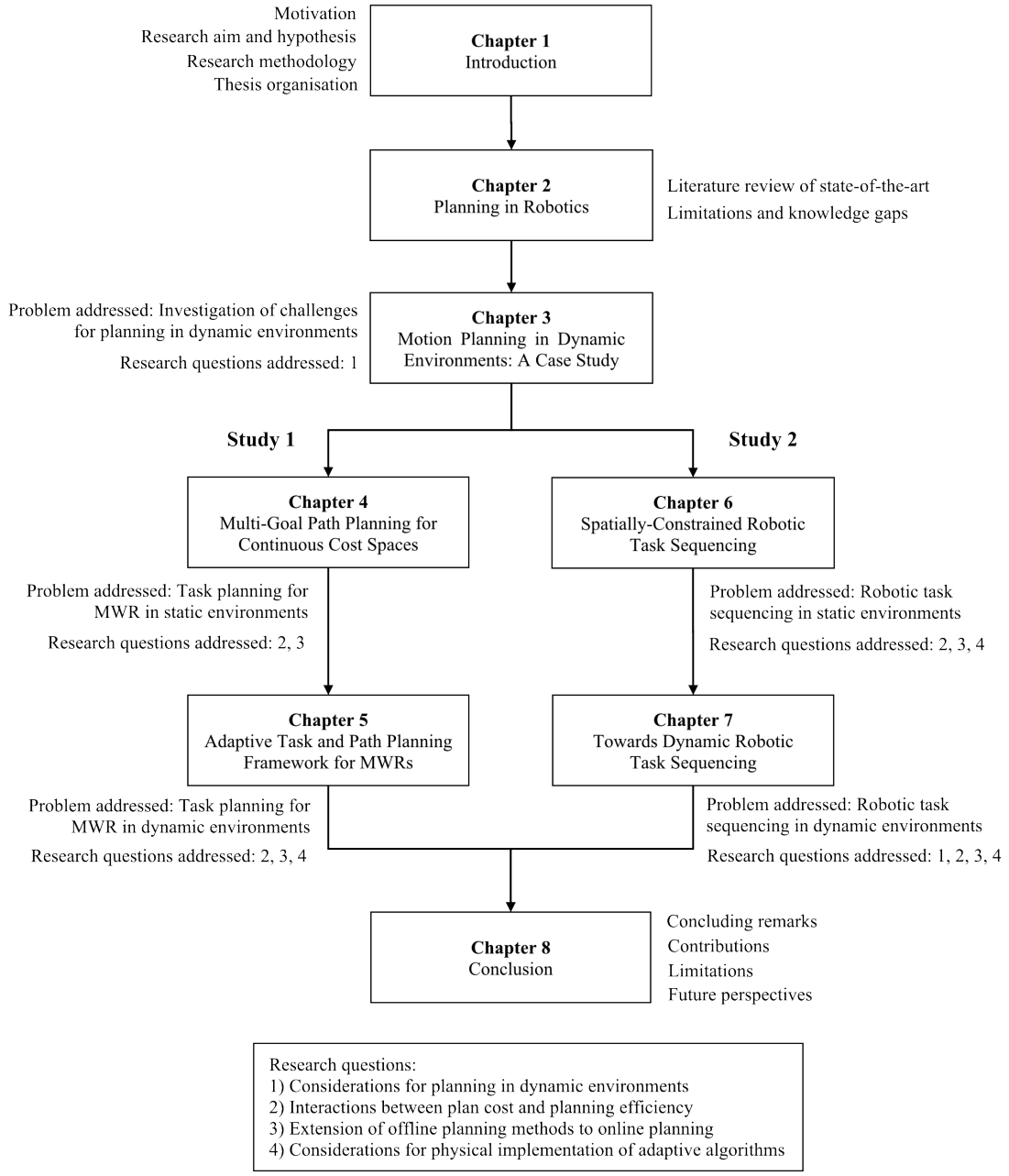


Figure 1.1: Organisation of the thesis, indicating the problems investigated in each chapter and the corresponding research questions that are addressed.

and dynamic re-planning to RTSPs and show how they extend the capabilities of the original algorithm to achieve adaptive task sequencing for dynamic environments.

Finally, **Chapter 8** concludes this thesis with a collective discussion of the research findings derived from each of the studies in relation to the overall research

aim and objectives. This is followed by a summary of the individual contributions to knowledge made in this thesis. Limitations of the reported work are described, followed by a discussion of possible directions for future work.

The organisation of this thesis is summarised in Fig. 1.1, where the focus of each chapter and the underlying research questions addressed are shown on a chapter-by-chapter basis.

Chapter 2

Planning in Robotics

This chapter provides a literature review of task planning and motion planning, where I cover the progression of successful planning techniques and discuss the current state-of-the-art. The chapter is comprised of two sections covering motion planning and task planning methods, respectively. I focus particularly on optimal planning methods and adaptive approaches, and provide a summary of reviewed literature to highlight their limitations, the current knowledge gaps and how these are addressed in this thesis.

I note that the bodies of literature for these areas are vast, and the purpose of this chapter is not to exhaustively review every major contribution made to date. Instead, I aim to provide readers with the necessary knowledge to understand and appreciate the significance of the contributions made in this thesis in relation to the broader literature.

2.1 Motion Planning

Motion planning is a fundamental planning problem that commonly exists in domains consisting of a moving agent. A classic example is the *Piano Mover's Problem*, whereby a piano located inside a house must be moved from one room to another without collision [18]. The essence of this problem remains the same for all motion planning problems. Simply put, the objective is to find a valid, collision-free motion between a *start configuration* and a *goal configuration* subject to environmentally-imposed constraints (e.g. no-go regions, one-way roads, obstacle regions) and body constraints (such as nonholonomic constraints, joint limits, self-collision etc). These problems exist not only in robotics, but have also

been particularly common in the context of gaming.

In this thesis, I narrow down on the *path planning* sub-problem, where we are interested only in determining the geometric path that connects the start and goal configurations while satisfying domain-specific constraints. In simple problems involving agents that can be assumed to be a point body, motion planning can be solved in the Cartesian (or x-y-z) space, which is generally most intuitive to humans. This assumption can generally be adopted for gaming applications. However, for agents in the real-world, it is necessary to take into consideration the space of valid configurations. For example, a bicycle or 4-wheel car is incapable of rotating on the spot due to the nature of the rolling wheel. Similarly, robots of different forms possess different state transition constraints that must be adhered to.

The entire space of possible state transitions is described by the *configuration space* (C-space). The C-space is generally composed of the DoFs of the robot and is specific to each robot. Consider a serial manipulator such as a conventional industrial robot with 6 DoFs, where each joint is a revolute joint. The C-space for such a robot would consist of 6 dimensions, associated to the axial angles of each joint constrained within the robot's joint limits. A **configuration** corresponds to a single point in this C-space and fully describes the state of a robot at any given time.

Planning in the C-space can provide a number of key advantages depending on the application. First of all, any continuous path in C-space guarantees smooth motion when executed as kinematic constraints of robots are adhered to in the C-space. This can ensure that *singularities* and *poor manipulability* are avoided. When a manipulator reaches a singularity, any further desired movement in a certain direction would lead to large changes in joint angles (often 180 degree changes) instantaneously, which is beyond the physical capabilities of the robot. Manipulability describes how close a manipulator configuration is to a singularity. As robots move into configurations that lie close to a singularity, the robot suffers from poor manoeuvrability and fail to move at a given velocity at the end effector. [19, 20] This phenomena often arises when transforming a Cartesian space trajectory for a manipulator end effector to joint motor commands. By planning in C-space, this Cartesian space to joint space transformation is not required. Secondly, because any continuous path in C-space can be directly executed on the physical robot, any path planning algorithm that can be applied to

one robot in the C-space can be generalised to any C-space without modification. This is not equally true for Cartesian space planning. Finally, solutions to path planning problems in C-space provide more information about *how* a robot would move. This is because any point in the C-space completely describes the entire robot configuration. Thus a continuous path would describe the robot's complete pose at any point along the path. Conversely, if we consider path planning for a serial manipulator in Cartesian space, we could only represent the trajectory of the end effector (or any single point on the robot) at any point along the path. This is important as many robots cannot be represented as a point model for *collision-free* path planning. Collision checking requires information about the entire robot geometry that could only be derived from the complete configuration information.

Despite these advantages, planning in C-space does come with a major shortcoming: it is comparatively difficult to represent external world objects in C-space. Not only is it easier for humans to understand geometric knowledge in Cartesian space, perceptive sensors universally sense objects in X-Y-Z coordinates. A means for mapping objects in Cartesian space to C-space is therefore necessary if we wish to represent obstacles in C-space, which, unfortunately, is not a trivial process. Numerous works have presented strategies to perform this transformation, including [21–23]. However, generally speaking it is a computationally expensive procedure and often difficult to scale for high dimensions, which makes it particularly difficult to achieve real-time path planning.

In the following, I review the body of literature on motion planning for computing a single path between a start and goal configuration. These work are broadly grouped according to the following categories: deterministic methods, sampling-based methods and machine learning-based methods.

2.1.1 Deterministic Methods

All deterministic path planning methods share two common traits. Firstly, they are *exact*, and thus will always give the same solution to a path planning problem each time. Secondly, they are either *complete* such that they will always find a solution if one exists, or they are *resolution complete*, meaning they will always find a solution for a problem at the resolution considered if it exists at that resolution level. The methods presented herein are well established, some dating

back as far as 1950s and have all been proven in 2D applications. However, in this section I present these methods in the context of C-space planning, which highlights the deficiencies of these methods for high-dimensional path planning in robotics.

One of the classic approaches to 2-dimensional path planning is cell decomposition, which consists of decomposing an environment's free space into smaller regions (referred to as cells) by using either *exact* or *approximate* cell decomposition. A connectivity graph (or roadmap) is generated based on the adjacency of the cells, where each node is a cell and an edge between two nodes indicate that they are adjacent. The start and goal cells are assigned by determining the cells that contain the start and end configurations of the path planning problem. A graph search algorithm can then be applied to the connectivity graph to find the shortest route between the start and goal cells.

In exact cell decomposition [24], each vertex of the interior polygons of the configuration space is used to produce a vertical segmentation towards the exterior boundary of the space. Once a path through the connectivity graph is obtained, an actual path in free space can be determined by connecting the start point to the mid-point of each cell intersection and finally to the goal point by following the path sequence previously obtained (see Fig. 2.1a).

Approximate cell decomposition [25] differs in the process of subdividing the cells. Implementing the exact cell decomposition can be mathematically challenging in practice. Approximate cell decomposition resolves this by recursively subdividing the search space into 4 equal cells until (i) all cells are either completely contained within an obstacle or is completely free space, or (ii) a pre-defined resolution is reached (see Fig. 2.1b).

After the decomposition ends, a path is obtained in the same way as exact cell decomposition. A key difference in performance between exact and approximate cell decomposition lies in their completeness. Exact cell decomposition is guaranteed to be complete, while approximate cell decomposition is resolute complete.

The visibility graph method [26, 27] also consists of generating a roadmap, but unlike the cell decomposition approach, the edges of the roadmap represent an actual segment of a path. The visibility graph is constructed by generating nodes from the vertices of all polygonal obstacles, and edges are drawn where any two vertices connect to each other by a straight segment without intersecting

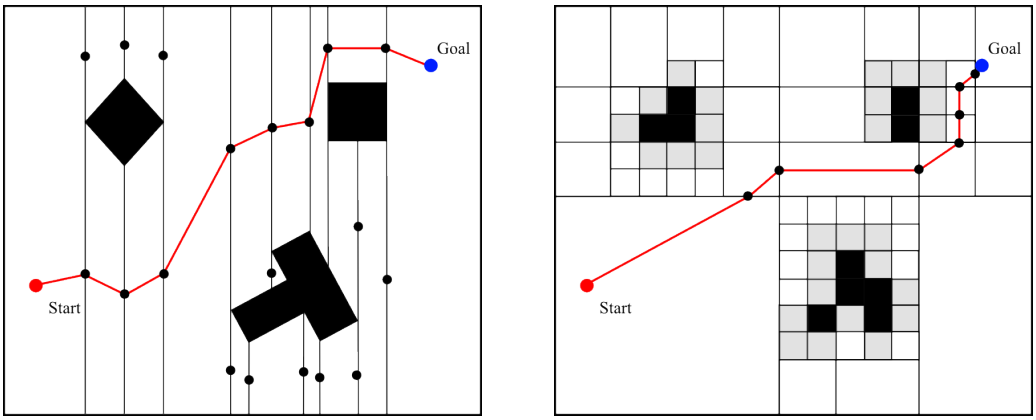


Figure 2.1: An example of paths (shown in red) obtained by the cell decomposition approach for 2D path planning among obstacles (shown in black), (a) exact cell decomposition, where vertical lines represent segmentation boundaries, (b) approximate cell decomposition, where black cells are fully occupied by obstacles and grey cells represent partial occupancy.

any obstacle regions. The start and goal points are also added as nodes of the graph. Once the graph is constructed, a shortest path search algorithm may be applied to find the shortest path to traverse the graph from the start node to the goal node (an example is shown in Fig. 2.2). For obstacles with no vertices, such as circular regions in 2D space, tangents of the surface may be used to generate edges in the roadmap [28]. Indeed in this way the final path could be composed of straight and curved segments.

While this method is conceptually simple, constructing the visibility graph has been the focus of much research, specifically with the goal of reducing the complexity of the graph construction step. Using a brute force approach to evaluate every possible connection for n obstacle vertices in a 2D environment would require $\mathcal{O}(n^3)$ time (every vertex must be checked with $n-1$ other vertices, and each corresponding edge must be checked against n polygonal edges). Various algorithms have been proposed to improve the complexity of computing visibility graphs. For example, [29] proposed two alternative methods that runs in $\mathcal{O}(n^2)$ and $\mathcal{O}(m \log n)$ time, respectively.

The visibility graphs method also carries two major shortcomings. As the graph is computed from obstacle vertices, the shortest path found will always approach very closely to obstacles, providing little tolerance for collision-free motion. Unfortunately in the physical world where certainty and errors in path tracking exist, such tight tolerances are generally inadequate for guaranteeing

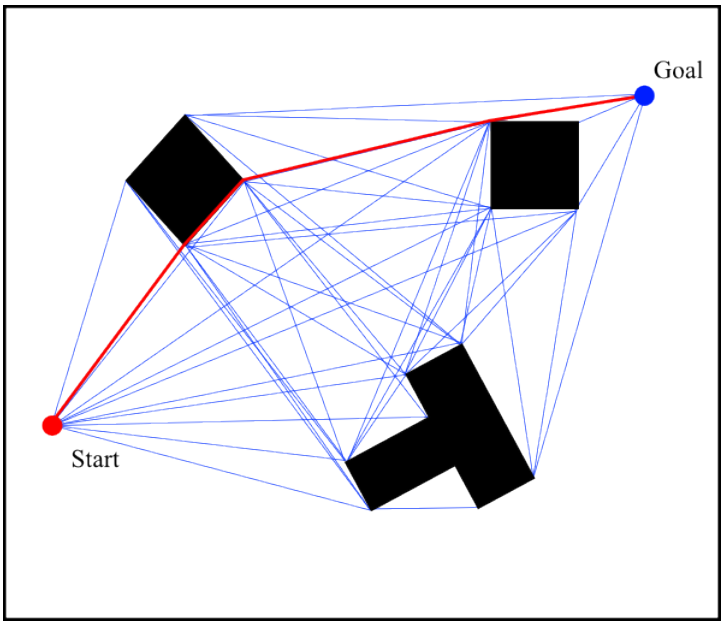


Figure 2.2: The visibility graph path planning method, where blue lines indicate a direct line-of-sight between two vertices that form a valid path segment and the red line shows the final path solution.

safe movement of the robot. Furthermore, the algorithm does not scale particularly well with the number of dimensions and have mostly been applied to 2D applications. Even now its extension into 3D applications is an area of ongoing research [30].

Another branch of methods adopt the concept of *occupancy grid mapping* for the representation of an environment and solves the path planning problem using graph-search methods. [31] This mapping discretises the environment into an evenly spaced grid, where each cell in the grid stores a binary value to represent the free and occupied space. For a basic occupancy grid, a value of 0 is used to represent free space, while a value of 1 indicates that the cell contains an obstacle (an example can be found in Fig. 2.3). The extended *probabilistic occupancy grid* replaces this with finer representation by assigning a probability value to each cell for the likelihood of an obstacle belonging in each cell.

The Dijkstra's algorithm [32] was a pioneering algorithm developed by Edsger W. Dijkstra in 1956 for finding shortest paths among nodes in a graph. In fact it was commonly used to solve the search for the shortest path in a constructed visibility graph and in the cell decomposition methods and have found extensive use in applications far beyond the area of path planning.

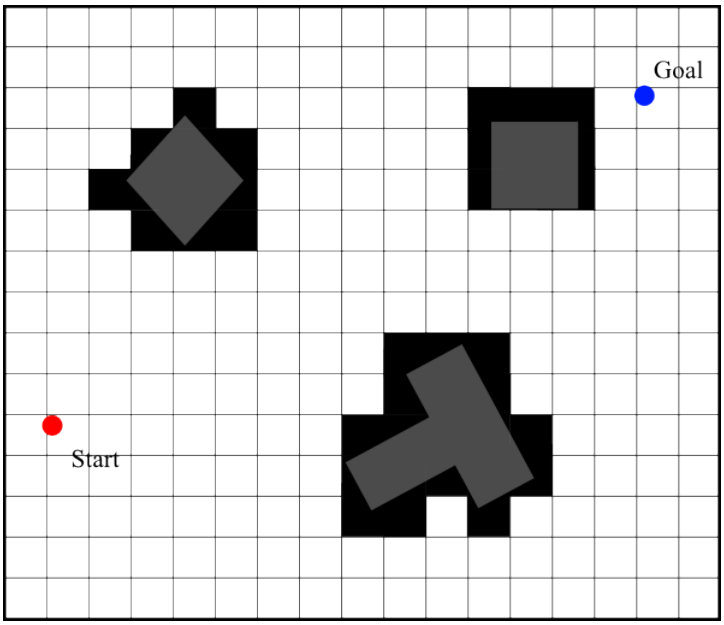


Figure 2.3: A simple 2D example of a binary occupancy grid, where black is used to represent cells occupied by the obstacles (in grey).

Dijkstra's algorithm is composed of the following steps. A set of unvisited nodes is created from all the nodes in the graph and each node is assigned an initial distance value. This value corresponds to the distance of the node from the starting node, n_s (determined by a distance function such as the Euclidean or Manhattan distance) and is initialised to zero for n_s and infinity for all others. Beginning with n_s, the algorithm recursively removes a node with the lowest distance value from the unvisited list and expands it to compute new *tentative* distance values for the neighbours of the expanded node by summing the distance between the two nodes and the expanded node distance from n_s. In the case that the tentative distance is smaller than the neighbour's current distance, the neighbour node's distance value is updated. For example, if A is n_s with distance value of zero, B is a neighbour node of A with a distance of 5 from A, and C is a neighbour node of B with a distance of 1 from B, then the distance value of C from A is $5 + 1 = 6$. Following this procedure, the algorithm continues to expand nodes until the goal node is found as a neighbour of an expanded node or all nodes in the set of unvisited nodes has a distance value of infinity (indicating that there are no other nodes in the unvisited list that is connected to n_s). In this way the algorithm searches through nodes with the shortest distance until the goal node is found. During the expansion, the predecessor node (or the *parent*

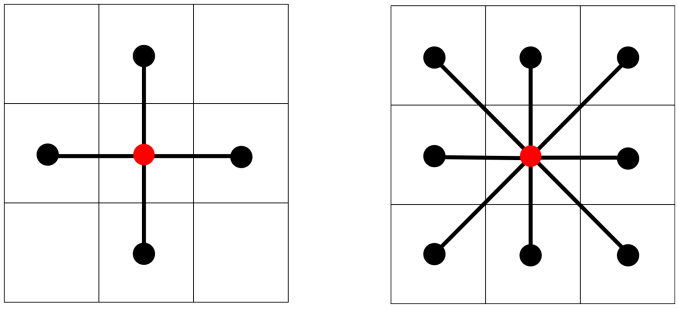


Figure 2.4: Cell connectivity in an occupancy grid (a) 4-point connectivity, (b) 8-point connectivity.

node) that led to the shortest distance value for a given node is kept tracked of. When the goal node is found, the complete path can be obtained by backtracking through the parent nodes starting from the goal node until the starting node is reached.

When applied to occupancy grids, Dijkstra's algorithm effectively treats the grid as a connectivity graph, where cells adjacent to each other in the grid is considered neighbours. Depending on the application, either 4-point connectivity or 8-point connectivity is used to expand nodes. In 4-point connectivity, nodes are considered a neighbour of an expanded node if it lies horizontally next to or vertically above/below the expanded node. In 8-point connectivity, all nodes that satisfy 4-point connectivity or lie diagonally to the expanded node are considered neighbours (these are depicted in Fig. 2.4).

One shortcoming of Dijkstra's algorithm is the lack of guidance towards to the goal during node expansion. While searching through the shortest path nodes guarantees that the optimal solution will be found (within the resolution of the occupancy grid), much computation is spent on expanding "useless" nodes - those which do not lead towards the goal node. This was addressed with the emergence of the A* algorithm.

The popular A* algorithm [33] was developed for graph traversal and highly resembles the Dijkstra's algorithm, albeit requiring less node expansions to find the optimal solution. It follows the same steps as Dijkstra's algorithm for node expansion, but replaces the distance function with a new cost function $f(n)$ for an arbitrary node n. It is composed of a cost from the starting node $g(n)$ (equivalent

to the distance function in Dijkstra's algorithm) and a heuristic function $h(n)$ that estimates the cost of n to the goal:

$$f(n) = g(n) + h(n) \tag{2.1}$$

This heuristic function provides the algorithm with the characteristics of *informed search*, such that node expansion prioritises those nodes that are not only the shortest distance away from the starting node, but would also lead to the goal node via the shortest path. In this way A* is guaranteed to find the optimal solution under the condition that the heuristic function is *admissible*, meaning it never overestimates the true cost of n to the goal node. In 2D path planning problems, the Euclidean distance function often serves as a simple but effective heuristic function for the A*.

While both Dijkstra's algorithm and A* guarantee an optimal solution, the degree of optimality is strictly limited by the resolution of the occupancy grid. This is due to the constraint of advancing between cells along the horizontal, vertical and 45° directions. Seeking to address this, Daniel et al. [34] proposed the basic Theta*, which enabled any-angle path planning on occupancy grids. Theta* consists of a single modification to the A*: when calculating the tentative cost function $f(n)$ for node n, Theta* considers two candidate paths as opposed to one. The first path corresponds to the path from the expanded node to node n in the same way as A* does by default. The second path, however, considers the path from the *parent* of the expanded node to node n. The path that provides a lower cost function value is used to determine the parent of n, if the tentative cost is accepted. As a result, the solution path can consist of path segments of any angles and in general are shorter than solutions obtained by A*. However, it is important to recognise that this does not completely eliminate the correlation between resolution and optimality, as points of angle change along the path must still lie on the corners (or centroids, depending on the implementation of the algorithm) of the cells in the occupancy grid.

In terms of performance, all of the aforementioned methods for path planning using occupancy grids are resolution complete. Furthermore, they possess similar memory complexity as all algorithms store every found node in memory. In the worst case, the memory complexity is $\mathcal{O}(b^d)$, where b is the branching factor (the average number of neighbour nodes for each node expansion) and d is the depth

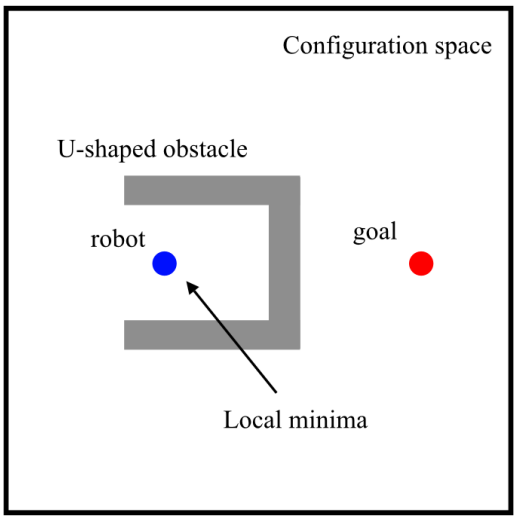


Figure 2.5: Local minima problem in the APF path planning method. The robot is attracted towards the goal but the repulsive potential from the U-shaped obstacle traps the robot such that movement in any direction would lead to an increase in energy.

of the solution (or the nodal length of the shortest path). The worst-case time complexity of these algorithms is also $\mathcal{O}(b^d)$. However in practice, A* and Theta* generally outperforms this as the heuristic function allows the pruning of many branches by informed search.

Unlike all of the aforementioned methods that consist of generating a graph representation of the environment and subsequently applying a search, the Artificial Potential Field (APF) [35] is a reactive path planning method where an explicit path is not directly computed. Rather, the movement of the agent is influenced by the total effect of a potential *energy* function generated by the obstacles in the environment and the goal point. Obstacles produce a *repulsive* potential, while the goal point produces an *attractive* potential. In this way an artificial vector field representing the net potential at each point in the search space is obtained. The path taken by the robot then follows the negative gradient of the potential field such that energy is minimised. Thus the movement of the robot by APF highly resembles the behaviour of gradient descent, and therefore suffers from local minima problems. Unfortunately this occurs rather commonly in path planning problems. A simple example involving a U-shaped obstacle is shown in Fig. 2.5.

The Bug family of algorithms, developed for mobile robots, also adopt a reactive path planning strategy. Bug2 [36] was the first practical algorithm within

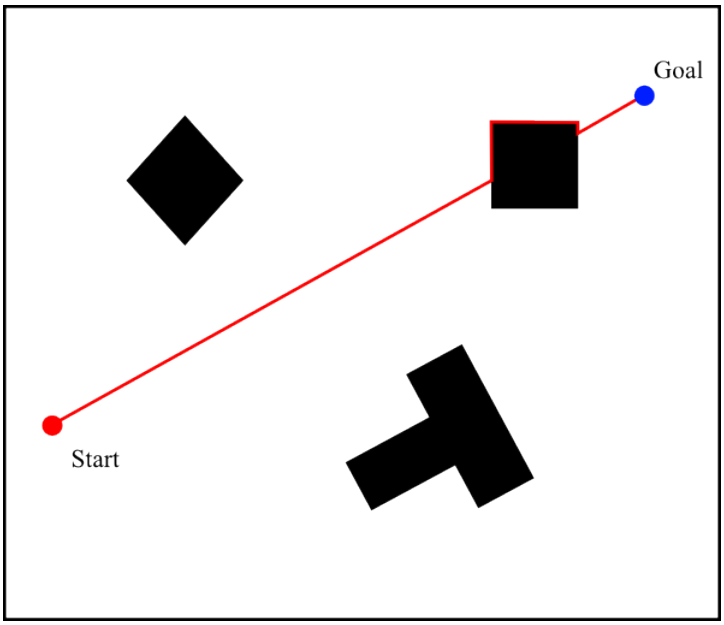


Figure 2.6: An example path obtained by the Bug2 algorithm.

this family whereby the robot seeks to follow a direct line from the start to the goal point. When this line encounters an obstacle, the algorithm redirects the robot to follow the contour of the obstacle until it returns to the original line of sight to the goal on the other side of the obstacle (see 2.6). The resulting path is almost certainly sub-optimal as the robot would follow the boundary of the obstacle further than necessary to reach the goal. This was addressed by the VisBug algorithm [37], which differs from Bug2 in the leaving condition used to stop contouring an obstacle. Instead of requiring the robot to reach the line of sight, the algorithm reattempts to send the robot directly to the goal once it can "see" the original line of sight. Indeed this resulted in shorter paths in all instances.

Despite this improvement, the VisBug algorithm could still fail to find optimal paths depending on the shape of the obstacle. Kamon and Rivlin improved upon this algorithm with DistBug [38], where two new rules were introduced to induce an earlier leave condition taking into consideration multiple obstacles. Rather than seeking to find the original line of sight, the DistBug numerically compares the distance between the current robot location to the next obstacle in the direction of the goal and the original distance to the goal at the first point where collision wasdetected. This enabled the robot to navigate past obstacles with far less contouring than with VisBug and Bug2. Finally, the K-Bug algorithm [39] was introduced by Langer et al. to further improve the quality of solutions obtained,

particularly in environments involving multiple obstacles. Rather than seeking to travel along a direct line from the current robot position to the goal, the algorithm seeks to visit visible vertices of obstacles when obstruction is encountered, choosing the direction (left or right) based on the nearest vertex. Only when no further obstructions are observed does the algorithm resume a direct path to the goal position.

This bug family of algorithms possess an interesting set of advantages and drawbacks. First of all, their reactive behaviour to obstacles in the environment enable to algorithm to adapt to both unknown and dynamic environments as they do not involve explicit planning at a global level. While APF methods share a similar advantage, bug algorithms do not suffer from being trapped in local minima as they always seek to navigate past obstacles by following its contour. The simpleness of these algorithms also mean they are fast and capable of responding quickly to perceived obstacles in dynamic applications. However, one inherent drawback of the algorithm being reactive in nature is the inability to account for obstacles before the path intersects an obstacle. For instance, returning to the concave obstacle example in Fig 2.5, the bug algorithms would not begin to contour the obstacle until it has reached the inner wall of the concave. Naturally, this leads to a sub-optimal path as the robot unnecessarily navigates further into the concave obstacle. Furthermore, the natural tendency of these algorithms to follow closely along the boundaries of obstacles mean that any deviation from the nominal path during execution will likely lead to collision. This is a similar problem to the visibility graph that can often limit the practicality of these algorithms for deployment in robots with poor tracking performance.

A common observation for all of the presented deterministic path planning methods is the requirement to have explicit knowledge of the obstacles represented in the search space. As discussed in 2.1, this is convenient for path planning in Cartesian space as obstacle information are generally easily attainable in this spatial representation. However, considering the difficulty of transforming obstacles into C-space representation, many of these algorithms cannot be easily extended to path planning in C-space. This has led to the rise in the popularity of *sampling*-based approaches, which have proven to be effective for robots possessing an arbitrary number of DoFs.

2.1.2 Sampling-Based Methods

Sampling-based algorithms are generally described as being **probabilistically-complete**, meaning that as the run-time tends towards infinity, the algorithm's rate of failure in finding a solution, if it exists, falls to zero. Unlike deterministic methods, which always return the best solution within the resolution considered by the planner, these algorithms are stochastic in nature as they explore the search space through random sampling of points. As a result, different solutions are obtained over multiple runs of an algorithm. Furthermore, these methods are able to guarantee **asymptotically-optimal** solutions at best, meaning that as the number of iterations tends towards infinity, the solution returned by an algorithm converges towards the optimal solution. These two properties imply that the quality of a solution returned depends on the amount of planning time that is allocated to solving a motion planning problem.

A key advantage of sampling-based algorithms over deterministic methods is the removal of any dependency on explicit knowledge of obstacles represented in the C-space. Instead, these algorithms evaluate the validity of individually sampled configurations by performing collision detection queries between the geometric models of the robot at the given configuration and the Cartesian space obstacles within the environment. In the case of serial manipulators, the principle of **forward kinematics** is used to determine the geometric configuration of the robot given the axial angles of each joint defined in the C-space. For practical implementation, the Flexible Collision Library is a widely-adopted open-source library for performing fast collision and proximity queries [40].

The two fundamental algorithms that have widely been considered as the backbone of sampling-based algorithms are the **Probabilistic RoadMaps** (**PRMs**) and the **Rapidly-exploring Random Trees** (**RRTs**). Both methods adopt a random sampling strategy to explore the search space, but identify themselves differently in the way that they connect sampled points.

2.1.2.1 Probabilistic Roadmap Methods

The PRM, proposed by Kavraki et al. [41], seeks to build a roadmap across the search space during the *construction* phase by randomly sampling nodes and accepting those which are collision-free. Each accepted node is then edge-connected to neighbours that either lie within a ball of fixed radius r, or by applying the

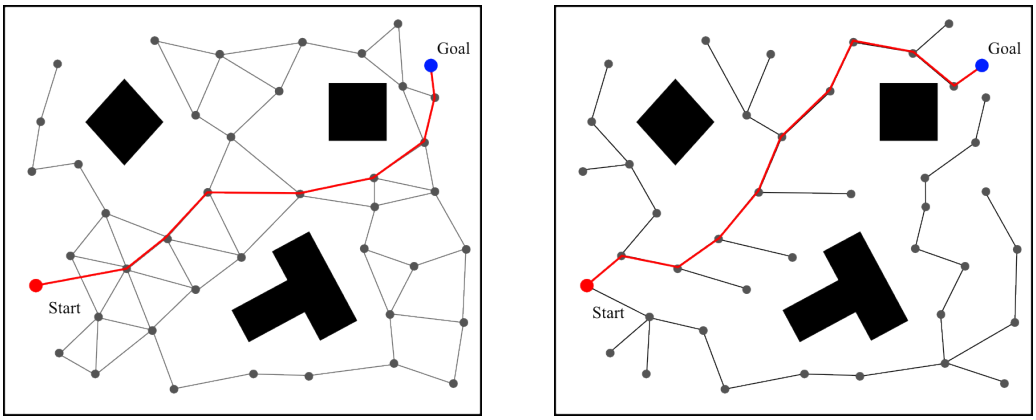


Figure 2.7: (a) A simple path planning problem in 2D environment solved using the PRM algorithm, (b) same problem solved using the RRT algorithm.

k-nearest neighbours selection [42], under the condition that they are not already *graph-connected*. This avoids the formation of cyclic paths and reduces the number of edges stored. The construction phase runs until a given roadmap density criteria is reached (e.g. when the number of nodes reaches a pre-determined value). During the *query* phase, the start and goal configurations for a particular path planning instance are added into the roadmap as nodes and connected to neighbours in a likewise manner to the construction phase. With this complete roadmap, a graph search algorithm such as Dijkstra's algorithm or A* can be applied to obtain the shortest path in the graph. An example of a roadmap and resulting path for a simple 2D path planning problem is shown in Fig. 2.7a.

The PRM algorithm is particularly well suited for multi-query applications as the same constructed roadmap can be used to solve path planning problems with different start and goal points provided the environment remains static and sufficient resources is devoted to the construction of the roadmap such that it adequately explores the entire search space. However, this offline, pre-processing stage is computationally costly for single-query applications as the construction of the roadmap is not influenced by any particular path planning problem. It simply seeks to explore the entirety of the search space without considering the start and goal configurations.

Focusing on multi-query problems, a key limitation of the PRM is its non-optimality characteristics. Due to the condition for edge-connection requiring that neighbour nodes and newly sampled nodes are not already graph-connected, often a detour is necessary to move between two graph-connected nodes that lie

in close proximity via a *central node*. While the motivation for PRM's design was to reduce complexity while encouraging exploration, it meant solution paths were generally longer than necessary. The PRM* algorithm was proposed by Karaman and Frazzoli [43] to address this flaw. The PRM* consisted of two modifications to the original PRM algorithm. Firstly, it removed the condition for edge-connection that prevented an edge to form between nodes already graph-connected. This "simplification" of the original PRM algorithm provided asymptotic optimality as any neighbour node could be connected to a newly added node, but reintroduced the problem of high complexity [44]. To compensate for this, the authors introduced a variable connection radius $r(n)$ for the selection of neighbouring nodes, which decayed at a logarithmic rate to the number of nodes in the roadmap, as shown in Eq. 2.2:

$$r(n) = \gamma \left(\frac{\log(n)}{n} \right)^{1/d} \tag{2.2}$$

where d is the dimension of the search space C, and, letting ζ_d be the volume of a unit ball in d dimensions, γ is given by:

$$\gamma = 2 \left(1 + \frac{1}{d} \right)^{1/d} \left(\frac{\mu(C_{free})}{\zeta_d} \right)^{1/d} \tag{2.3}$$

where $\mu(C_{free})$ is the Lebesgue measure (or volume) of the collision-free space. This new variant of the original PRM algorithm became a powerful multi-query sampling algorithm that, given sufficient offline pre-processing time to construct the roadmap, could obtain a high-quality solution to general path planning problems. However, as previously discussed, this requires that no changes are made to the environment. When applying the PRM* to C-space, it can be computationally intensive to readjust the roadmap in response to changes in the environment. Unfortunately in real-world applications, planning domains are often subject to change. This renders the PRM and PRM* particularly unsuitable for dynamic applications.

The Dynamic RoadMap (DRM) method [45] was proposed to overcome these challenges by planning specifically in the C-space. It similarly consists of a pre-processing stage that involves the construction of a roadmap. However, it does not consider any environment-specific obstacles during this offline phase. Instead, it discretises the robot's geometric workspace into uniform cells (much like an

occupancy grid). The roadmap is constructed either by uniformly or randomly sampling configurations in the C-space and connecting nodes in the same way as PRM or PRM*. Importantly, all nodes are accepted without performing collision checks as obstacles are not considered. Then, a mapping process is performed to map each cell in the discretized workspace to nodes and edges in the roadmap. A cell is mapped to a node or edge if collision would occur at those configurations when an obstacle is present in that cell. Given this mapping, the roadmap can be dynamically adjusted by removing nodes and edges that become invalidated due to changes in the geometric environment as long as the cells that become occupied by obstacles can be identified (which is generally an easy problem as both cells and obstacles are represented in Cartesian space). During the online planning phase, the start and goal configurations are connected to the roadmap, which is trimmed according to the obstacles in the environment.

Experimental investigations have shown that this method is capable of enabling dynamic re-planning, where paths are re-planned online in response to dynamically-moving obstacles [46]. Furthermore, it would only be necessary to compute the pre-processing stage once for a particular robot make and model as it remains correct regardless of the environment it is deployed in. However, the algorithm also possesses a number of shortcomings, including resolution problems leading to reduced optimality due to workspace discretization, very expensive offline computations and high memory requirements as the entire mapping between cells, nodes and edges must be stored. These are explored in further detail in Chapter 3.

2.1.2.2 Rapidly-exploring Random Tree Methods

The RRT algorithm [47] and its variants differentiate themselves from the PRM-based methods by predominantly being a single-query path planner with strong exploration capabilities and efficiency in solving high-dimensional problems.

As the name suggests, the basic RRT seeks to explore the search space by growing a tree from a root node q_s, taken as the start configuration of the path planning problem. The algorithm iteratively samples a random configuration q_{rand}. The nearest node in the tree to q_{rand} is chosen and steered towards the sampled configuration by an incremental distance Δq to generate a new candidate node q_{new}. The candidate node is tested for collision and, if accepted, an attempt to add the node to the tree is made. The nearest neighbouring node that

provides a collision-free edge to q_{new} from among all nodes within the current tree is then assigned as the parent node of q_{new} to connect q_{new} to the tree. The algorithm iterates through these steps until a node that lies within a pre-determined distance from the goal configuration is added to the tree, or when an alternative termination criteria is met (e.g. when N nodes have been added to the tree or a specified computation time has been exceeded). The path from the start configuration to the goal configuration is subsequently obtained by following the parent nodes of the tree beginning from the node that lies in the goal region.

The transition-based RRT (T-RRT) [48] considers continuous cost spaces during sampling through a filtering procedure by performing a transition test. In this procedure, newly sampled points are immediately rejected if their cost is greater than a defined maximum threshold. Points that satisfy this criteria is compared with the cost of the parent node and accepted based on the Bolzmann probability. In this way, the algorithm converges towards lower cost paths. The optimal RRT algorithm (RRT*) [43] likewise improves the solution quality of the original RRT algorithm by introducing a rewiring function. Firstly, rather than assigning the nearest neighbour as the parent of a newly added node, the best neighbour node (the point with the shortest path length) within a neighbourhood region of the sampled configuration is chosen as its parent. Then, the parent of neighbouring nodes are reassigned to this newly added node q_{new} if the path quality to these nodes improve by going through q_{new}. The modification provides the algorithm with an additional asymptotic optimality guarantee at the cost of longer computation times (resulting from the use of the rewiring function). In [49] the strengths of the T-RRT and RRT* algorithms were combined such that solutions were asymptotically optimal in relation to the continuous cost space considered by the transition test.

RRT-connect [50] sought to address the slow convergence rate of the RRT algorithm for complex problems. Rather than growing a single tree to explore the configuration space, two trees are grown simultaneously, with the second tree rooted at the goal configuration of the planning problem. As these trees are expanded, attempts are made to connect the trees by extending towards each other using a simple greedy heuristic. This was later combined with the concepts of RRT* [51], proposed by Jordan et al., to improve the optimality of solutions. The authors also presented additional modifications that improved the planning efficiency of the algorithm. Other authors have extended the work in [50] to

consider the use of several RRT trees for applications such as inspection [52]. For example, the authors of [53] presented a multi-tree implementation of the T-RRT (multi-T-RRT) algorithm for complex planning problems involving multiple waypoints. The CFOREST parallelization framework [54] introduced the use of multiple RRT* trees *in parallel*, where each tree was grown simultaneously to solve the same planning problem between a start and goal configuration. These parallel trees shared the best path found with all other trees to enable all trees to bias growth towards regions with a high potential of improving the current solution. Each time a better solution was found, *tree pruning* was used to remove the nodes in every tree that, even at best, did not provide paths that would improve the solution any further. This parallelization enabled significantly faster convergence towards high quality solutions but, from a practical perspective, its implementation required a computer system with multiple CPU cores. These physical requirements can be demanding for lightweight robots.

In another direction, a number of extensions have been proposed to enable RRT re-planning in dynamic environments by reusing the information available from an initial tree. The execution extended RRT (ERRT) [55] introduced the waypoint cache concept. When a path becomes invalidated due to a new obstruction, nodes along this path are inserted into a waypoint cache. A new tree is then grown using the original RRT algorithm with one modification: during node sampling, there is some probability that nodes are drawn from the waypoint cache rather than from random generation. Consequently, the algorithm reuses information from the previous planning query to guide the tree expansion process. On the other hand, the dynamic RRT (DRRT) algorithm [56] adapts the tree to dynamic obstacles by trimming disconnected branches and continuing to expand the remaining tree until a new path to the goal is found. A further extension of this algorithm consists of biasing sampling towards regions where dynamic changes took place to recover a previously feasible path. Following on from the direction of the ERRT and DRRT, the multipartite RRT (MP-RRT) [57] attempts to preserve disconnected branches of the tree by inserting the roots of disconnected subtrees into a cache. During sampling, there is some probability that a root in this cache is sampled. In this way it is possible for the main tree to reconnect to subtrees of valid nodes previously sampled. The RRT^X [58] was developed for unpredictable obstacles and is capable of adapting to suddenly appearing and disappearing obstructions. When nodes of the tree are invalidated,

the algorithm propagates a series of rewiring procedures across affected nodes to reconfigure the tree such that a shortest path to the goal is maintained. Likewise, nodes that are reintroduced into the tree when obstacles disappear trigger a rewiring cascade to update the tree.

It is crucial to note that while for many variants of RRT it is conventional to select the robot starting configuration as the root of the tree, several dynamic variants of RRT [56–58] set the root as the target configuration of the planning problem. This modification eliminates the requirement to update the root of the tree as the robot traverses along a planned path. Furthermore, in real-world applications newly perceived obstacles often lie in the vicinity of the robot. As such, algorithms that retain valid branches are able to preserve more information as detected changes would generally affect branches furthest away from the root. Consequently, re-planning to the goal in these instances refer to planning a new path from the target configuration to the current robot configuration.

Others work in RRT have demonstrated the possibilities of anytime applications, whereby an initial feasible solution is quickly found and subsequent solutions from the planner improves in quality as further computation time is allowed. In [59], each run of the RRT algorithm was guided by sampling only nodes that may contribute to a solution with a lower cost than a previously found solution, determined by simple heuristics. These heuristics were again applied during node selection and extension procedures to permit only those tree expansions that may improve the quality of the solution. This work was combined with DRRT in [60], incorporating the properties of continuous quality improvements during initial planning time with adaptive re-planning in dynamic environments. Nevertheless, these methods provided no optimality guarantees. To address this, anytime RRT* [61] applies two extensions to the RRT* to enable real-time implementation: committed trajectories and branch-and-bound. A committed trajectory consists of the initial segment of a current solution path. After the initial planning phase, the robot commits to executing this committed trajectory as the planner seeks to improve the remainder of the path. This repeats until the robot reaches the goal by iteratively traversing each of these committed trajectories. Similar to the concept of tree pruning, in branch-and-bound heuristics were used to determine a set of nodes that even at best provided a higher cost to reach the goal than the current solution. These nodes were periodically discarded to improve computational performance.

2.1.3 Machine Learning Methods

2.1.3.1 Reinforcement Learning

Numerous researchers have investigated the viability and potential of machine learning techniques for solving motion planning problems. Reinforcement learning in particular has attracted much attention for their potential to *learn* solutions and reapply them to similar scenarios.

Reinforcement learning seeks to replicate the biological behaviour in humans and animals when interacting with its environment and revolves around the notion of choosing actions based on a current world state and rewarding actions that progress the agent towards a goal state in the environment (as illustrated in Fig. 2.8). Q-learning methods have been demonstrated particularly successfully for motion planning, where the general problem is modelled as a Markov Decision Process. For example, the work in [62] applied Q-learning to solve a path planning problem in a 2D environment for mobile robots. Since Q-learning relies on being able to choose discrete actions at discrete states in the environment (referred to a state-action pair), the authors chose to represent the environment as an occupancy grid, while restricting the motion of a robot to vertical and horizontal movements between cells. During each learning trial, *Q-function values* for each action, which describes the probability of the action being taken at a given state, were updated according to the path found in the trial. Actions that consistently led to the goal being reached converged to higher values than those actions that did not contribute to reaching the goal.

A key limitation of using the classic Q-learning approach is the necessity to represent the environment as a grid, which introduces resolution problems familiar to methods that use an occupancy grid such as the A* algorithm. Jiang et al. [63] extended the Q-learning approach to enable path planning in free space, eliminating the dependency on fixed and uniform grids. This was achieved by applying a fuzzification to state variables to partition the C-space, effectively reducing the size of the state space addressed in Q-learning. These state variables were obtained by segmenting the distances from the robot to the target point and obstacles, and the robot heading. 8 discrete actions were permitted for the navigation of the robot, corresponding to 45° heading directions between 0° and 360°. Q-learning was applied to choose an action based on the fuzzified state variables. While this approach no longer restricted the movement of a robot to

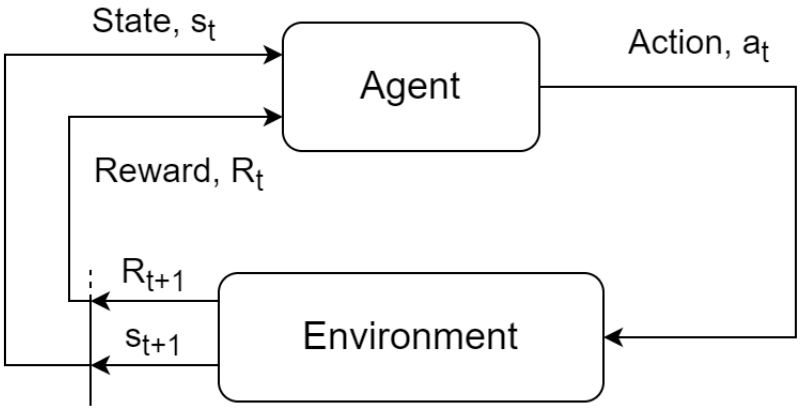


Figure 2.8: The state-action-reward cycle in reinforcement learning, where actions are chosen according to the current state of the world and known reward values. Once an action is performed in the environment, a new reward value is assigned according to the progress achieved by the action to advance towards the goal state.

horizontal and vertical movements across a grid, it still involved discretising the environment, which could prevent the algorithm from finding a solution due to poor resolution.

The work in [64] further addresses this problem by combining the RRT algorithm with Q-learning. Here the concept of growing a search tree was employed to explore the obstacle-free C-space, while Q-learning was used to bias the sampling of configurations. In each iteration of tree growth, the algorithm obtained the nearest node in the tree to the goal configuration and applied Q-learning to sample a set of configurations that were added to the tree. It achieved this by partitioning the range of possible heading angles (i.e. from 0° to 360°) into 8 directions, which were defined as the set of actions for Q-learning. Based on the distance and direction of obstacles around a robot at any given configuration, Q-learning was used to select a direction for configuration sampling such that a new node always lay within a bounded angle range taking into consideration the distribution of obstacles around the robot. In practice, this produced a planning behaviour similar to the Bug algorithms, where the path avoided obstacles by following their contours (though a minimal distance from obstacles was maintained, leading to safer paths). This algorithm was shown to provide faster convergence to a solution compared to the original RRT as it enabled the planner to learn the best samples to use for tree growth, reducing the number of samples required to explore the search space.

So far, all of the above methods based on Q-learning do not extend well for

higher dimensions. In fact their success has only been proven for 2D environments where the heading angle between the robot and obstacles could be determined quickly and easily. This, however, cannot be generalised for higher DoF C-spaces, where obstacles cannot be easily represented in the state space.

In this regard, some preliminary work have been conducted to extend Q-learning to path planning for manipulators. For example, Park et al. [65] investigated the integration of PRM with Q-learning to extend the PRM's path planning capability to dynamic environments, where changing obstacles could affect the validity of nodes and edges in the roadmap. The key idea of their work was to model each node in the roadmap as the states in the environment and the set of roadmap edges, corresponding to motion between two configurations, as the set of possible actions. After the Q-values for each roadmap edge converged through extensive learning, the algorithm was capable of finding new high-quality paths when small changes in the environment invalidated portions of the roadmap. However, this work highlighted one of the key challenges of deploying Q-learning for general path planning problems - explicit representation of C-space obstacles are required. The authors relied upon a spherical wrist assumption [66] to reduce the C-space from six to three by exploiting the unique characteristics of industrial manipulators, where the last three joints affect only the orientation of the end effector without changing the geometry of the entire robot. This meant that the explicit representation of obstacles in C-space could be derived more easily by adopting a simplified slice projection strategy. Another limitation specific to this approach is the lack of evidence showing the advantages of applying Q-learning for searching a roadmap compared to conventional search algorithms.

Meyes et al. [67] also investigated the use of reinforcement learning for industrial manipulators, where Q-learning was applied to teach a 6-DoF robot to play a wire loop game[1]. The problem investigated here could be considered a reduced motion planning problem as it simply involves following a predefined path with the end effector. The authors further reduced the complexity of the problem by addressing the problem in two dimensions only. Here a camera was used to detect the two-dimensional shape of the wire, and an end effector carrying a loop was permitted to perform 6 actions: move left, move right, move up, move down, rotate left by 45° and rotate right by 45°. The environment was represented as a

[1]The wire loop game consists of a wire with various bends, and the objective of the game is to move a loop along the wire from start to end without making contact with the wire.

grid where states were defined by a local window containing the relative position of the end effector, the wire, and an intermediate goal specified a fixed distance away from the robot along the wire. The authors demonstrated that the algorithm could successfully solve the problem for a given wire after a few minutes of computation time. Furthermore, the authors showed that the method could enable a robot to adapt to different wire shapes using only the learnt behaviour from a single training problem, but this required several hours of training.

As seen in the above reviewed literature, a number of common limitations exist across the various implementations of Q-learning for motion planning. Firstly, Q-learning methods generally require the discretisation of the environment in some way to reduce the continuous C-space into a discrete state space. This has important implications on the completeness of the algorithm as poor resolutions can lead to failure in finding solutions that exist. Secondly, the process of discretising the environment has, without exception, required the use of obstacle information (either to specify the occupancy of cells or for defining the states of the environment). Therefore, like deterministic methods, Q-learning algorithms require explicit representation of the environment in C-space, which, as we already know, is a computationally expensive process. These two limitations mean that a reinforcement learning approach generally scales poorly with the dimensionality of the problem. Furthermore, its effectiveness and reliability for general motion planning problems involving different start and goal configurations have yet to be proven. In all of the reviewed literature, the process of learning was applied to a single problem involving a fixed start and goal configuration. It is currently unclear how a learning-based algorithm's performance deteriorates when these start and goal configurations change. However, reinforcement learning has proven useful for dynamic applications where the start and goal configurations remain fixed but the environment is prone to change. They offer a fast, adaptive approach to these problems as explicit computation is not required once extensive learning has identified useful actions in reaching a goal state. As the algorithm continues to learn through repeated trials, improved performance can be observed. This does, however, come at the cost of a computationally long learning phase involving large iterations of solving the same planning problem before good results can be achieved.

2.1.3.2 Learning from Experience

An alternative learning-based approach consists of using a "learning from experience" framework, where solutions of past problems are stored in a database that is retrieved for similar problems and repaired to satisfy any new problem constraints. The Lightning framework proposed by Berenson et al. [68] is a highly influential work that follows this paradigm. The authors proposed a parallel planning framework comprising of a planning-from-scratch component and a retrieve and repair component. The idea revolved around the use of both components to attempt to solve a path planning problem in parallel. Whichever method obtained a feasible solution first was taken forward for post-processing and returned as the solution to the problem. This solution was then stored within a library containing past solutions. Planning-from-scratch simply consisted of using a standard motion planner such as those already covered to compute a solution. Retrieve and repair, on the other hand, used the past solutions stored in the library to devise a solution to the current problem. It first retrieved a number of similar solutions to the problem at hand by comparing the respective start and goal configurations. All similar solutions were then graded by evaluating the percentage of each path that were in collision for the current problem. The best solution was taken forward and repaired by removing any infeasible segments and applying a bi-directional RRT to reconnect broken paths.

One drawback of the Lightning framework was the high computational cost of storing many past solutions. Coleman et al. [69] addressed this with the Thunder framework, which can be considered an extension of the Lightning framework. Here the authors leveraged the fact that past solutions often share some of the same nodes that make up a path. Instead of storing each past solution directly, the framework employed a sparse roadmap spanner [70] (a more efficient variant of roadmap structures) that was progressively built up using the nodes of past solutions, allowing duplicate nodes across past solutions to be removed. As continued planning and learning took place, those nodes that were rarely used were also removed to limit the size of the sparse roadmap spanner. Like with the Lightning framework, this was run in parallel with planning from scratch such that entirely new planning problems could still be solved using conventional motion planning methods.

The key advantage of both the Lightning and Thunder frameworks is their capability to leverage past solutions to speed up the search for a solution, which is

particularly important for complex environments where individual motion planning queries are expensive to solve. However, a major drawback of these frameworks is the lack of guarantee for finding high-quality solutions. Since retrieved solutions are repaired locally, there is no guarantee that a previously optimal solution will remain optimal after repairing for the current problem. Furthermore, if a past solution was generated from a complex, cluttered environment and subsequently reused for a clutter-free environment, the learning process will carry over a highly sub-optimal solution for the new environment. These problems can be partly addressed through post-processing, but the quality of such a path is highly dependent upon the computation time allocated to post-processing.

This was later addressed by Abdelwahed et al. in [71], who proposed two new case-based reasoning methods for motion planning based on learning from past solutions. These methods, named CBR-FPath and CBR-Graph, respectively, follow the same principle of storing the solution of past planning problems in a library. For any new motion planning query, both methods retrieved a set of similar solutions based on the start and goal configurations and the percentage of paths that weere invalid (much like the Lightning framework). However, rather than seeking a single best solution from among this set and repairing it, the CBR-FPath method used the nodes that form these solutions as a candidate set of configurations for the expansion of a path from the start configuration towards the goal configuration. At each iteration of this expansion, the algorithm selected the newest point along the point and sought the best configuration from among the candidate set according to an A*-like heuristic function. This was added to the path and the expansion continued. In a similar way, CBR-Graph retrieved all nodes from among the similar solution set. It then constructed a PRM* roadmap using these nodes and the path planning problem was solved by searching the roadmap with A*.

The difference between CBR-FPath and CBR-Graph lies in the performance of the algorithms in terms of plan quality and planning efficiency. CBR-FPath is a greedy method, where the best candidate configuration at a given state is always added to the path and no backtracking takes place. This can lead to sub-optimal solutions or, in the worst case, failure to find a solution. On the other hand, CBR-Graph guarantees that the best possible solution that can be obtained from the set of configurations is always found. However, since it involves the construction of a roadmap during an online planning query, it is less computationally efficient

compared to CBR-FPath. Nevertheless, both methods overcome the problem of Lightning and Thunder where the retrieve and repair framework can lead to sub-optimal solutions.

A key limitation of using either CBR-FPath and CBR-Graph is the necessity to sacrifice either solution quality or planning efficiency to achieve good performance in the other aspect. This is apparent as CBR-FPath can fail to find a solution altogether due to its greedy characteristics, while CBR-Graph requires the construction of a new roadmap every time is it used. For dynamic planning applications, these respective limitations can negatively affect the robustness of the robot to respond and adapt to changes quickly.

2.1.3.3 Evolutionary Algorithms

Numerous authors have investigated the viability of applying evolutionary algorithms to solve motion planning problems. For example, the Genetic Algorithm (GA) was used in [72] to solve a 2 dimensional motion planning problem where the environment was represented by a discretised grid. An objective function was developed to evaluate solutions according to both path length and the number of invalid points along the path that lie in collision. The approach possessed two key flaws that limit its practicality for solving real-world motion planning problems. Firstly, a good initial population consisting of feasible solutions was required to ensure the rate of convergence to a high-quality solution. Yet the time required to generate this population was substantial, which accumulated with the already costly computation of solving the GA problem. Secondly, the approach scales poorly with the dimensionality of the search space and the granularity of the grid representation.

Achour et al. [73] sought to improve the performance of GAs for motion planning by combining it with sampling. Rather than explicitly discretising the environment into cells, the algorithm first generated a large set of random samples across the C-space, much like the idea of sampling the search space adopted by PRM and RRT. The chromosome used in GAs then encoded a path as a sequence of samples. By applying the GA, it was possible to obtain a path formed by a connection between the sampled configurations. Compared to the PRM, this algorithm was able to find shorter, smoother solutions as the connections between samples were not limited by an upper bound on its length, unlike the PRM. However, this came at the cost of very high computation time. Though this method

could in theory be extended to higher DoFs, the computational cost of doing so would generally be impractical.

In [74] and [75], authors studied the use of classic Ant Colony Optimisation (ACO) for motion planning. ACO was inspired by the biological tendency of ants to deposit pheromones along their route that evaporates over time. These pheromones attract other ants to follow the same trail. When sourcing food, shorter paths taken by a sub-population of ants will have a greater deposit of pheromones as ants traverse these paths more quickly than the longer routes, leading to a greater concentration of pheromones. Over extended periods a population of ants will converge to the shortest path according to their attraction to stronger pheromones. Likewise, the ACO in motion planning consists of conducting a large iteration of trials in which a population of virtual ants explore the environment in search for the goal. Like the GA and many other algorithms already discussed, the environment is represented as a grid. The probability of an ant to advance from one cell to the next is determined by the pheromones in each cell. Higher pheromone content equates to a higher probability. At the end of each trial, all successful paths are used to update the pheromones of the grid according the length of the path found. Thus after a large number of iterations, the algorithm will converge towards the best solution found across trials. The authors of [74] highlighted that this behaviour gives the algorithm nice properties for adapting to dynamic environments as previous pheromones provide a good starting point to find the new shortest path, with their values continually being updated as new solutions are obtained.

A major drawback of the ACO is the long planning times required to find a solution due to the slow convergence to a solution during early iterations where no pheromones have been laid. Dai et al. addressed this by combining the ACO with A* heuristics [76]. Rather than relying solely on the pheromone content to guide the actions of ants when navigating the environment, a heuristic value was computed according to the pheromone content and the estimated cost to reach the goal through the considered cell. Furthermore, the authors also proposed the use of a MAX-MIN ant system to overcome premature convergence of solutions in complex, narrow and cluttered environments. This can occur when a large number of ants follow a sub-optimal route, leading to a higher probability of future trials to converge to the same solution. Rather than using all successful paths found in a trial to update the pheromones in each iteration, only the best solution

among those found in the same trial were used. Using these two modifications, the authors showed that higher quality solutions could be obtained with lower computation time. However, despite this, the authors reported planning times of up to 88 seconds for the 2D environments considered in their work. Indeed subsequent re-planning in dynamic environments could be achieved much faster as the majority of this time was consumed by the initial convergence of trials. Thus considerations for the use of the ACO must account for the trade-off between long computation times for solving an initial instance of a planning problem and the potential for fast re-planning in dynamic environments.

While both the GA and ACO generally require a discrete representation of the environment, the Particle Swarm Optimisation (PSO) method, which was inspired by the swarming behaviour of biological animals, has been demonstrated in continuous space more successfully in motion planning applications. The work in [77], for example, applied the PSO for path planning in dynamic environments. The initial behaviour of their approach was similar to the Bug algorithms, where the planner sought to maintain a direct path from the start to the goal configuration. When an obstacle was encountered, the planner called the PSO to locally generate a new intermediate target point for avoiding the obstacle. When called, the PSO algorithm distributed a population of points around the robot according to a maximum radius and sought to find a point that minimised an objective function. The objective function evaluated the quality of a point according to the length of the path through the evaluated point from the robot's location to the goal. It included a penalty term for segments of the path that collided with obstacles.

This method naturally extends well for dynamic applications as it adopts a reactive strategy for navigating past obstacles. In fact, the PSO method for motion planning produces behaviours very similar to the Bug algorithms, where the robot contours obstacles as they are encountered to navigate past obstructions. It is advantageous over the Bug algorithms as the PSO does not require explicit specification of rules for leaving the contour of an obstacle and can reliably produce high quality solutions. However, like the Bug algorithms, it also requires the explicit representation of obstacles in the search space, which limits its practicality for high DoF robots.

2.1.4 Key Findings

Table 2.1 summarises the key limitations of each class of motion planning methods based on the findings from the literature review. As it shows, every class of methods possess their own set of limitations that limit their effectiveness in certain scenarios. In particular, a common limitation across deterministic methods, reinforcement learning-based methods and evolutionary algorithms is the dependency on explicit knowledge of obstacles in the C-space and the requirement to discretise the environment. This leads to poor scaling for high dimensionality problems and expensive computations not suitable for dynamic applications.

Bug algorithms, APF and PSO methods provide an interesting reactive navigation approach to planning, where explicit planning is not required for finding a high-quality path to a goal. Their reactive behaviour by nature lend themselves as effective approaches for responding quickly to dynamic obstacles. However, they are susceptible to other problems such as the risk of being trapped in local minima and failure to account for general optimisation criteria as they simple seek to follow a straight path from the current robot configuration to the goal.

Sampling-based algorithms have proven popular in the robotics community for their efficiency in exploring high-dimensional C-spaces. They are particularly advantageous over other methods for being able to handle continuous state spaces without discretization and do not require explicit representation of obstacles in the C-space. However, they are probabilistically-complete and asymptotically optimal at best, meaning that extended planning time is often required to find high-quality solutions. Nevertheless, the concept of anytime planning has appeared in these methods, providing a means of finding feasible solutions quickly and subsequently improving the quality of solutions over time. This behaviour also appears in ACO-based methods, which iteratively seeks to converge to the optimal solution through repeated trials. Various works have also successfully demonstrated the extension of these methods to dynamic applications, making sampling-based algorithms very versatile.

Learning from experience opens up new possibilities for learning from past planning problems, providing a mechanism to quickly retrieve and reuse existing solutions without impairing the capability of the planner to find completely new solutions through conventional planning. The effectiveness of these methods are highly dependent on storing a large set of solutions. However, a key challenge with these methods is the high memory costs required to store this information.

Table 2.1: Motion Planning Limitations

Method	Limitations
Deterministic methods (non-reactive)	• Obstacles must be explicitly represented in the C-space, which is computationally expensive to compute. • Some methods require discretizing the environment, which limit the algorithms to being resolution-complete at best. • Due to the above two limitations, all methods scale poorly and many have not been proven for C-spaces beyond 3 dimensions.
Reactive methods	• Unable to provide high-quality solutions as paths are planned locally as obstacles are detected. • The robot can be trapped in local minima. • Some reactive methods cause the robot to remain close to obstacles, increasing the likelihood of collision in practice. • Unable to account for other optimisation criteria as the algorithms naturally follow the shortest length path by default.
Sampling-based methods	• Algorithms are probabilistically-complete at best, meaning they are unable to identify the non-existence of a solution to a problem. • Algorithms are asymptotically-optimal at best, requiring extended planning time to obtain high quality solutions. • Multi-query algorithms require pre-processing, which is computationally costly and requires high memory requirements.
Reinforcement learning-based methods	• Feasibility of reinforcement learning has not been proven for higher dimension problems. • Requires some form of discretisation to determine the best action to take at any given state, which limit the scalability of these approaches to low-dimension problems. • The generality of a learnt behaviour has not been proven for varying start and goal configurations - learnt behaviours are confined to a single planning problem. • Requires extensive learning to develop high-quality solutions.
Learning from experience	• These methods have a high memory cost for storing past solutions. Ongoing research in this area are seeking to identify more efficient representations of past solutions and to provide fast mechanisms for their retrieval and reuse. • Running two planning routines in parallel can be challenging for systems with limited computing capacity.
Evolutionary algorithms	• GA and ACO requires the discretisation of the environment and explicit knowledge of obstacles in the C-space. • Studies have been limited to 2D and 3D environments, and reported computations are generally considered long for solving motion planning problems. • Requires carefully chosen parameters that are specific to each problem and may not be known a priori.

Based on these findings, this thesis proceeds with the use of sampling-based algorithms for the investigation of optimal and adaptive planning in robotics. While other classes of methods possess certain features that are also relevant to adaptive robotics, sampling algorithms have been chosen for their reliability and versatility. Crucially, sampling algorithms possess the nice property of scaling efficiently for high DoF robots while being able to provide high-quality solutions.

However, with reference to the research questions identified at the start of this research, a number of challenges must be addressed to effectively enable the use of sampling-based algorithms for adaptive planning. Firstly, single-query sampling algorithms are generally inefficient for solving multiple path planning problems in the same environment as each query must be solved from scratch. Seeking to achieve optimal solutions require long planning times to repeat the search across each problem, while reducing the planning time results in poorer quality solutions. Multi-query algorithms, on the other hand, are able to achieve high quality solutions quickly but require an extensive pre-processing phase to extensively explore the search space. However a large portion of explored information may never be used for solving the actual planning problems. This makes multi-query algorithms more costly than necessary, which is undesirable for fast, online planning. In Chapter 4, I explore a new method that consists of leveraging the idea of using one path planning problem to solve another for faster planning (a key paradigm in learning-from-experience methods) by simultaneously planning a solution for all problems such that the same sampled information is reused. Through a series of evaluations, I demonstrate that the algorithm is capable of solving multiple motion planning problems more quickly compared to using single-query planners while eliminating the requirement for pre-processing. This chapter also discusses how the algorithm can account for general cost criteria, a feature that is not commonly available with other motion planners in literature.

Another limitation of sampling-based algorithms is the growth in memory resources consumed by sampling algorithms over large iterations. This can significantly slow down the system when deployed on-board to robots with limited computing capacity. To overcome this limitation for the physical implementation of the algorithm described above, I present a new strategy in Chapter 5 for limiting the number of stored configuration samples without impairing the capability of the algorithm to find high-quality solutions.

In Chapter 5 I also describe how the motion planner introduced in Chapter

4 can be extended to enable dynamic re-planning such that new collision-free, high-quality solutions can be computed without re-planning from scratch when new observations of the environment is perceived.

2.2 Task Planning

Task planning seeks to provide intelligence to a robot by enabling reasoning capabilities that translate a high-level set of commands into clearly-defined executable actions. This type of problem falls under the umbrella of AI, which studies the idea of intelligent agents that interact with its environment autonomously through perception and actions. A common sub-problem of AI lies in defining an efficient representation of world[2] knowledge and the planning problem, while the final objective is to determine a set of actions to meet defined goal requirements.

Planning actions toward a set of objectives require a clear specification of the *world states*. The initial description of the world at the start of a planning problem is an *initial state*, while the set of objectives describe the *goal state*. Each *action* results in a state transition in the world described by its *effect* that depends upon the prior state when performing the action. Thus the set of all possible robot actions and their effects define the complete state space. To solve a task planning problem, the set of actions, effects, and the state space (collectively described as the problem *domain*) must be represented in a way that allows the state space to be searched for a solution that gives the sequence of actions to transition from the initial state to the goal state.

The first half of this section reviews fundamental literature in general task planning for robotics. Given the wide range of research in the field, this thesis will then narrow down on two specific task planning problems: (i) task planning for mobile wheeled robots and (ii) robotic task sequencing. Thus the second half of this section is devoted to covering the state-of-the-art relevant to these areas. I first cover these two problems from a static planning perspective, and then go on to highlight some recent work in the direction of adaptive task planning and discuss the current knowledge gap that is addressed by the research presented in this thesis.

[2]Note that the definition of *world* here strictly refers to the local environment that the robot operates in, and representing this world often involves capturing a highly reduced set of domain physics applicable to the local planning problem.

2.2.1 Fundamental Task Planning Techniques

2.2.1.1 Constraint Satisfaction Problem

The constraint satisfaction problem (CSP) is a notably successful formulation for solving symbolic task planning problems. In the CSP formulation, three descriptors are used to represent all the components of a planning problem [78]:

1. $X = \{x_1, x_2, ..., x_n\}$, a set of variables used to describe a state, where each variable takes on a value from a set of allowed values defined by the domain.

2. $D = \{D_1, D_2, ..., D_n\}$, a collection of sets of valid domain values associated to each variable in X. $D_i \in D$ is composed of all the valid values $\{v_{i,1}, v_{i,2}, ..., v_{i,m}\}$ that can be assigned to the corresponding variable, x_i.

3. $C = \{c_1, c_2, ..., c_k\}$, a set of constraints that specify the allowed relationships between variables, where c_j is described by a pair $\langle scope, relation \rangle$. *scope* is the tuple of variables that are members of the constraint and *relation* specifies the combination of values that can be assigned to each of the involved variables. *relation* can be given as a list of all the accepted combinations of values, or as a relational descriptor (e.g. $x_1 \geq x_4$ or $x_3 \neq x_6$).

A solution to the CSP is composed of an assignment of values to each variable, which defines the goal state based on the initial conditions of the problem. This assignment is *complete*[3] if every variable is assigned an allowed value, or *partial* when only a portion of the variables have a value assigned. Additionally, the assignment is *consistent* if all assigned values satisfy the constraints defined in C.

Consider as an example the simplified problem shown in Fig. 2.9. It consists of finding a task plan for a mobile manipulator (e.g. a mobile robot integrated with a manipulator) to transport two objects, a cup and plate, from a table for washing at a sink, and then storing them on a shelf. The robot has three possible actions for each object: *move2sink*, *wash* and *store*. For convenience, let *move* be equivalent to *move2sink*. Representing these as variables in the CSP formulation gives: $X = \{move\_cup, move\_plate, wash\_cup, wash\_plate, store\_cup, store\_plate\}$.

[3]Note the difference in its definition compared to the use of this term for path planning. In path planning, *complete* describes an algorithm's guarantee to find a solution if it exists.

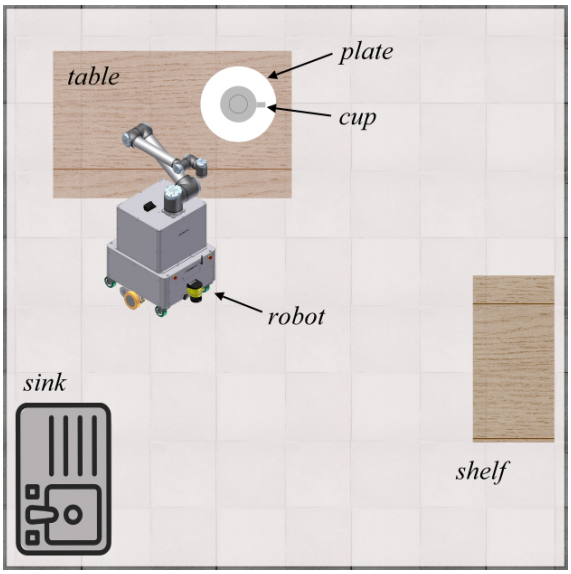


Figure 2.9: A simple task planning problem for a mobile manipulator in a kitchen environment. The plate and cup placed on the table is required to be moved to the sink, washed and transferred to the shelf.

As this planning problem consists of finding a valid sequence of actions, each set of domain values in D can correspond to integer numbers that describe the order of the action in the solution sequence. In this case we could assign $D_i = \{1, 2, 3, 4, 5, 6\}$ for all actions such that they may take any value between 1 and 6. Alternatively, we could model the values as the time for the task to start by representing minutes with an integer value.

In either case, constraints must be formulated appropriately for the domain value representation chosen. In this example, the robot is given the following rules: (i) the robot can only move one object at a time, (ii) the plate cannot be moved before the cup as the cup sits on top of the plate, (iii) an object cannot be washed until it has been moved to the sink, and (iv) an object cannot be stored until it has been washed. These rules can be translated into *precedence constraints* (that is, constraints that dictate that a certain action must take place before another) as follows. Rule (i) and (ii) can be formulated as $move\_cup < move\_plate$[4]. This constrains the value of *move_plate* (and its order of appearance in the sequence) to always be greater than *move_cup*. Likewise, rules (iii) and (iv) impose the following constraints: $move\_cup < wash\_cup$, $move\_plate < wash\_plate$, $wash\_cup < store\_cup$ and $wash\_plate < store\_plate$. Finally, rule (i) also im-

[4]For brevity, let this be equivalent to $\langle(move\_cup, move\_plate), move\_cup < move\_plate\rangle$. The same shortened representation is used for all constraints when referring to CSP formulation.

poses the final constraint $store\_cup \neq store\_plate$ such that the cup and plate cannot be moved for storage simultaneously.

Even for this simple problem, there exists multiple solutions. A valid solution could involve moving the cup to the sink, then the plate to the sink, wash the cup and plate sequentially, and finally move both to storage one at a time. Alternatively, moving, washing and storing the cup and repeating for the plate is also a valid solution. To obtain these solutions by search, a choice between two main types of algorithms are generally adopted.

In **backtracking search**, a depth-first search strategy is applied to generate a partial assignment that grows towards a complete, consistent solution. Starting from a single variable with an initialised value, each step involves advancing through a branch by adding a value assignment to an empty variable and checking for consistency. The search cycles through all allowed values for the considered variable until either a value that is consistent with all constraints is found or all values would violate a constraint. In the latter case, the search backtracks to the previous level and selects another branch to explore, terminating when a valid solution is obtained or when all branches have been explored [79].

Alternatively, one may choose to use a **local search algorithm**. Local search algorithms differ from backtracking search by initially assigning a value to all variables. It then seeks to eliminate all violated constraints by locally adjusting one value at a time. Heuristics are used to determine the best value to reassign to a variable from among the domain value set. The **min-conflicts** heuristic deserves particular mention for its effectiveness in solving CSPs. It objectively selects values based on minimising the number of conflicts with other variables and have proven its effectiveness for complex problems including the scheduling of observations for the Hubble Space Telescope [80].

CSPs offer a number of advantages for solving task planning problems compared to conventional state space search. Firstly, rather than individually assessing single states to check satisfaction of goal objectives, partial assignments to CSPs can determine quickly whether a branch of candidate solutions violate the constraints of the goal objectives. This can enable fast elimination of large regions of the search space when values have been assigned to a number of variables. Furthermore, CSP provides transparency in *how* an inconsistent solution violates goal requirements, which can inform a search algorithm to correct such conflicts. Ultimately, CSPs are able to solve planning problems very quickly in

many scenarios where conventional search methods would suffer from intractability.

The downside to using CSPs for robotic task planning is the difficulty in representing some rules and constraints in the real-world using CSP formulation. This is particularly true for modern robotic applications involving complex interactions between a robot, obstacles, and objects, especially in a dynamic environment. Furthermore, a large number of variables are needed to represent a robot's diverse skillset, which can be complex and time-consuming to model for largescale problems.

2.2.1.2 Planning Domain Definition Language

While CSPs have demonstrated high efficiency in solving problems that can be represented as constraints, it lacked the expressiveness to represent many common task planning problems in robotics in a concise way. As we saw in the example shown in Fig 2.9, every action involving the transfer of an object must be defined individually as an action variable for every object and every possible transfer. This can quickly expand into to a very large number of variables for a set of very similar actions. This section introduces the Planning Domain Definition Language (PDDL), an expressive language to overcomes these limitations.

The PDDL was first conceptualised in 1998 as a standardised way of representing AI planning problems [81] and serves as a standard for assessing entries to the International Planning Competition (IPC). With each IPC, the number of features supported by PDDL grew, resulting in the release of several official versions of PDDL (the latest being PDDL3.1). Since PDDL is a standardised language fit for a broad range of planning problems extending well beyond the scope of robotic task planning, I do not exhaustively describe the individual features supported by the various versions of PDDL. Instead, I cover the core features of PDDL set in the context of robotic task planning, though these features discussed is common to all planning problem representations.

At the core of PDDL are two definitions that completely describe a task: the *domain* and the *problem*.[5] As the name suggests, the domain definition describes the representation of world **states** and all state transitions through

[5]In practice, these two definitions are stored in two separate text files, both with a *.pddl* file extension, which are then parsed by whichever planner is used to solve the actual planning problem.

four components: *predicates*, *actions*, *preconditions* and *effects*.

Predicates are used to describe propositional formulas (or *atomic formula*, **atom** for short) to be assigned to objects considered in a task planning problem. It is important to recognise that PDDL represents planning problems in symbolic form. Consider an **object**[6] called *Arm1* representing a manipulator for which we would like to plan a set of tasks for. By itself, *Arm1* carries no meaning, and could in fact be assigned any arbitrary name such as *6dofArm*, *manip*, or even *John*. Now let us define a predicate *(IsRobot ?object)*. We may think of this predicate as a symbolic function. Here, we instantiate a predicate called *IsRobot*, which is assigned to any object passed into the predicate as the input for *?object* (the '*?*' prefix is used to identify an input to the predicate). For example, once we have defined this predicate, the line *(IsRobot Arm1)* specifies that Arm1 is a robot. Predicates enable the description of 'relationships' between objects by supporting multiple inputs to a predicate definition. A predicate defined as *(holding ?robot ?object)* would specify that the first input object is holding the second input object. All objects are initially false for all predicates. Once an assignment is made to an object, this instance of the predicate becomes true. The truth values for each atom is called a **fact**. By using these predicates, the complete state of the world can be described by a conjunction of facts. Conditional evaluations of the world state can then be performed by testing the logical value of atoms. To reverse a predicate assignment, a negation can be used such as *not(holding ?robot ?object)*, which assigns the value of false to the atom (indicating that the robot is no longer holding the object).

Actions are used to describe the possible state transitions in the world and are comprised of **preconditions** and **effects**. For every action, a set of *parameters* are defined to specify the objects that are considered in the action. Preconditions impose requirements that must be met for an action to take place, while effects describe the new state of the world resulting from said action taking place. Following the examples before, let us first define a new predicate *(At ?object ?location)*, which specifies that input 1 is located at input 2. Now suppose the parameters of the action *transfer* are *?object1*, *?object2*, *?location1* and *?location2*. A precondition *(IsRobot ?object1)* $\wedge$ *(holding ?object1 ?object2)* $\wedge$ *(at*

[6]Note that the term *object* does not necessarily refer to a physical object, but any subject of interest. For example, this could be a point in space that is of particular importance (e.g. a home position).

?object2 ?location1) enforces the constraint that *object2* cannot be transferred unless *object1* is a robot, *object1* is holding *object2*, and *object2* is at *location1*. An effect specified as *not(at ?object2 ?location1)* $\wedge$ *(at ?object2 ?location2)* gives the resulting state change corresponding to *object2* no longer being located at *location1* and is located at *location2*.

Following this *action schema*, it is easy to see how the numerous action variables required in CSP can be condensed into a single action description in PDDL. Simply by substituting different objects into the specified parameters for the transfer action, all possible state transitions associated to the transfer of objects can be accounted for. Thus PDDL provides a powerful platform for representing the range of actions that can be executed by a robot when operating in less structured environments.

The *problem* definition defines the objects considered in the problem, and the initial and goal states of the world. These are specific to a particular problem instance for the given robot domain. For example, a service robot could be tasked on one occasion to set up a table with plates and cutlery, while on a separate instance be required to clean dirty dishes on the same table. Each of these would be defined as an individual problem but correspond to the same domain definition.

The set of objects are specified as a list of readable symbolic names chosen for human interpretation. Initial and goal states are then defined as a conjunction of atoms using this set of objects, e.g. for a problem consisting of the objects *Robot*, *Sink*, *Cupboard* and *plate*, an initial state may include *(IsRobot Robot)*, *(at plate Cupboard)* and *(at Robot Sink)*, while a goal state may consist of *(at plate Sink)*. Any atoms that are not explicitly defined in the initial state are assumed to be false by default, while any atoms not included in the goal are not considered a constraint on the solution. For the above example, solutions with either *(at Robot Sink)* or *(at Robot Cupboard)* as an end state would satisfy the problem as long as *(at plate Sink)* is satisfied.

Once a problem is represented in PDDL, the objective of a planner is to find a sequence of symbolic actions (including the corresponding objects involved for each action) that, when applied sequentially at the initial state, leads to a sequence of state transitions ending at the goal state. Importantly, the preconditions of actions must be met at the state in which they are performed.

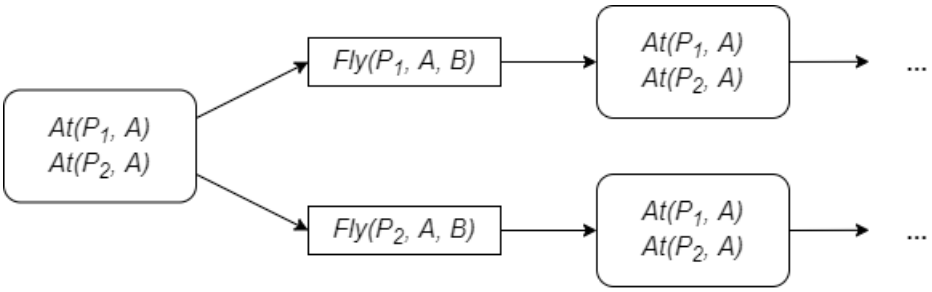


Figure 2.10: An example of forward state-space search in symbolic task planning, where two planes P_1 and P_2 are located in airport A. Two possible *fly* actions can be taken to transport either plane from airport A to airport B. The forward state-space search evaluates each of the two possible resulting states to determine which branch to pursue.

2.2.1.2.1 FastForward Planning System A number of planners have gained popularity for solving PDDL-based problems after finding success in the IPCs. The **FastForward (FF)** planning system [82] is based on a forward state-space search that is informed by heuristics to estimate the goal distance. This forward chaining approach involves expanding the search from the initial state and exploring branches through possible actions. A pure search without effective heuristics for guiding the search is impractical as it would result in an exhaustive search through many irrelevant branches of actions. Problems quickly become intractable when states have on average a moderate to high *branching factor*. FF addresses this limitation by adopting a goal distance heuristic estimate based on the number of actions to reach the goal state. A simple illustration of forward state-space search is shown in Fig. 2.10.

The FF planning system obtains this heuristic estimate by *relaxing* a task. Generally speaking, a relaxed problem is subject to less goal requirements, which makes it easier to solve. In the case of FF, the relaxed problem ignores all 'delete' operators that lead to the negation of atoms in the effects of actions (such as *not* functions). In other words, no action undoes the effects of actions earlier in a plan. During a forward search, a heuristic goal distance estimate must be computed for each expanded state. This estimate is obtained by representing the planning problem from the expanded state to the goal as a relaxed planning task with no delete operators. The authors of FF used the GRAPHPLAN algorithm [83] to obtain a relaxed plan, which is substantially easier to solve. The length of the relaxed plan is then used as the heuristic for the goal distance estimate. Critically, this heuristic is admissible as it never overestimates the length of the plan.

The forward search is conducted using an Enforced Hill-Climbing (EHC)

search to find successor states. Starting from the initial state and an empty *current plan*, EHC seeks a successor state with a lower heuristic value (evaluated using the relaxed plan) by performing a breadth-first search, and adding the resulting sequence of actions that reach the successor to the current plan. The FF planning system also incorporates search space pruning to improve FF's computational performance based on the concept of *helpful actions*. During the construction of a relaxed plan for heuristic evaluation, all actions that can take place in the first time step and would advance the search towards the goal are stored as a set of helpful actions. When seeking a successor state in the EHC algorithm, only helpful actions are considered for expansion during breadth-first search, which avoids time spent exploring irrelevant action paths.

While these procedures have been shown to be effective in solving most problems, the EHC search is prone to fail (it does not guarantee a solution even when one exists). The authors overcame this through the inclusion of a recovery procedure that reverts to an exhaustive best-first search using the goal distance estimate to expand all states until a solution is found when a mode of failure is encountered.

The FF planning system originally handled only logical variables, that is atoms possessing either a true or false value. The authors shortly extended FF into **Metric-FF** [84] to incorporate support for numerical state variables, or **fluents**. These enable the assignment of numerical variables that keep track of numerical quantities such as battery levels, accumulated cost, distance travelled etc. Comparisons of numerical values can be used to define constraints in preconditions of actions, while numerical expressions enable changes to fluents in the effects of actions using operators such as $+$, $-$, $\times$ and $\div$. Planning a task with Metric-FF enables the *optimisation* of solutions by minimising costs, or introduce additional constraints that could not be represented by purely logical values (e.g. limitations due to battery limitations of a mobile robot).

2.2.1.2.2 LPG The **LPG** [85] is a planner than solves a planning task represented as a *planning graph* [83] through local search. Planning graphs are composed of two alternating types of layers, an action layer consisting of nodes made up of actions, and a facts layer made up of fact nodes. The level of actions and facts layers are separately numbered chronologically as shown in Fig. 2.11. Each level t in the action layer represents the set of actions that can take place

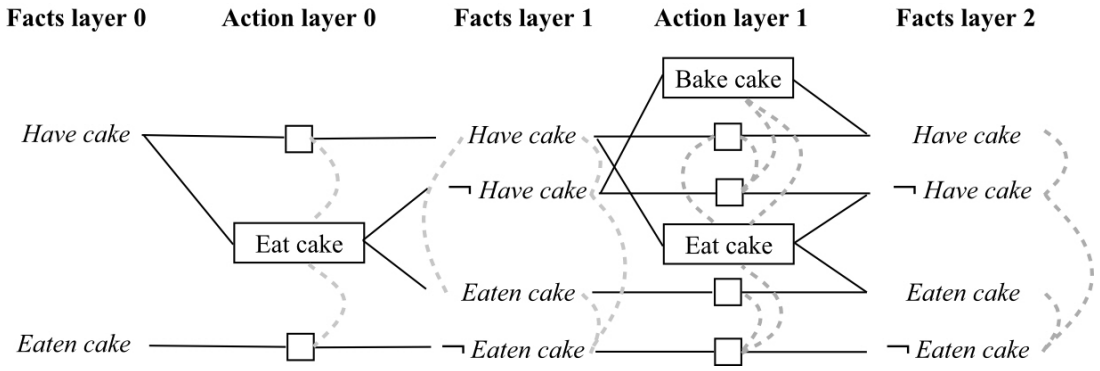


Figure 2.11: Example of a planning graph for a *have cake and eat cake* problem consisting of up to facts layer 2. Blank squares represent dummy actions, straight lines represent pre-condition and effect links, and grey dashed lines represent mutual exclusion. (Adapted from [79])

in the t^{th} step of a plan, while the t^{th} layer in the facts layer represents the pre-condition of actions in action layer t and the effects of actions in action layer *t-1*. Edges connect fact nodes to action nodes (pre-condition links) and action nodes to fact nodes (effect links), while dummy action nodes and edges are used to pass facts from one fact level to the next unchanged (i.e. when facts are unaffected after an action takes place). When constructing a planning graph, the last level of the fact nodes should contain all the facts that satisfy the goal state of the planning problem.

Conflicts between actions are represented through *mutual exclusion*. This occurs when an action either negates a required precondition or an add effect of another action. Alternatively, two actions are considered mutually exclusive when their preconditions are mutually exclusive. The latter occurs when two facts in a layer cannot be true at the same time (i.e. if all ways of making one fact true prevents the second fact from becoming true). Conflicting pairs are described as possessing a *mutex* relationship.

LPG uses *sub-graphs* called **action graphs** as a search space to find partial plans. When constructing an action graph, any action taken from the planning graph must also include the fact nodes directly connected as a precondition or effect in the planning graph. The objective is to construct a complete action graph that contains the facts required to satisfy the goal state of the problem in the final layer. LPG achieves this by iteratively adapting the graph through addition and removal of action nodes starting from an initial action graph (generated randomly at the start of each planning instance). In each iteration, the addition and removal of actions are determined by the mutex relations in the action graph

and any precondition facts that are unsupported. That is when no actions exist in the action graph that connects the fact nodes of the first layer (i.e. the initial state of the world) to the required precondition of an action later in the graph. These two criteria for graph modifications are collectively referred to as *constraint violations*.

When resolving a constraint violation, each possible modification to the action graph that resolves the violation to generate a new action graph is stored in a **neighbourhood** set. These graphs are ranked by an evaluation function that determines the best action graph to take forward. This evaluation score, termed *action evaluation function*, is determined by the number of mutually exclusive actions, the number of unsupported facts, the number of unsupported actions that become supported after adding an action, and the number of supported actions that become unsupported after removing an action. Additionally, the quality of a plan can be consolidated through weighted terms for criteria such as plan length and action costs within the evaluation function. This mechanism provided LPG with the capability to progressively improve the quality of a solution once an initial feasible solution is obtained.

LPG was later extended into the **LPG-*td*** planner [86], which provided additional support for numerical expressions within preconditions and effects, as well as compatibility with *time durative actions*. Durative actions carry an additional duration parameter that specifies the time required to execute an action. LPG-*td* thus enables the progressive improvements of solutions in relation to either action costs, total plan duration or some other numerical function.

2.2.1.2.3 Sub-Goal Partitioning and Resolution in Planning The **Sub-Goal Partitioning and Resolution in Planning (SGPlan)** system [87] solves a planning problem by partition the entire task into smaller planning tasks, each with separate sub-goals that are solved independently. Inconsistencies in the solutions to individual sub-goals are corrected for by using the concept of an extended saddle-point condition to evaluate the satisfaction of global constraints.

The SGPlan system comprises of a **global-level** planning step and a **local sub-goal level** planning step. In the global-level, the planning task is broken down into sub-problems, each consisting of only one sub-goal. The sequence in which these sub-goals are evaluated matters as sub-goals are often dependent. Solving one sub-problem may lead to reversing the result of a previous sub-goal.

A meaningful ordering can significantly reduce the frequency of these conflicts. Based on these considerations, SGPlan employs a multi-level procedure to determine a suitable ordering of sub-goals. This procedure applies the following heuristics in chronological order for partial ordering of sub-goals:

1. **reasonable ordering** - consider two goals A and B. According to the reasonable ordering heuristic, B is ordered before A if it cannot be achieved without negating A. In this scenario, solving A first would be meaningless as its result would be undone.

2. **irrelevance ordering** - if reasonable ordering is not applicable to A and B, then the irrelevance ordering heuristic evaluates the number of irrelevant actions for each sub-goal. If B has **less** irrelevant actions than A, it is considered a more difficult sub-goal that should be achieved first. Thus B would be ordered before A.

3. **precondition ordering** - if neither of the above applies to A and B, then precondition ordering compares the minimum number of preconditions required to achieve each sub-goal by considering all the actions that achieve them. Following the idea that complex goals should be ordered first, B is ordered before A if it requires a greater number of preconditions.

4. **random ordering** - if none of the above apply, then A and B are ordered randomly.

At the start of any search, the global ordering of all sub-goals are randomly generated while ensuring that the above heuristics are satisfied for all partial sub-goals.

While these heuristics provide an ordering that helps minimise the number of inconsistencies between sub-goals, it does not guarantee that all conflicts are avoided. To resolve these inconsistencies, the concept of Extended Saddle-Point Condition (ESPC) [88] is used to evaluate the satisfaction of global constraints that ensure goals do not conflict. The ESPC is used to identify saddle-points in the Langrangian space of a planning problem, which correspond to the local minima of a constrained planning problem. SGPlan recursively computes a global plan and evaluates the feasibility of the solution using ESPC. The values of Lagrange multipliers, used to weight the objective and constraints of the

problem in the Lagrangian function, are adjusted according to the feasibility of solutions. This recursion terminates when the Lagrange multipliers converge to stable values, which corresponds to the satisfaction of the ESPC.

In the local sub-goal planning level, each sub-goal is solved by further decomposing it into smaller sub-problems based on the difficulty of solving the sub-goal. These are referred to as intermediate goal agendas, and are solved as individual planning instances by applying a planner such as FF or LPG. SGPlan takes advantage of the partitioned structure of the planning task to improve planning performance by applying a pruning procedure prior to this step. Pruning is used to reduce the search space of each sub-problem by removing irrelevant actions that would otherwise have likely been relevant to the overall planning task. It achieves this by recursively examining each atom inside an *open* list (initially containing only the sub-goal atoms), and maintaining a list of relevant actions (initially empty). As each atom is examined, all relevant actions are added to the relevance list and those supporting atoms that are preconditions to these actions are added to the open list if they have not already been examined. This process is called *backward relevance analysis* and outputs a list of all relevant actions once the open list becomes empty.

2.2.1.2.4 LAMA The **LAMA** planner [89] was developed around the Fast Downward system [90], a forward state-space search that first translates a problem represented in PDDL into the *multi-valued planning tasks* representation. Fundamentally, this converts implicitly represented constraints into explicit constraints.

In the LAMA planner, two types of heuristics are used to direct the state space search. The first type called the *landmark heuristic* involves the use of **landmarks**, which are defined as facts that must be true at some point in any valid plan. Since each of these landmarks must be true at some point in time, an estimate of the remaining goal distance from any state can be obtained by the estimated number of landmarks that has yet to be achieved at the evaluated state. The landmark heuristic is thus evaluated as the sum of the *unaccepted* landmarks and *required-again* landmarks. The *unaccepted* landmarks are all landmarks that have not yet been true at some point along the path to the evaluated state. The *required-again* landmarks are all the landmarks that have been achieved previously along the path to the evaluated state, but which is false at the evaluated

state yet must be true in order to satisfy the conditions of the goal state or achieve another landmark later in the plan. The second type of heuristics employed is a modified FF heuristic that estimates the goal distance through the use of *relaxed* tasks. While the FF planning system uses the plan length (i.e. the number of steps) as the heuristic, LAMA takes into consideration the summation of action costs (and the costs of action preconditions) added together with the plan length when evaluating the modified FF heuristic. This mechanism balances the effort to explore the search space to find a valid plan against the objective to seek a low-cost solution, which is essential in complex planning tasks where purely seeking a low-cost solution may hinder the effort to find a solution at all.

When performing forward search, rather than aggregating the two types of heuristics directly, LAMA uses a *multi-queue heuristic search* strategy that consists of maintaining two separate open lists, one for each heuristic. Both heuristics are computed for all expanded states and stored in their corresponding open lists, while the search algorithm switches between these lists based on priorities assigned in the search.

LAMA uses two approaches to perform forward state-space search. At the start of any planning task, a greedy best-first search is applied to quickly find an initial feasible solution by always expanding the state with the lowest combined heuristic value. Once a state has been expanded, it will not be revisited during the greedy best-first search. When two states have the same heuristic value, the search chooses the state that can be reached using a less expensive operator (e.g. by comparing the last action needed to reach the state).

Once a valid solution is obtained, the LAMA planner seeks to improve the quality of the solution by reiterating through the planning task using the weighted A* algorithm. Recall that the A* algorithm evaluates the cost of a node by the sum of the cost to reach the node from the initial node and the heuristic estimate of the cost to reach the goal. In weighted A*, the heuristic value is scaled by a weight that is reduced over each iteration. Thus over long iterations the weighted A* relies greater on the true plan cost up to the considered state over the heuristic estimate of the goal distance. Furthermore, the weighted A* permits revisiting a state after it has been expanded if a less costly path to the state is found during a search. In this way, LAMA provides anytime planning by continuing to find better quality solutions until a termination criteria is met.

2.2.1.3 Combined Task and Motion Planning

The task planning methods described in Sections 2.2.1.1 and 2.2.1.2 approach a planning task in a purely symbolic form and provide plans that satisfy symbolically defined constraints (i.e. action preconditions and goal states). However, there are limitations to solving a planning problem purely in this discrete, symbolic state space when considering the nature of many robotic domains.

Consider, for example, the common action *move* that requests a robot to move from configuration A to configuration B. Symbolically, the preconditions may only require that the robot is at configuration A. However, the true feasibility of this action depends on additional geometric and spatial constraints such as joint limits, collision avoidance and reachability of configuration B. Attempting to represent these constraints using the standard PDDL representation of preconditions would require a discretisation of configurations, which for high-dimensional spaces is generally infeasible [91]. Furthermore, the effect of moving obstacles in the space of the robot is difficult to capture in symbolic form as it involves spatial and geometric reasoning (i.e. motion planning theory). One workaround involves searching for a complete plan in symbolic state space and sequentially evaluating the feasibility of individual actions in the plan using a motion planner. When an infeasible action is found, the task planning domain is updated and re-planning is performed to obtain a new task plan. This would repeat until a fully feasible plan is obtained. Problems arise when dealing with high-dimensional spaces as this strategy quickly becomes highly inefficient or even intractable. Moreover, it would be impossible to *optimise* a task plan based on costs derived from the continuous configuration space without involving spatial reasoning.

Combined task and motion planning (CTMP) comprises of methods that seek to address these challenges through the integration of motion planning in **continuous configuration space** with task planning in **discrete symbolic state space**. In this section I summarise a number of key developments in the body of literature devoted to CTMP. Note that this body of literature primarily addresses problems comprising of manipulation tasks and therefore involves complex, high-dimensional search spaces. In these problems, it is generally not possible to prove the non-existence of solutions as an exhaustive search through all state expansions cannot be performed in practical times. As a result, all competitive planners to date are at best probabilistically complete and simply reports failure to find a solution at time-out.

In [92], a hierarchical planning in the now framework that interleaves planning with execution was proposed. This planner constructs a plan at a high level abstraction and commits to it by solving for the first sub-goal, and then the next and so forth. Primitive actions are executed as they are found, progressively achieving sub-goals of the task and continuing to *plan in the now*. This planner, however, assumes that actions are reversible such that backtracking can take place when the robot has committed to an infeasible plan. Another notable consequence of the approach is that it sacrifices optimality by commiting to a plan solved at a highly abstracted level. [93] performed CTMP by precomputing motion graphs and collision tables for a mobile manipulator, which were used in lookup procedures within a classical AI planning stage. During the compilation phase, the workspace was discretised to consider virtual object locations that were later used to transform object references from a virtual base reference to its real-world reference. Manipulator trajectories and mobile base trajectories were considered separately. A key contribution to the effectiveness of this approach was the elimination of motion planning and collision-checking during actual planning. However, the quality of solutions found were limited by the number of samples considered in the compilation stage.

Other approaches have emerged that begin to address aspects of probabilistic completeness and generality. aSyMov [94], a pioneering work in CTMP literature, integrated the Metric-FastForward (Metric-FF) planner [84] with motion planning functions derived from Move3D. aSyMov expanded several linked probabilistic roadmaps (PRMs) for individual agents and objects to reduce the number of DoFs considered in each roadmap. This was performed in tandem with planning at the discrete task level, with a certain bounded computation time devoted to roadmap expansion whenever this step was performed. The choice between advancing an action to create a new state and expanding the roadmaps was determined by a cost value derived from a heuristic cost and the number of action failures. A key feature of aSyMov is the preservation of invalid actions, which may later become valid through continuing expansion of roadmaps. This behaviour provides probabilistic completeness as aSyMov balances between finding a plan with the current level of geometric knowledge and exploring the free configuration space. [95] presented a CTMP interfacing layer developed with general applicability to off-the-shelf task and motion planners. In this approach, instantiations of pose references in the task planner were linked to continuous variables

in the geometric environment through an interface layer. An initial task plan was generated from setting ground atoms to default values. By employing a motion planner, the interface layer determined the feasibility of each action in the initial plan. In cases of infeasible actions, the interface layer updated the task planning definition by identifying the obstructions to these actions due to movable objects. This procedure iterated over all possible instantiations of pose references until a solution was found.

The authors in [96] proposed a probabilistically complete planner that applies an incremental approach to constraint-based planning to dynamically provide the task planner with motion feasibility information. This approach was later extended into the Task-Motion Kit (TMKit) [97], a general-purpose framework for CTMP. The TMKit interfaces the symbolically-defined task domain with the geometric-relational properties of the motion domain through a domain semantics layer. The domain semantics describe the conversion between geometric references with discrete task states and relates task actions with corresponding motion plans. This enhances the generality of the framework by enabling additions to actions and domains without strictly requiring changes to the high-level framework.

The FFRob [98], [99] is an integrated task and motion planner that extends the FastForward heuristics used in symbolic planning to robot motion planning. The approach consists of a preprocessing stage that generates a roadmap structure, referred to as a *conditional reachability graph*, to sample a subset of robot configurations, manipulator grasps and object placements. By integrating these two components, the planner was shown to perform efficiently for tasks involving the rearrangement of many objects. The original FFRob [98] was not a probabilistically-complete planner as it relied on offline sampling to generate the necessary conditional reachability graph. This was later addressed in [99], which extended the FFRob to iterate between the sampling and planning phases such that further symbolic actions were generated when a discrete search of the existing finite sample set was insufficient to produce a solution. The authors further presented the *extended action specification* (EAS), derived from simple action specification (SAS+) to extend the expressiveness of the planning representation to include condition evaluations. The authors showed its application to pick and place planning, where the validity of a movement action was evaluated using the EAS representation. Through this framework, FFRob provides an integrated

search that focuses the symbolic planner with geometric details efficiently.

2.2.1.4 Task Sequencing

An interesting yet highly relevant sub-problem of task planning is the problem of task sequencing. Unlike the general task planning problem where a large set of actions are considered, the task sequencing problem considers only visiting a set of task locations with no consideration for what action takes place at each of these *task space* points. These problems can exist where the goal is to perform the same action at multiple locations, or where a continuous action takes place throughout the entire task. An example of the former is an industrial robot tasked with performing a number of spot welding or drilling operations on a component, while an example of the latter includes a mobile surveillance robot that continually monitors and inspects its surroundings while following a tour around the environment.

In a standard task sequencing problem, the objective is to find an optimal tour around all the task space points that minimises a cost function. Common criteria for optimisation include task execution time, which is strongly correlated to efficiency and throughput of a system (this has often been cited as the most important key performance indicator in industry [100]), energy consumption, clearance from obstacles and curvature level (e.g. the frequency of sharp turns, which is challenging for non-holonomic mobile robots). Since the cost of performing an action is associated to each task point and is independent of the visiting order of tasks, the optimisation lies solely in how these tasks are sequenced. This problem draws parallels with the classic algorithmic **Travelling Salesman Problem (TSP)** [101] in the field of Computer Science. The TSP describes the problem of a salesman who must travel between a set of N cities using the shortest route possible. There are no precedence constraints on the order of cities that must be visited, but he should only visit each city once and return to his starting location after visiting all cities.

At first glance, the standard TSP appears to be an effective means of formulating a task sequencing problem. However, its use is accompanied by a number of caveats that must be considered when solving a task sequencing problem. First of all, solving a TSP generally requires the computation of a *pairwise distance matrix*, which stores the cost to move between any two points considered in the TSP. Standard applications of the TSP use fast-to-compute metrics such as Eu-

clidean or Manhattan distance as the cost measure, which is manageable for a large number of sequencing points. In the robotics domain, these simple measures can only provide an approximation of the true cost to move between two task points, as this is determined by the actual motion required to realise a movement between two task points. In complex or cluttered environments, path planning is a necessity to determine a cost that takes into consideration obstacle avoidance and robot kinematic constraints. This becomes computationally heavy for even a moderate number of task points if solved by brute force search, as the path between every pair of points must be computed. Fortunately, when dealing with the MTP problem for mobile robots, the number of points considered for sequencing is generally few, making such an approach viable and indeed preferable thanks to the higher quality solutions provided by computing true motion costs.

The sequencing problem becomes notably more complex when dealing with the RTSP [102] for serial manipulators with arbitrary number of DoFs. Consider a conventional 6-DoF industrial robot. Generally speaking, this kind of robot is able to reach any single task space point using 8 different configurations (with the exception of points near singularities). This capability to reach a point in multiple ways is termed *kinematic redundancy* and provides a robot with a degree of flexibility to evade obstacles while reaching a desired task point. This opportunity to leverage the flexibility of manipulators also introduces greater levels of complexity in the sequencing problem. Now, the problem of determining an optimal sequence of motions to visit a set of task points involve choosing a configuration that helps minimise the overall cost of execution. Solving the RTSP using a standard TSP formulation for the set of task space points would not take into consideration the varying costs associated with using different configurations for each task point. Since the actual cost of motion between any two points is determined by the chosen C-space configurations, we cannot expect a high quality solution for complex problems using this approach.

The Generalized TSP (GTSP) conveniently provides a modified formulation of the standard TSP that fits this class of sequencing problems well. In the GTSP, rather than considering a salesman that must visit every city in the problem, the salesman must now visit one (and only one) city in every state considered in the problem, where each state contains a discrete number of cities [103]. Like before, the salesman must return to his starting location at the end of the tour (see Fig. 2.12 for an illustrative example of the distinction between GTSP and

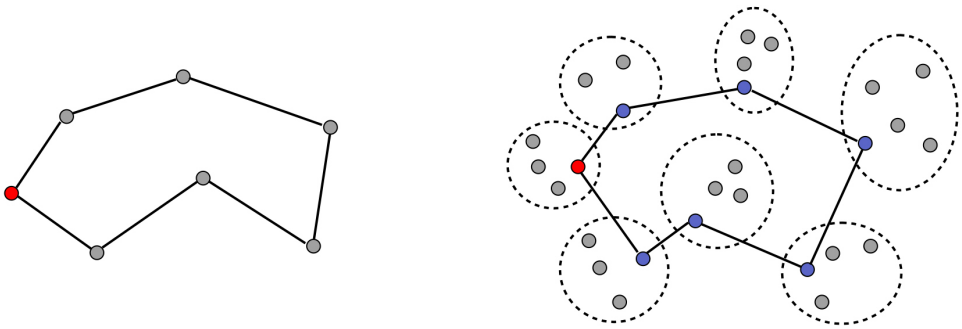


Figure 2.12: Illustration of the TSP and GTSP. Red nodes represent the starting city and solid lines represent the tour, (a) the standard TSP, (b) the GTSP, where dashed ellipses represent the grouping of nodes and blue nodes represent the visited node within each group.

TSP.) Framed in the context of RTSP, every task point forms a *cluster* (the state), containing all the configurations that can reach that point (the cities in the state). The objective is to find a sequence of robot motions that starts from a home configuration, reaches exactly one configuration within each cluster contiguously, and then returns to the home configuration. Authors in [104] have indeed addressed RTSPs as a GTSP, demonstrating the possibility to solve the problem in this way. However, their investigations highlighted the complexity of the problem, as reported results showed long computation times for problems containing a small number of task points. Furthermore, the trade-off between using simple cost metrics for faster computations versus considering true motion costs (via path planning) for higher quality solutions still applies in the GTSP formulation. In fact, in the latter case, the method of solving the RTSP aligns with the *multi-goal path planning*[7] formulation described by Wurll et al. [105], where the authors referred to a GTSP involving obstacle regions in the search space that the GTSP tour should not intersect. Interestingly, the use of the term MTP in mobile robot sequencing literature somewhat overlaps, albeit with the possibility of only one point being associated to each task rather than the multiple configurations associated to a single task point considered in RTSPs.

In certain applications involving higher kinematic redundancy, the set of feasible robot configurations for an arbitrary task point may be defined by a C-space *region*. Take for example the problem of remote laser welding, involving a multi-

[7]Be careful that the definition of this term used here differs slightly from its appearance in the context of task sequencing problems for mobile robots.

DoF scanner mounted on a robotic arm. The scanner possesses adjustable mirrors and variable focal length. Thus it is possible to position a laser beam at a target point from among many configurations within a bounded region determined by the specification of the scanner system. In these kind of scenarios, one may approach the problem as a TSP with Neighbourhoods (TSPN). TSPN is yet another extension of the GTSP, where instead of clusters containing discrete points, the problem involves visiting at least one point from within each region. This time the constraint to visit only one point per task (as in the GTSP formulation) is relaxed as it may be necessary to pass through a region in order to reach a chosen configuration and subsequently move to another region [106]. Despite the convenient formulation of TSPNs for RTSPs of this nature, solving the problem directly becomes exceedingly complex when obstacle avoidance is also taken into consideration. This was highlighted in [107], where the TSPN formulation was applied to RTSP. The authors noted that obstacle avoidance had not been considered directly in their method. Instead, they reported the use of post-processing to correct for any collisions in the TSPN solution.

Finally, I wish to clarify the difference between the use of the terms TSP (and its variants) and MTP/RTSP. While both generally considers an equivalent problem, I use MTP and RTSP as terms to describe real problems that must be solved through planning, while TSP and its variants are **formulations** of these problems to enable a way of solving them. However, it is entirely possible to solve an MTP/RTSP without adopting TSP-based formulation. Several authors have, for example, shown the feasibility of applying the Genetic Algorithm (GA) to solve the same type of problems. I go on to explore these related works in further detail within the context of MTP and RTSP in Chapters 4 and 6, respectively.

2.2.1.5 Key Findings

Table 2.2 summarises the strengths and limitations of each method discussed in relation to the specific form of task planning addressed. CSP and PDDL are two successful approaches for addressing **symbolic task planning**, where all the possible states of the world are contained in a finite set and the feasibility of actions are determined by the satisfaction of symbolic (and in some cases numerical) pre-conditions. The CSP approach has proven to be an effective method for solving problems efficiently, but suffers from a lack of expressiveness for representing the set of all actions in a compact and standardised way. In contrast, PDDL

Table 2.2: Fundamental Task Planning Techniques

Problem	Method	Strengths	Limitations
Symbolic task planning	CSP	• Capable of solving large problems very quickly compared to conventional search methods • Gives insight into how an invalid solution violates constraints	• Difficult to represent certain rules and constraints in the real-world • Large number of variables required to represent the complete set of possible actions • Domain values and constraints must be specified uniquely for each problem
Symbolic task planning	PDDL	• Compact representation of domain actions • Standard representation for all problems • Human-understandable language • Able to represent many problem features	• Large complex search space • Not as computationally efficient to solve as CSPs • Unable to account for continuous search spaces
CTMP	Planning in the now	• Efficient planning for complex problems • Interleaves spatial reasoning to provide feasible actions	• Requires actions to be reversible • Unable to provide high-quality task plans • Cannot support dynamic task planning
CTMP	aSyMov	• Probabilistically-complete planner • Combines motion planning knowledge with task planning layer	• Does not provide optimal task plans • Requires extended planning time to alternate between task planning and motion planning • Cannot support dynamic task planning
CTMP	TMKit	• Probabilistically-complete planner • Generalised framework for any robot • Allows for the addition of new actions and domains without changing the framework	• Does not provide optimal solutions • Cannot support dynamic task planning

Continued on next page

Table 2.2 – continued from previous page

Problem	Method	Strengths	Limitations
CTMP	FFRob	• Support condition evaluations • Efficient integration of motion planning with symbolic task planning • Probabilistically-complete planner	• Does not provide optimal solutions (but shows sign of possible extension to compare solution quality) • Cannot support dynamic task planning
Optimal task sequencing	TSP	• Simple formulation • Many algorithms exist to solve classic TSPs • Simplest form of task sequencing to solve	• High time complexity (exponential in practice) • Unable to account for kinematic redundancy
Optimal task sequencing	GTSP	• Suitable for robots with finite kinematic redundancy properties • Reverts to a TSP for problems involving only one configuration for each task point	• A significantly more time-consuming problem to solve compared to TSPs • High time complexity (exponential in practice) • Requires sampling for domains involving continuous state spaces, which may fail to find the optimal solution
Optimal task sequencing	TSPN	• Suitable for robots with infinite kinematic redundancy properties • High flexibility for addressing different sequencing problems • Capable of optimising solutions for continuous state spaces	• Exceedingly complex to solve - long planning times required even for small problems • Intractable for large problems

stands out as a highly expressive language that provides a consistent approach to representing a vast range of planning problems while being able to support a large range of features that would otherwise be very difficult to represent.

CTMP is an extension of symbolic task planning, where considerations for motion planning in continuous C-spaces are combined with the discrete state space in task planning. Task planning for manipulator robots have been investigated extensively for the study of CTMP, and indeed a number of different methods have been proposed to interleave motion planning with task planning for this domain. However, these research efforts have been dedicated to the development of probabilistically-complete planners for finding **feasible** solutions to highly complex planning problems. Yet there exists an opportunity to leverage the concepts of CTMP to enable optimal task planning and facilitate adaptive re-planning capabilities.

Task sequencing represents a special case of task planning, where the only actions of interest consist of moving between goal points in the task space. Though it shares much similarity to the class of TSPs, it carries increased complexity due to the phenomena of kinematic redundancy and the necessity for collision avoidance. This makes finding optimal task sequences very tricky due to the large search space for these kinds of problems. While feasible solutions can be obtained quite easily in this domain (e.g. by solving it as a standard TSP), achieving optimal solutions is deceptively difficult. GTSPs and TSPNs provide a more effective formulation for these problems and provide the opportunity to use kinematic redundancy to improve task plans. However, these also carry increased complexity that manifests in substantially longer computation times. A key research challenge therefore lies in reducing the computation time required to obtain high-quality solutions for sequencing problems involving both obstruction and kinematic redundancy.

In the remaining sections of this chapter I will describe how these techniques and methods are built upon by the current state-of-the-art to enable optimal and adaptive task planning in the domains studied in this thesis.

2.2.2 Task Planning for MWRs

The concept of guiding a task planner with geometric knowledge inferred from motion planning is not unfamiliar in the mobile robot planning domain. Numer-

ous works have tackled mobile robot task planning as an MTP problem, whereby solutions to individual path planning instances between pairs of goals were used to determine an optimal visiting sequence of all goal locations. [108] presented one example in which the authors treated the problem as a generic TSP and solved it using the Genetic Algorithm (GA) applied to a graph representation of the task. The GA chromosome represented a visiting sequence of goals and nodes, evaluated according to idle time and energy consumption required to perform the sequence. One caveat of their work was the assumption that the actual paths between nodes and corresponding energy required to execute such motions were known a priori, with no consideration for how to obtain these efficiently.

The work presented in [109] similarly applied a TSP representation for MTP under uncertainty. Here motion planning was achieved through an offline-generated feedback-based information roadmap, which consisted of nodes (representing robot states) and collision-free edges that span across the robot's configuration space. During online planning, new nodes could be inserted into the roadmap to improve its coverage when new information was discovered. One drawback of this method was the high computational cost associated with the use of roadmaps. It is generally time-consuming to generate a roadmap for large environments and must therefore be restricted to offline pre-processing. Furthermore, large roadmaps consume significant memory due to the exponential increase in the number of roadmap edges and are generally unsuitable for large, high-dimensional or complex planning problems.

A self-organising map approach to solve polygonal MTP was presented in [110]. In polygonal MTP, goal locations were specified as a goal region rather than a single point, and only one point within each goal region must be visited. The authors applied a two-layer network to incrementally improve the arrangement of a pre-defined number of nodes to visit each goal region. This self-organisation is based on the concept of advancing winning nodes towards a corresponding goal centroid, which was determined from a shortest path approximation. Much like the work in [109], the approximation of shortest paths from any point in the search space to each goal region were pre-computed. Therefore, although authors reported fast planning times for considered problems, the long pre-processing time required for evaluating path feasibility and cost was not considered.

The use of multiple RRTs have proven advantageous in other works for applications such as in [52], where a multi-RRT approach was used to explore the

search space for finding a flight path to maximise visual coverage of a pre-defined manifold. Likewise, the work in [53] presented a multi-tree extension of the T-RRT algorithm (Multi-T-RRT) for feasible path planning in MTP problems. Through the use of the *transition test*, the Multi-T-RRT sought paths that favored lower path cost values, but the algorithm did not provide any optimality guarantees.

Indeed MTP cannot represent more general mobile robot task planning problems as they fail to support additional actions and task dependencies. Numerous pieces of existing work addressed this through the use of PDDL for general task planning problems in the MWR domain. One recent example was presented by Crosby et al. in [3]. Here the authors introduced an integrated mission and task planning framework consisting of a top-level mission planner that allocated abstracted tasks to a fleet of robots. A local task planner that adopts PDDL planning was used to solve each set of tasks independently to generate a more detailed set of action plans for each robot to execute. Notably, the framework plans entirely in the symbolic state space, with the allocation of tasks in the mission planner solved in relation to the individual skill set or capabilities of each robot and the type of objects involved in the tasks. At the level of task planning, the authors noted that re-planning (from scratch) took place when a state of failure was encountered during execution. While this behaviour is useful for *recovering* from failures, in the majority of applications we would normally prioritise *avoiding* failures where possible. When considering obstacle uncertainties for example, collision may result in damages to the robot or colliding object, which could lead to downtime or early system failures over extended operation and should be avoided entirely.

In [111], Zhang et al. demonstrated how heuristics could be combined with PDDL planning to enable the search of solutions that meet application-specific needs. The authors introduced the concept of plan explicability and predictability to describe how easy it is for a human to understand the intentions and predict a robot's behaviour from observing its actions. Using heuristics that numerically expressed these concepts, a modified FF planner that was guided by these two concepts was used to find plans that were optimised for better explicability and predictability.

The authors of [112] described the integration between a PDDL task planning layer with a *semantic* information layer, which represents the relationship of

objects as a hierarchy tree (e.g. the house contains a kitchen, which contains a fridge and table). Through an interface between these two layers, the search space of the task planner could be reduced significantly by filtering out irrelevant information through the semantic hierarchy tree. Starting from a categorical plan constructed from the semantic hierarchy, AI planners (such as the Metric-FF) were recursively used to solve multiple planning sub-problems at various abstract levels to finally obtain a solution to the problem. The authors explained that the generation of semantic information was achieved through the use of sensors for navigation and object localisation. However, their method does not capture the geometric relationships of such objects within the task planner. As a result, this approach was unable to optimise a task plan according to the cost of execution.

So far the work described above fail to account for the geometric knowledge of the environment require for optimal task planning. This gave rise to the opportunity to exploit the strengths of CTMP to improve the capability of mobile robot task planners. The Unified Path Planning and Task Planning Architecture (UP2TA) [113] pursued this direction for rover-based mission planning. The UP2TA is composed of a PDDL task planning layer that integrates a deterministic path planner (such as the A* [33] and Theta* [34] algorithms) to find optimal plans. This approach consisted of a fast greedy algorithm for approximating shortest path lengths between key locations, which were used to guide the task-level deliberation layer. Once a task plan was obtained, a path planner based on exact algorithms was invoked to determine the true motion path for each movement action in the symbolic task plan.

The concept of task re-planning is not entirely new territory. For example, Cirillo et al. [114] considered the problem of *human-aware* task planning, which dealt with applications where the robot must take into consideration the presence of humans (e.g. in households and offices). The authors proposed a planner that, at its core, relies on a knowledge base of detectable human actions. Given a set of possible human plans (e.g. in the form of a schedule of the human's tasks during the day), the planner computes a task plan for the robot that avoids interference with the human's anticipated tasks using PTLplan [115], a probabilistic, conditional, temporal-logic planner that was developed to solve planning tasks that involve a set of scenarios with associated probabilities of occurrence. Re-planning takes place whenever the robot detects a change in the intended actions of the human to avoid interference. One limitation of this work is that only those actions

that have been provided in the knowledge base are detectable, which does not accommodate for unforeseen events. Other similar work on human-aware task planning can be found in [116, 117].

The authors in [118] explored the application of PDDL combined with motion planning for manipulation and navigation re-planning. Using the FF-planner to solve task planning queries, a reachability graph was initially constructed using the solution to an initial task planning problem. Optimal solutions to individual motion planning queries for each action in the reachability graph was obtained by applying lazy kinodynamic motion planning by interior-exterior exploration (L-KPIECE) [119], a motion planner developed for systems with complex dynamics. When changes in the environment or failure was detected, the algorithm generated an alternative plan by sampling a new symbolic state and performing a sub-task planning query to connect this state to the reachability graph. A key feature of this method was the capability to compare alternative plans and select the one with the least *effort* to achieve the goal during the re-planning stage. For example, when a movable obstacle is encountered, the planner would compare the option of avoiding the obstacle against transferring the obstacle to another location and choosing the option that required less effort. Although this approach enabled the planner to choose the better plan from among multiple feasible solutions, it did not provide any guarantee for optimality at the level of task planning, as it did not integrate geometric knowledge within the task planner itself (i.e. motion planning was performed **after** a task plan was obtained).

Key Findings The limitations of each of the above algorithms are summarised in Table 2.3. As we have seen, MTP methods approach the task planning problem from the direction of low-level motion planning, where the objective is to determine the most effective paths for navigating between goals. However, all of these methods only supported simple task planning requirements and could not accommodate additional planning features such as a wide set of actions and task precedence constraints.

PDDL-based approaches on the other hand naturally cope well with these requirements thanks to the expressiveness of the language, but they are generally incapable of providing high-quality solutions that account for the cost of robot motion. Various different mechanisms have been proposed in the literature to determine a better plan from among several plans, but these have been inadequate

Table 2.3: Limitations of Existing MWR Task Planning Methods

Ref.	Method	Problem	Limitations
[108]	GA	MTP	• Assumed known paths between every goal a priori • Only able to solve the MTP problem, not general task planning • Not suitable for dynamic environments due to pre-computed paths between goals
[109]	TSP + roadmap	MTP	• High computational cost • Requires offline pre-processing • Large memory required to store roadmap
[110]	Self-organising map	MTP	• Requires pre-computing approximate shortest paths from any point to goal • Very expensive offline pre-planning required • Cannot be extended to dynamic environments due to pre-computed paths
[53]	Multi-T-RRT	MTP	• Does not possess any optimality guarantees • High memory requirements for extended iterations
[3]	PDDL	Task planning	• Unable to re-plan to adapt to uncertainties in the task or environment • Plans in purely symbolic space • Does not return optimal solutions • No guarantee that symbolic actions are feasible at the execution-level
[111]	PDDL + intention-based heuristics	Task planning	• Does not account for geometric knowledge when optimising task plans • No guarantee that symbolic actions are feasible at the execution-level
[112]	PDDL + semantics	Task planning	• Does not account for geometric knowledge - no optimisation performed • Requires multiple calls to AI planners - expensive for large state spaces • No guarantee that symbolic actions are feasible at the execution-level
[113]	PDDL + Theta*	Task planning	• Requires discretisation of environment • Not scalable to high dimension problems • Requires explicit knowledge of obstacles

Continued on next page

Table 2.3 – continued from previous page

Ref.	Method	Problem	Limitations
[114]	PTLplan	Task planning	• Supports adaptive planning to some extent but does not support unforeseen events (relies on a priori knowledge of possible events) • Does not account for geometric knowledge - no optimisation performed
[118]	FF-planner + KPIECE	Task planning	• Considers alternative plans to identify better solutions, but does not actively search for an optimal task plan • Infeasible actions in symbolic domain not known until after task planning - re-planning until a feasible solution is found can be inefficient for complex problems

for finding globally optimal solutions. I highlight the work in [113], which instead followed the direction of CTMP to combine motion planning with task planning. Their algorithm demonstrated the capability to achieve optimal task plans by incorporating estimated motion costs derived from motion planning into PDDL. However, the ongoing challenge in this direction lies in devising more efficient approaches that scale well with the dimensionality of the problem. Even among MTP methods that focus purely on the motion planning aspect of task planning, efficient motion planning remains a challenge due to the number of path planning queries that must be solved for multiple goals.

This thesis presents work that advance the state-of-the-art in this direction. Specifically, Chapter 4 introduces a new method for integrating motion planning with PDDL to solve task planning problems for MWRs. This work is inspired by CTMP methods but, as discussed in Section 2.2.1.5, the concept of informing a task planner with spatial reasoning is transferred to the domain of MWR to enable the computation of **optimal** task plans. This work follows an approach similar to UP2TA, but differs in its use of a novel sampling-based motion planner that maintains high planning efficiency for high dimensional C-spaces. Focusing in particular on Research Question 2 (Section 1.2.3) experimental evaluations are performed to quantify the performance of the algorithm in terms of planning efficiency and solution quality.

The work reviewed in this section have also shown that the practicalities of some approaches were limited by the high memory resources required by the

algorithms. Thus throughout the analysis of the presented algorithm, I give particular attention to the viability of deploying the algorithm to lightweight robots with limited computational power, which is a core research issue captured by Research Question 4.

2.2.3 Robotic Task Sequencing

To highlight some of the ongoing challenges in the RTSP, this section provides a concise summary of the progression of research developments found in literature to date. Dubowsky and Blubaugh [120] were one of the first authors to consider the use of TSP concepts for solving sequencing problems in robotics. They addressed point-to-point tasks for an industrial robot arm and formulated the problem as an asymmetric TSP, where the cost of moving between two task points accounted for the influence of gravitational forces. In this early piece of work, the authors did not consider the kinematic redundancy of the robot, and thus the problem of configuration assignment had not been addressed. Likewise, the work in [121], which studied a robotic sequencing problem for a fruit-harvesting manipulator robot involving the use of TSP, had arbitrarily assigned a single configuration to each task point. As a result, neither of the algorithms used in these works provided a means to reduce the cost of a solution through the consideration of assigning alternative configurations to task points.

The work presented in [104, 122] was one of the earliest works to consider multiple IK solutions when solving RTSP. The authors formulated the problem for a 3-DoF robot as a GTSP and solved it using the Traveling Salesman Algorithm [123]. While this study proved the feasibility of the approach, the dimensionality and scale of the problem considered were too small to be of relevance for practical applications. Building upon this direction, Kolakowska et al. [124] tackled the RTSP for a 6-DoF robot, where they proposed a multi-objective optimisation formulation that is subsequently solved by a constraint optimization model. Reported results suggested that the method was viable for problems involving up to 10 task points, but for larger problems the computation time required became impractical (extending up to several hours for tasks involving up to 12 task points and an upper limit of 6 IK solutions per point).

Some authors have approached the problem from a different direction and formulated the RTSP as an optimization problem solved using the Genetic Algo-

rithm (GA). Vitolo et al. [125] tackled a task sequencing problem for an optical inspection system where each area requiring inspection could be observed from any point within a system-defined task region. Any robot configurations that positioned the on-board optical inspection system within this task region would satisfy the requirement to inspect the given target area. Interestingly, rather than explicitly considering the sets of IK solutions for each task region, the authors computed multiple numerical descriptors such as pose quality, pose reachability and collision to collectively evaluate each task region. Nevertheless, a TSP formulation based in the task space was adopted and solved using the GA. Unfortunately, it is difficult to assess the performance of their proposed approach as the authors did not provide a comprehensive evaluation of the described method. The work in [126] also applied the GA to solve RTSPs. Here the authors proposed a GA encoding that captured both the assigned configurations for each task point and the task space tour of the points. In this way an optimal solution to the RTSP that directly considered the kinematic redundancy of the robot could be obtained by applying the GA. The authors reported planning times of approximately 1,800 seconds for a problem involving a 6-DoF robot and 50 task points.

Kovác [107] addressed the sequencing problem in the context of robotic laser welding applications. In his work, a modified TSPN was used to model the problem in task space, which was solved within a fixed allocated planning time. In the experiments, a planning duration of 600 seconds was used to tackle problems of up to 71 task points. This work assumed the absence of obstacles within the operational workspace of the robot taking into consideration the nature of a purposely designed laser-welding cell, but the author acknowledged that this is often not the case in practice. The author went on to show that when design needs prioritised other objectives, the approach would require post-processing to check for collisions and correct infeasible paths.

In all of the work discussed thus far, none had taken into consideration the aspect of collision avoidance when solving the task sequencing problem. Contrastingly, Jing et al. [127] treated the RTSP as a combined Set Covering Problem (SCP) and TSP (SCTSP), which was subsequently solved using a Random-Key GA. Their method involved travelling cost evaluations that used motion planning queries to compute motion plans for all pose-to-pose pairs (only one configuration, obtained by sampling, was considered for each task point). The exact costs of

these paths, which captures the true motions required for navigating between any two configurations, were used to directly inform the SCTSP planner. While the authors had considered mostly clutter-free environments, their approach overcame the limitations of simple distance metrics providing poor estimations of motion costs required for collision avoidance. Despite this, computation times of approximately 3 hours were reported for an obstacle-free environment involving up to 400 task points. Unsurprisingly, it was shown that the most time-consuming steps corresponded to the computation of pose-to-pose motion plans. The authors had also presented an extension of this work to consider multiple IK solutions for each task point [128]. In this case, the problem was formulated as a combined SCP and GTSP. The efficiency of the approach had not been reported for the trials conducted in their study, but one could expect planning times to be at least several folds greater than their original work due to the exponential increase in the number of pose-to-pose pairs introduced.

It is clear that an exhaustive search involving the computation of true motion costs for all possible pose-to-pose pairs is impractical as it scales poorly with the dimensionality and size of the planning task. In view of this, Spitz and Requicha [129] proposed the use of a probabilistic roadmap to establish connectivity between task configurations and subsequently solving for a tour of points in the C-space. However, the authors did not consider the case of multiple IK solutions. Rather, the method had been demonstrated on a problem in which each task point was already assigned a single configuration, with no indication of how these configurations were chosen. The authors of [130,131] extended their work in [126] to account for obstacle occupancy information in the 2D and 3D space, respectively, by adopting a *bump surface* concept. The distribution of the obstacles in the search space was captured as a single mathematical entity and encoded into a single objective function that is solved using a GA. However, in the experiments reported in their papers, only tasks involving up to 15 task points were solved using this method.

In all of the aforementioned work, a key limitation is the long, and sometimes impractical, computation times required to solve problems involving a relatively moderate number of task points. In real-world applications, a task sequencing problem can often involve hundreds to thousands of task points, which simply cannot be solved efficiently using many of the work described above. In constrast to these work, Gueta et al. [132] proposed the use of clustering to reduce the

planning time required to solve larger sequencing problems. Given the set of task points for sequencing, their approach consisted of first partitioning the points into a pre-defined number of groups (or clusters) according to their topological distribution in task space. The 2-Opt algorithm [133], a popular method for solving TSPs efficiently while providing near-optimal solutions, was used to minimise the cost of visiting the set of clusters according to the Euclidean distance metric in task space. Entry and exit points between clusters were then chosen based on closest points between successive clusters, and the Lin-Kernighan heuristic [134] was applied to solve each sequencing sub-problem within the clusters. The assignment of robot configurations to task points was performed once the entire task sequence was obtained.

The more recently proposed RoboTSP algorithm [135] has been demonstrated successfully in solving sequencng problems involving large sets of task points by several orders of magnitude less computation time while achieving plans of comparable quality to alternative methods. The algorithm achieves this by first formulating a TSP problem in task space, which is solved using the 2-Opt algorithm to obtain a task space tour of task points. Once this tour is obtained, the assignment of robot configurations is achieved by generating a graph where nodes represent the IK solutions of task points, while edges connect configurations that correspond to the immediate predecessor/successor task points in the task sequence. A graph search is performed using the Dijkstra's algorithm to find the best assignment of configurations for the given task sequence. Like many other previous work, however, the limitation of this method lies in its inability to account for obstacles when determining an optimal sequence as it relies upon simple distance metrics applied in the task space for ordering points. Therefore, even though the algorithm is able to find the best assignment of configurations for a given task space tour, the tour itself can often be sub-optimal in the presence of spatial constraints.

In [136], authors presented an Equality GTSP formulation to the RTSP and applied the Lin-Kernighan-Helsgaun (GLKH) solver to compute optimal task sequences for tracing open and closed contours using a remote laser processing system. Here the objective was to optimize the sequence of entry and exit points to sequentially access each contour, where open contours could only be accessed through its end points, while closed contours could be accessed from any point along the contour. Using a redundant 7-DoF system, sampling was used to gen-

erate 100 configurations for every viable entry and exit points in the task. By applying the GLKH solver, a tour across the access points (alternating between entry and exit points) could be optimised taking into account the assignment of the best configuration for the overall task. Importantly, the authors recognised that the GLKH solver did not provide a guarantee for globally optimal solutions. Instead, the GLKH solver provides incrementally better solutions by iteratively solving the task sequencing problem. For a study involving a combined total of 52 access points (both entry and exit) and 3000 iterations for the GLKH solver, an approximate computation time of 15 hours was required to obtain a solution. These long planning times were also noted in the benchmarking studies conducted in [135]. It can be concluded, therefore, that GLKH solvers offer the potential to find high quality solutions (even among obstacles), but are strictly limited to expensive offline planning.

For a more comprehensive review of existing methods for RTSP, I refer readers to [102], which covers literature up to and including 2014.

Key Findings The limitations of existing work on RTSPs have been summarised in Table 2.4. While a different approach has been used in each of these works, their limitations broadly remain the same. With the exception of [135], all of the reported methods were evaluated on small sets of task points that do not accurately represent the scale of planning problems for many real-world applications. Consider for example the use of a point probe to carry out high-resolution measurements of a component. A large distribution of points across the surfaces of the component would be necessary to achieve a detailed profile. This is a common requirement for applications such as in-service inspection, quality monitoring and failure analysis, where problems can often involve several hundred to thousands of points. Though various authors have demonstrated the viability of their methods to find optimal solutions to small sequencing problems, the long computation times (of up to several hours) reported prevent these algorithms from scaling well in practice for numerous real-world applications. This is also why these methods are restricted to offline planning.

The work in [132] and [135] are exceptions, as they have been shown to provide solutions considerably faster. The RoboTSP method [135] in particular stands out for being able to solve problems involving several hundred points in the order of minutes while being able to provide near-optimal solutions in relatively clutter-

Table 2.4: Limitations of Existing RTSP Methods

Ref.	Method	Limitations
[120]	Asymmetric TSP	• Only considers one configuration per task point • Fails to find optimal solutions due to failure to account for kinematic redundancy
[121]	TSP	• Only considers one configuration per task point • Fails to find optimal solutions due to failure to account for kinematic redundancy
[104, 122]	GTSP	• Only evaluated on a 3-DoF robot - does not scale well with robot DoF • Method has only been evaluated on a very small set of task points
[124]	Constraint optimisation	• Only evaluated on very small task sets • Very long computation times (several hours for simple problems)
[125]	GA	• Does not account for motion planning and collision avoidance • Evaluation results not available - performance not proven • Does not explicitly consider the set of IK solutions
[126]	GA	• Have only been evaluated for problems involving up to 50 task points • Long computation time (1800 seconds for 50 task points) • Does not account for motion planning and collision avoidance
[107]	TSPN	• Assumes the absence of obstacles in the environment • Only evaluated on relatively small task sets (<100 points) • Relatively long computation time (600 seconds for 71 points)
[127, 128]	SCTSP	• Requires computing motion plans for all pose-to-pose pairs between every task point • Very computationally expensive - requires approximately 3 hours when not accounting for kinematic redundancy • Planning becomes intractable when accounting for kinematic redundancy for large task sets

Continued on next page

Table 2.4 – continued from previous page

Ref.	Method	Limitations
[129]	TSP + PRM	• Does not account for kinematic redundancy • No clear approach to choosing one configuration for each task point prior to applying the algorithm • Memory-intensive for motion planning using PRM
[130, 131]	GA + *bump surface*	• Method has only been evaluated on very small task sets (up to 15 task points) • Poor scalability
[132]	CTSP	• Sequencing problem solved in task space - can result in sub-optimal solutions • Task sequence determined without considering kinematic redundancy
[135]	RoboTSP	• Sequencing problem solved in task space - can result in sub-optimal solutions • Task sequence determined without considering kinematic redundancy
[136]	GLKH	• Does not guarantee a globally optimal solution • Very long computation time (up to 15 hours) • Evaluated on a comparatively small set of task points (up to 52 points)

free environments. This was achieved by solving the sequencing sub-problem in task space, which has the drawback of producing sub-optimal solutions in heavily spatially-constrained environments as it fails to account for problems such as poor reachibility of task points and the possibility of encountering singularities. Thus the improvements in planning efficiency are offset by poorer robot performance during execution.

Achieving an effective balance between finding high quality solutions and minimising the computation time of algorithms remains an ongoing challenge in RTSP literature. This is particularly true for problems involving large sets of task points and complex spatial constraints. Chapter 6 of this thesis is devoted to these research challenges, where I present a new method for solving static RTSPs with a particular focus on improving planning efficiency without impairing the quality of solutions obtained in spatially-constrained environments. This work serves as an important milestone for advancing techniques towards online and adaptive planning by addressing Research Question 3 (Section 1.2.3) in the context of

RTSPs.

2.2.4 Adaptive Task Planning

Whilst the literature discussed until now have touched upon the concepts of adaptive and dynamic planning, this section provides a detailed review of the current state-of-the-art in adaptive planning. Section 2.2.4.1 covers adaptive techniques and methods for general robotic task planning domains, which is of particular relevance to task planning for MWRs investigated in this thesis. Section 2.2.4.2 then narrows down on the RTSP, which is a problem that, to the best of my knowledge, has not been addressed thus far for online and dynamic environments. In fact, the related work discussed for adaptive RTSPs do not adequately address the problem, but rather solves a closely related *dynamic travelling salesman problem* (DTSP).

2.2.4.1 General Task Domains

Enabling adaptive behaviours in robotic planning requires considerations for replanning at two levels: the top-level action sequence and the low-level robot motions for executing any individual action.

Vannoy et al. introduced the real-time adaptive motion planning (RAMP) method [137] to address the motion planning component of adaptive planning in dynamic environments. Inspired by concepts from EAs, RAMP consists of maintaining a population of trajectories (both feasible and infeasible) for a given motion planning problem. At each planning cycle, the algorithm seeks to improve the population of trajectories by iteratively applying operators to seek higher quality trajectories according to a fitness function (the approach closely resembles the strategy adopted by GAs). The fitness function evaluates the quality of a trajectory according to the feasibility of the solution (e.g. the percentage of the path in collision with the environment) and the cost of the path (e.g. length or execution time). During execution, the planner selects the trajectory with the best fitness value and deploys it to the robot. A key feature of this method is the allowance for the execution of infeasible trajectories up until the point of expected collision is reached. Planning cycles are continually performed during execution and a control cycle is used to define the update rate of the executed trajectory. When a better trajectory has been found at a new control cycle, the current

execution is updated with the new trajectory. This enables the robot to adapt to dynamic changes in the environment by switching to a better trajectory when the current trajectory's fitness value deteriorates as a result of new obstacles.

The method is advantageous for its high planning efficiency and its anytime characteristics - that is, the planner is able to begin execution quickly while additional planning time is used to improve the quality of the final trajectory. RAMP was later extended to task-constrained manipulator motion planning in [138]. The planning domain consisted of a robot whose motion was constrained by a given task (e.g. holding a tool or object and maintaining a fixed orientation for the end effector). Where no solutions exist for avoiding the obstacles due to task constraints, the authors proposed the relaxation of constraints to enable the robot to avoid the obstacle before returning to its original task. This was achieved using RAMP by maintaining two populations of trajectories, corresponding to those that achieve the task by following given constraints, and trajectories that seek an intermediate goal location for obstacle avoidance without constraints.

While RAMP has proven efficient for motion planning in dynamic environments, it is limited by the inability of the algorithm to guarantee optimal solutions. Since the quality of trajectories are determined by stochastic operators without explicit considerations for the environment, the rate of convergence to an optimal solution is low. This is particularly true for complex environments where intricate motions are required to overcome obstacles.

The work in [139] presented an alternative approach to motion re-planning for CTMP problems involving dynamically-changing task goals. Rather than explicitly computing a defined trajectory in the planning phase, the authors adopted a reactive control approach that produced the necessary actuation commands to move a robot towards changing goals through considerations at the controller level. Task goals corresponded to goal poses for the robot end effector or manipulated objects, which were defined relative to a target frame in Cartesian space such that any changes in the world pose of the target frame would not affect the representation of target goals. Robot motion was realised through a reactive controller that sought to drive the robot to the defined goal poses according to a sequence of actions obtained through PDDL task planning. This approach resulted in fast, real-time adaptive behaviour to dynamic goals, but fails to account for obstacle avoidance, joint limits and singularities as the actuation of the robot was determined by a control scheme defined in Cartesian space. Furthermore, this

method only supports adaptive motion planning. Task plans were assumed to remain valid (and optimal) across the entire duration of the task, but in real-world applications this assumption generally does not hold true.

James et al. [140] also investigated end-to-end control for the execution of a robot task without explicit motion planning. They proposed the use of convolutional neural networks (CNN) that were trained in simulation to map visual images of an environment to feasible trajectories that accomplish the task. A large set of randomised planning domains were generated in simulation to provide training data that was generalised across different environmental variables such as dynamic lighting and moving objects. Each of these planning domains were solve using conventional motion planners to obtain a feasible solution, which were supplied to the CNN with corresponding images of the environment to conduct training. The authors demonstrated that this approach could enable a robot trained fully in simulation to conduct tasks of the same nature robustly in the real world. The strengths of this work was the scalability of the method and efficiency of the planner once trained for a given task.

This CNN approach to adaptive motion planning carries a number of limitations that drastically affect the practicality of the approach for many applications. Firstly, the performance of the system is highly dependent on the quality of solutions generated for training instances. If the training data do not reflect the current task well (e.g. if the task has not been seen before), the planner may entirely fail to obtain a feasible solution. Secondly, the CNN requires a large set of training data, making the process of training very time-intensive. Whilst generating this data through simulations is comparatively easier than using real-world data, it is still not possible to apply this method to new planning problems rapidly. Finally, the method presented in the paper performs basic motion planning to obtain linear paths between Cartesian points, which can be encoded efficiently without loss of detail in the CNN. However, it is unclear if this extends well to complex, C-space trajectories necessary for obstacle avoidance in cluttered environments.

Focusing on re-planning at the task-level, Zhou et al. studied the problem of human-intention aware task planning and proposed the use of behaviour trees (BTs) to represent a planning domain and problem more conveniently for adaptive re-planning [141]. BTs involve the use of internal nodes to represent logical operators that capture the preconditions of actions and relaying these to leaf

nodes that implement primitive actions such as the actuation of motors in a particular way. This representation streamlines the logical decisions for executing specific actions given a known state of the world. The authors adopted the hierarchical planning in the now framework [92] to map BTs to the planning domain. As actions were executed according to the planning in the now framework, the planner queried the BT to re-establish a new set of actions to reach the goal whenever failed actions were encountered. While this method allows for re-planning in response to changes in the validity of actions, it does not provide optimal action sequences nor guarantee the feasibility of individual actions due to the inherent limitations of the planning in the now framework, as discussed in Section 2.2.1.3. Furthermore, this framework does not account for low-level motion re-planning, preventing the planner from avoiding collision with dynamic obstacles.

Planning in the now was similarly adopted in [142], where a cloud-based framework was used to combine perception information on the wider world with robotic task planning and execution for dynamic environments. While task planning was performed on-board the robot, queries for global knowledge of the world was fulfilled by the cloud. The authors introduced the concept of continual planning in the now, where an original task was divided into sub-tasks. Rather than computing a complete plan to fulfill the requirements of all sub-tasks, the strategy consisted of solving the task and motion planning problem for each sub-task only immediately before they were due to be executed. This reduced the frequency of re-planning for actions that would not be performed in the early stages of a plan. Though this method improved the efficiency of planning in the now for online planning, it also failed to compute optimal task plans and required actions to be reversible to overcome dead-ends. Furthermore, the framework proposed in this work relied upon the cloud to provide processing capacity for perception information, which is not always available for robots deployed into remote environments.

He et al. [143] introduced a task planner developed to accomplish tasks that involved human intervention, where a human was capable of assisting or interfere with the tasks of the robot. As the robot began to execute a planned sequence of actions to accomplish a task, adaptation was required to respond to changing world states that resulted from human actions. The authors proposed a reactive synthesis approach based on game theory to develop planning strategies that guaranteed the success of the task by evaluating all the modelled human behaviours.

The symbolic task planning problem was modelled using a compositional method applied to *linear temporal logic over finite traces* (LTLf) synthesis to capture the task and domain as *binary decision diagrams*. These were combined into a game that was solved to obtained a winning strategy, which was subsequently mapped back into the planning problem. This enabled the robot to adapt actions quickly online by observing the state of the world during execution and performing the corresponding actions based on the winning strategy to ensure the success of the task. The limitations of this work is the absence of spatial reasoning, derived from motion planning, to inform the search for an optimal task plan. Furthermore, planning decisions took place after the execution of each action, but no support for online dynamic obstacle avoidance was provided.

In [144], CTMP was combined with reinforcement learning to address task planning problems with dynamic and uncertain environments. The framework consisted of alternating between an inner and outer loop to progressively improve the quality of solutions found. The inner loop used a conventional task planner that solved the problem in purely symbolic form. The feasibility of the resulting plan was then evaluated using a motion planner. This task-motion sequence was then passed to the outer loop that employed reinforcement learning, where the solution from the inner loop was used to learn rewards based on its quality. When solutions were deployed through the outer loop, new observations of the environment affecting the executed plan would be used to learn new rewards. Any updates to the reward values were fed back into the inner loop to update the quality of actions in the task planner. In this way the planning framework enabled a robot to adapt to dynamic environments while learning higher quality solutions over time. A key drawback of this method is the slow pace of learning that results from incrementally updating the rewards and feeding it to the task planner. This is not of substantial concern for slowly changing environments, but in highly uncertain or rapidly changing environments, the planner would fail to keep up with changes and consequently return sub-optimal solutions, or worse, fail to return a solution. The latter can occur as the task planning layer of the inner loop is not informed by motion planning knowledge and thus fails to identify geometrically-infeasible actions.

Schmitt et al. [145] similarly employed the use of reinforcement learning to achieve adaptive planning in dynamic environments. Focusing on the task of manipulation, the authors represented the C-space as a finite, intersecting set of

state spaces where transitions between a subset of states were mapped to discrete actions. This provided a convenient representation for describing the switch between grasp-free states and contact-constrained states. When planning online, a constraints-based model was used to derive a set of constraint-based controllers that served as the discrete set of actions required for reinforcement learning. Each of these controllers applied an APF-like behaviour for transitioning the robot towards switching states while avoiding collision and satisfying existing constraints. The reinforcement learning agent was trained in simulation to enable the selection of appropriate actions (each corresponding to a single low-level controller) that guide the robot towards the goal state. Under the right conditions, this method can provide fast adaptive behaviours to avoid collision while finding feasible actions that contribute to reaching the goal. However, since this approach does not perform explicit motion planning, the method cannot provide optimal task plans through spatial reasoning. Furthermore, the method inherits the problems of both APF and reinforcement learning, meaning that it can encounter local minima during execution, while an extensive learning phase is required to converge to optimal reward values.

Online re-planning for partially-observable planning problems was investigated in [146], where the task involved a partially-occluded environment formed by objects such as large obstacles, closed doors and walls. Accomplishing a task in this type of environment involves planning actions online to progressively improve the robot's belief space of the world. The authors addressed this problem by employing PDDLstream, a variant of PDDL that supports sampling of continuous variables, to model hybrid belief-state stochastic shortest path problems (SSPP). Two key strategies were introduced to improve the planning efficiency online. Firstly, the planner constrained the structure of new plans to retain the unexecuted actions of previous plans, speeding up the process of solving multiple task planning queries while avoiding the repetition of actions involving random variables. Secondly, the planner postponed the computation of plan parameters that were not immediately required for execution. This minimised the number of computations performed by leveraging the assumption that changes would likely be made to actions that appear in the later steps of a plan due to new observations. Nevertheless, this work did not address the problem of online re-planning in dynamic environments, nor did it seek to optimise a task plan through spatial reasoning.

Table 2.5: Limitations of Adaptive Planning Methods

Ref.	Method	Limitations
[137, 138]	RAMP	• No optimality guarantees • Adaptive motion planning only • Executed solutions may not be fully feasible - forced stops may occur
[139]	Reactive control	• Adaptive motion planning only • Motion planning takes place in the Euclidean space - does not account for joint limits and singularities • Does not address obstacle avoidance
[140]	CNN	• Adaptive motion planning only • Training is required for each new type of task, and large training data is required to achieve good performance • Quality of performance depends upon the quality of solutions for training data • Basic linear motion planning considered - performance of system for complex trajectories unknown
[141]	BTs + HPN	• No optimality guarantees • Adaptive task planning only • Does not account for infeasibility of motions when planning task actions • BTs require careful construction to achieve convergence to desired goal condition
[142]	HPN + task partitioning	• No optimality guarantees • Requires reversible actions • Requires access to the cloud - not practical for remote deployment of robots
[143]	compositional method + LTLf	• Adaptive task planning only • Re-planning takes place only at the end of each action • Does not address re-planning in dynamic environments
[144]	CTMP + RL	• Slow convergence as task planning is not informed by feasibility in motion planning layer • Slow pace of learning is ineffective for highly dynamic environments • Extensive learning for an environment is required for good solutions - not effective for rapid deployment in new environments

Continued on next page

Table 2.5 – continued from previous page

Ref.	Method	Limitations
[145]	Constraint-based model + RL	• APF-like motion planning can encounter local minima problems • Reinforcement learning requires extensive learning for each new planning domain • Task planning not informed by motion costs
[146]	PDDLstream + SSPP	• Does not address re-planning in dynamic environments • No optimality guarantees • Re-planning takes place only at the end of each action

Key Findings Table 2.5 summarises the limitations of the literature discussed above. A key limitation common across the adaptive task planning methods that have emerged in recent years becomes apparent through this review: no method currently exists to achieve adaptive task planning in dynamic environments while at the same time provide optimal solutions. This has important implications to Research Question 2 introduced in Section 1.2.3 - it is generally difficult to achieve both high quality solutions and high planning efficiency in the context of robotic task planning. Evidently, this remains an ongoing challenge in the literature and authors have so far only achieved high planning efficiency for dynamic planning applications, or high quality solutions in extended offline planning.

The research reported in this thesis addresses this knowledge gap through the investigation of task planning for MWRs and the problem of robotic task sequencing for high DoF robots. Chapters 4 and 6, respectively, consider these two problems from an offline planning direction. This enables the algorithms developed in this work to be compared in terms of plan quality against baseline planners in literature. Attention is also given to Research Question 3 in these chapters as I describe techniques and considerations that improve the planning efficiency of the algorithms to better cope with online planning requirements. This is more fully explored in Chapters 5 and 7, where I extend developed algorithms to dynamic environments. In particular, I evaluate the efficiency of these algorithms for re-planning in terms of computation time and demonstrate the capability of the algorithms to maintain *near-optimal* solutions relative to their offline counterparts. The work presented in these chapters push the state-of-the-art, demonstrating how an effective balance between plan optimality and

computational efficiency can be achieved for adaptive planning problems.

Within the scope of dynamic planning, I make a clear distinction between dynamic goals and dynamic environments, both of which constitutes to a dynamic planning problem. As observed in the reviewed literature, some methods have been develop to adapt according to changes in the task goals. For example, the reactive controller method presented in [139] enabled the robot to adapt its motion to successfully manipulate objects where the target is changing in time. This requires the robot to *track* the moving target. This similarly applies to [140]. On the other hand, methods such as those reported in [137, 138, 142] seek to re-plan tasks and motions to *avoid* collision with dynamically-moving obstacles in the environment. In this research, I focus on the latter aspect of dynamic planning involving obstacles that invalidate existing plans and trajectories.

2.2.4.2 Adaptive Robotic Task Sequencing

This section gives special attention to adaptive planning for RTSPs. To my knowledge no existing work in literature directly addresses the RTSP online for dynamic environments. However, one may find somewhat close resemblance of the problem to the *Dynamic* TSP (DTSP), a variant of TSP that has been receiving growing attention in the optimisation community. Let us briefly review some of these work.

One of the first appearances of the DTSP actually came in the form of a vehicle routing problem introduced by Psaraftis in 1980 [147]. Here the problem considered a single vehicle required to respond to dial-a-ride requests with the objective of minimising the vehicle's total journey time and individual customer waiting times. The dynamic nature of the problem appeared through the intermediate requests made during the execution of an existing route, which were dynamically considered as they occurred such that the minimised route was updated to reflect new customer requests. Generalising this, the DTSP describes a problem in which a single agent is required to visit a set of $n(t)$ cities once and return to the starting city, where $n(t)$ is a function of time t. The distance matrix $D(t)$ contains the set of individual distances (or costs to travel) between each city $d(t) \in D(t)$, which are also subject to change with respect to time t. The objective of the DTSP then is to find the minimum-cost route to visit the set of $n(t)$ cities at any given time t subject to the time-varying distances in $D(t)$.

It has been noted by authors that, similar to the DRTSP, many methods

developed to solve the static TSP are generally too inefficient for DTSP due to the computational costs required to converge to high quality solutions [148]. As a result, many DTSP methods generally accept somewhat sub-optimal solutions in exchange for achieving higher planning efficiency for solving TSPs in dynamic scenarios.

Jurjee et al. [149] highlighted in their work the significance of Evolutionary Algorithms (EAs) for solving DOPs as they benefit from the mechanism of exploring the search space through population diversity. This particularly applies to swarm algorithms inspired by nature that involve multiple candidate solutions formed by individual members of a population. The general idea of solving DTSPs using EAs consists of retaining the population used to obtain the optimal solution of a previous search to solve a new instance of TSP when changes are observed in $n(t)$ or $D(t)$. This eliminates the need to start from a fresh search that begins further away from the new optimal solution. In fact as Tinós [150] had discovered through a simulation-based analysis of the DTSP, new best solutions resulting from changes in a TSP are generally not far away from the previous best solution. Following in this direction, the work in [149] subsequently proposed a modified harmonic search algorithm for DTSPs, where a multi-population search strategy was combined to maintain population diversity. Although the work addressed DTSP, the results presented only considered the offline planning performance in relation to the quality of solutions obtained, with no reported results for the planning efficiency of the approach.

In [151], a heterogeneous discrete Particle Swarm Optimization (PSO) algorithm was proposed to solve the DTSP. Normally in a *homogeneous* PSO, all particles that make up the population carry the same values assigned to search parameters that determine the balance between exploration (i.e. searching across the whole search space) versus exploitation (narrowing in on promising regions). However, in the *heterogeneous* PSO, individual particles are assigned different values for search parameters such that different particles exhibit different levels of exploration and exploitation. The discrete PSO is used to solve an initial instance of the TSP, and each subsequent change afterwards generates a new instance of the TSP that must be again solved (the authors referred to each instance as a sub-problem). For all sub-problems after the first, the population of particles used to obtain the previous solution is carried forward as a starting search for the new sub-problem. Thanks to the heterogeneous nature of the PSO, parti-

cles that prioritise exploration enable the algorithm to find the new optimal with improved efficiency compared to starting a fresh search. For a DTSP problem involving 442 cities and 11 sub-problems (i.e. an initial TSP plus ten subsequent changes to the problem), the method required between 11.2 to 108.3 seconds to solve all sub-problems. Note that this variation in time is correlated with the number of iterations permitted for the search (PSO is an iterative method), with more iterations leading to better quality solutions.

Following in a similar direction to the above work, Mavrovouniotis et al. [152] proposed the use of a modified Ant Colony Optimization (ACO) algorithm to solve DTSPs. The standard ACO, inspired by the trail-following behaviour of ants in the real-world, is incapable of solving DOPs as the *pheromones* converge towards a single trail (i.e. the solution) at the end of a search. To address this, the concept of immigrants schemes was combined with the ACO to increase population diversity such that trails from previous searches could be retained and re-used for subsequent searches. In later work published in [153], the same authors demonstrated that an ACO integrated with a local search operator could return a new solution in response to dynamic change in approximately 1.5 seconds for a DTSP containing 200 cities.

Perhaps the closest work in literature to investigate a problem that resembles the DRTSP lies in [154], where the authors addressed a dynamic task sequencing problem for a serial manipulator. Though at first this may appear to directly address the DRTSP, the presented method in fact formulated the problem as a DTSP, giving no consideration to a number of key features that define the DRTSP. The authors proposed the use of a Monte Carlo tree search to generate task sequences for a manipulator that dynamically received new tasks to execute online. However, these tasks were defined by positions which were directly used in a path planner to compute motions without considerations for kinematic redundancy (it is unclear whether planning was performed in task space or C-space). Secondly, in their considerations the environment remained static and only the number of tasks (or cities in the TSP notation) were added dynamically. In other words, the problem reverted to a standard DTSP with the distance between cities determined by a motion planner. Planning times for producing solutions to the sequencing problem had not been reported.

Despite the relevance of these advances in DTSP, it is important to recognise that addressing RTSPs in dynamic environments is a much harder problem in

comparison. In the RTSP, the distance between any two points is not simply a direct line between them as it corresponds with physical robot motion that must adhere to joint limits and avoid collision. Furthermore, kinematic redundancy still applies in the dynamic version of the RTSP, which increases the scale of the problem by manifolds. These two features of the RTSP jointly increase the complexity of the problem exponentially compared to the standard DTSP. Thus the methods reviewed here would not be adequate for solving RTSPs in dynamic environments. The research presented in this thesis specifically targets this knowledge gap and seeks to extend the state-of-the-art in RTSPs to enable adaptive task sequencing. To this end, Chapter 7 presents a preliminary study on Dynamic Robotic Task Sequencing Problems (DRTSPs), where I introduce the concept of partial planning to show how planning problems can be addressed more efficiently in light of the requirements for online planning (see Research Question 3 in Section 1.2.3). The chapter then presents an adaptive RTSP algorithm to address dynamically changing environments, where I also examine the necessary considerations required to apply the algorithm in physical robots (see Research Question 4 in 1.2.3).

2.3 Summary

This chapter has presented a literature review on optimal and adaptive task planning and motion planning, covering both fundamental developments and the current state-of-the-art.

Section 2.1 has been devoted to motion planning algorithms, where methods have been categorised under deterministic methods, sampling-based methods and machine learning-based methods (the latter further comprising of reinforcement learning, learning from experience and evolutionary algorithms). Sampling-based algorithms have emerged as the more reliable approach for their balance between solution quality, planning efficiency and scalability to high dimensional problems. While machine learning methods have demonstrated good potential for planning quickly online through building "experience", they are heavily limited by their poor scalability and requirement for a substantial learning phase. Taking into account the necessity for both performance and reliability in dynamic environments, the research in this thesis follows the direction of developments in sampling-based methods. Section 2.1.4 closed this section with a summary of the

limitations of existing work and goes on to describe how the contributions of this thesis overcomes some of these problems.

Section 2.2 comprises of task planning-related literature. I have introduced a number of fundamental techniques across the areas of CSP, PDDL, CTMP and task sequencing. Given the two main task and path planning problems studied in this thesis, I go on to cover literature specific to optimal task planning for MWRs and RTSPs, highlighting the limitations of these work and outlining the knowledge gaps that are addressed in this thesis with respect to the research questions introduced in Section 1.2.3. Finally, this chapter reviews recent state-of-the-art methods in adaptive planning across various robotic planning problems. Findings showed that planning for high-quality task plans while achieving efficient re-planning at the level of task planning and motion planning remains mostly elusive. In fact, within the scope of RTSPs, no existing work have investigated the problem of online planning in dynamic environments. These ongoing challenges serves as motivation for the developments presented in this thesis, which advance the state-of-the-art towards enabling both optimal and adaptive task planning in dynamic environments.

Chapter 3

Motion Planning in Dynamic Environments: A Case Study

3.1 Introduction

As Chapter 2 has illustrated, solving a planning task for robotic applications is by no means a trivial task. Both motion planning and task planning have largely been addressed as offline planning problems, where extensive planning time is invested into finding a high quality solution to each of the respective problems. Doing so, however, requires that the environment remains static during execution - no external influences should affect the state of the world. This imposes strict limitations on the application of robotics in the real world, where environments are often dynamic, unstructured, uncertain and unpredictable. *Adaptive* robotics capable of dealing with change and uncertainty require online planning strategies that allow the robot's behaviour to adapt in response to perceived changes in the state of the world. This introduces additional algorithmic challenges to an already complex planning task.

Enabling *dynamic* planning at the level of task planning is difficult as it requires a way to efficiently propagate changes to the feasibility of actions across the search space when environmental observations are made. Perhaps because of this the majority of research efforts in adaptive robotics to date has been devoted to the area of **dynamic motion planning**, a comparatively simpler problem that is often considered a sub-component of task planning.

This chapter presents a case study that explores some of the necessary con-

siderations when implementing motion planning for dynamic environments by investigating a dynamic pick and place task as a use case.[1] I examine the challenges that are introduced through this case study and put it in the wider context of both task and motion planning. The chapter concludes with a number of observations that have influenced the design of the planning algorithms presented in Chapters 4-7.

The case study examined in this chapter is set in the context of industrial robotics. The apparent shift towards more flexible and intelligent manufacturing acts as a key driver for adaptive and autonomous robots on the shop floor. Practitioners are becoming increasingly aware of the opportunities provided by human-robot interactions and collaborations to improve the flexibility and efficiency of many tasks, where robots and humans operate in shared spaces. To ensure the safety of human workers in these scenarios, the robot must adequately avoid collision with the human workers in all circumstances.

Various pieces of work in literature have sought to implement adaptive capabilities in robots to enable corrective behaviour to some extent when changes are detected in their environments [156–158]. Nevertheless, this area of research is still in its infancy, and the reliability of dynamic planning capabilities have yet to be proven in the real-world. To better understand the considerations that should be taken into account when implementing motion planning in dynamic environments, a case study was conducted on a pick and place task involving an industrial manipulator set within an environment consisting of a *dynamically moving obstacle*[2]. The robot was required to approach a grasp object, pick it up, and transport it to the placing location while avoiding collision with the dynamic obstacle. Achieving this involved the integration of machine vision, motion planning and robot control.

In this chapter, I first introduce the high-level integrated robotic system for

[1]The case study described in this chapter has been partly published as a journal article in [155], for which I am a co-author. Whilst the article reports research contributions from the perspective of machine vision, my contributions in this chapter lies in the development, implementation and critical analysis of motion planning for dynamic environments.

[2]I differentiate between a moving object and a *dynamically* moving object by the predictability of its motion. For any moving object, if its motion is known a priori it would be possible to determine a collision-free trajectory by planning in space-time (though I do not cover this in this thesis). Conversely, I assume that the motion of a *dynamically* moving object at some time in the near future is unknown and unpredictable. In these circumstances it would only be possible to determine the object's pose at the current time step and *estimate* its motion based on its past trajectory.

performing dynamic pick and place tasks. Sections 3.3, 3.4 and 3.5 then describe each of the core components comprising of machine vision, path planning and adaptive control, respectively, while in Section 3.6 I give the details of the experimental setup. Experimental findings are presented in Section 3.7, while Section 3.8 provides a discussion on the findings of this case study in relation to Research Question 1. Finally Section 3.9 summarises this chapter.

Note that although this chapter addresses the dynamic pick and place task for the study of dynamic motion planning, the validity of results discussed at the end of this chapter are not confined to this domain. Dynamic pick and place simply serves as context for the necessity of adaptive planning. In fact the findings observed in this chapter remain relevant for general adaptive task and motion planning problems and contribute to the development of the algorithms for the two core planning domains studied in the rest of this thesis.

3.2 System Overview

This case study was performed on the KUKA family of 6-DoF industrial manipulators. More specifically, the KUKA QUANTEC KR90 R3100 and KUKA AGILUS KR6 R900 robots were used for implementation. In addition to these, the integrated system comprised of a machine vision subsystem that used two optical cameras for 2D visual detection of the dynamic obstacle, a motion planning module to fulfill dynamic path planning requests online, and a robot control subsystem to execute planned paths on the aforementioned robots.

The **machine vision** subsystem provided the sensing capabilities of the system necessary for understanding how the state of the environment was changing during operation. Using images captured by the optical cameras, image processing was performed to obtain a precise geometric and spatial description of the obstacle in relation to the robot. This information was then transformed into the coordinate frame of the robot and used as an input for a motion planning query.

Each of the two cameras were installed in fixed locations in the environment, and only one image from either camera was required to be processed to determine the geometric and spatial properties of the obstacle at any moment in time. However, two cameras were used to overcome potential occlusion of the obstacle as a result of the robot invading the view of a camera. Thus the two cameras were deliberately installed to provide orthogonal perspectives of the environment

such that one camera would always be able to view the obstacle.

In the **motion planning** module, user inputs provided the pose of the desired grasp object and the required pose of the object after placement. The planner then retrieved the geometric and spatial information of the obstacle from the machine vision subsystem (via TCP/IP communications) to determine the state of the dynamic obstacle in the environment. Using the C-space path length as a measure of path quality, a collision-free path between the pick and place points of the grasp object was subsequently planned in real-time. In this case study, the time constraint for satisfying real-time performance was determined by the average human reaction time, quoted as 180 milliseconds according to [157]. The motion planning algorithm used in this case study was based on the dynamic roadmaps method, which was implemented on the MATLAB software. I revisit this algorithm in Section 3.4.

Once a target path was obtained, the execution of the planned motion was realised by the **robot control** subsystem. It performed path tracking and actuator control to ensure that the robot followed the planned path precisely and safely. For a fixed, unchanging trajectory, this is a well-solved problem and all commercially available robot controllers can achieve good tracking performance. However, when paths change on-the-fly during execution, path tracking becomes a much harder problem due to sharp changes in the required motion resulting from the switch-point between two trajectories. In these circumstances, care must be taken to ensure that the control commands sent to the robot adhere to dynamic constraints (such as velocity and acceleration limits) to avoid damaging the robot. The integrated system used in this case study employed the Interfacing Toolbox for Robotic Arms (ITRA) [159] to enable this adaptive real-time control capability. This toolbox is available as a Dynamic Link Library (DLL) in Windows operating systems, or a Shared Object (SO) in Linux operating systems, and can be called from within various development platforms to establish a connection with a KUKA robot.

3.3 Machine Vision

In this section I give a brief overview of the system used to obtain the geometric and spatial information of the dynamic obstacle in real-time. However, since the scope of this thesis lies in the aspects of planning, I do not give a rigorous

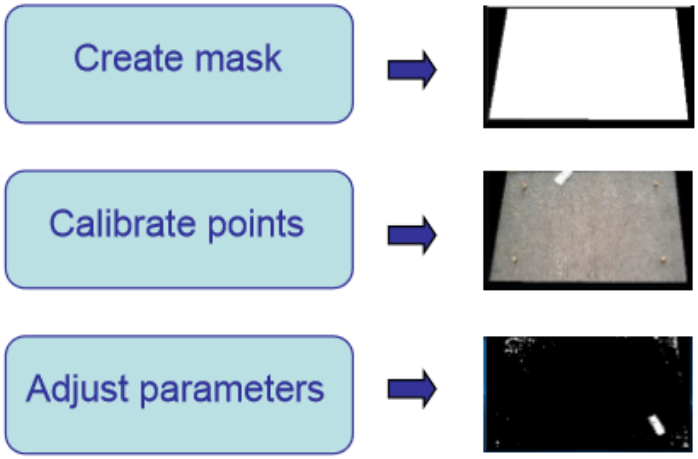


Figure 3.1: Three-step calibration procedure for machine vision image processing parameters. [155]

description of the methods used. Interested readers are directed to [155] for further details of the sensing strategy.

The method for the detection of the obstacle and the retrieval of its spatial information is based on color detection in the Hue-Saturation-Value (HSV) space combined with a number of filtering procedures. Prior to running the machine vision subsystem online, an offline calibration phase is required to configure a number of image processing parameters. This calibration is comprised of three steps: (i) mask specification, (ii) calibration point selection and (iii) HSV parameters adjustment, as shown in Fig. 3.1.

Mask specification is used to define a polygonal mask over the image that specifies a sub-region of interest. This region covers the areas of the image representing the workspace (the 2D view of the task space where the presence of an obstacle could interfere with a robot's task). Once this mask is applied, all pixels in the image that do not overlap with the mask are discarded during online image processing. This minimises the effects of noise and varying lighting conditions on surfaces that lie outside the area of interest within the image. Calibration point selection is used to choose a set of four points for which we know the equivalent true world coordinates relative to the robot coordinate frame. These points are used as reference points for a projection algorithm that transforms any pixel coordinate in the image to its equivalent 2D (XY) spatial coordinates (the detection of variable height of objects were disregarded in this case study as it was assumed a single fixed-height obstacle would be present at any one time throughout the

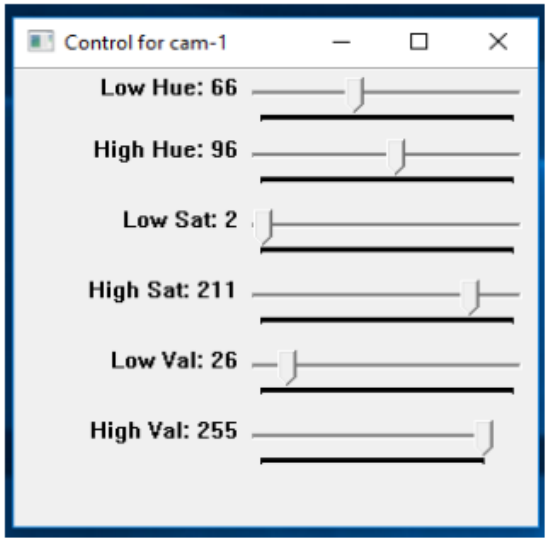


Figure 3.2: Example of a HSV parameter range threshold for color filtering. [155]

experiments). Finally, HSV parameters adjustment is used to specify the range thresholds of HSV parameter values that correspond to the obstacle in the image (see Fig. 3.2). By adjusting these parameters, irrelevant features in the image can be removed to isolate the colour range applicable only to the obstacle. Importantly, HSV was chosen over RGB colour representation as it provides a greater degree of colour specification through the parameterisation of true colour (hue), colour depth (saturation) and colour darkness (value).

Once offline calibration is performed, the online image processing procedure obtains all the pixels corresponding to the obstacle in the image by applying the following steps. First, the standard RGB image obtained from the optical cameras are transformed into a HSV image. Then, using the range threshold defined for the HSV parameters in the offline calibration, the HSV image is converted into a binary image where pixels are assigned a value of 1 if their HSV values lie within the accepted range threshold. Lastly, the resulting binary image is filtered to discard any groups of pixels that do not match the description of the obstacle (e.g. size of a cluster of accepted pixels and the location of the pixels relative to previous observations). From the remaining *cluster* of pixels, the centroid of the obstacle and corresponding bounding box can be estimated in spatial coordinates using the projection algorithm calibrated offline. Fig. 3.3 gives an example of these steps.

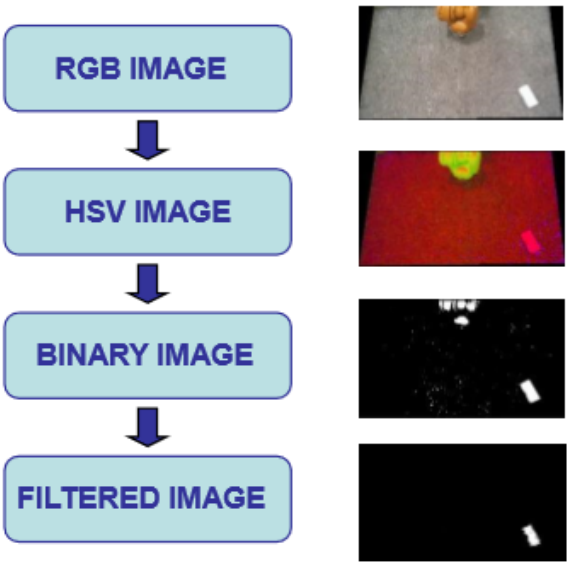


Figure 3.3: Illustration of the image processing workflow and examples of resulting images. [155]

3.4 Dynamic Path Planning

This section describes the implementation of the dynamic roadmaps method, which has been tailored towards the case study's specific use case of pick and place for practicality reasons. MATLAB® was used for the development and implementation of the algorithm within this case study.

The dynamic roadmaps method is a sampling-based real-time variation of probabilistic road maps (PRMs) that have demonstrated effectiveness in motion planning within changing environments [45]. It is characterized by an offline pre-processing phase followed by online planning and execution. During the pre-processing phase, the algorithm creates a mapping between states sampled in the C-space with cells in a discretized workspace. The sampled states are connected with neighbouring states as characterized by PRMs. During online planning and execution, the algorithm iteratively updates the roadmap using the spatial and geometric knowledge of obstacles that lie in the discretized workspace. With this information, motion paths can be planned quickly while taking into account dynamic obstacles. In this implementation, the algorithm is assessed for real-time performance based on its ability to plan paths faster than human reaction time, approximated as 180 milliseconds [157]. This section revisits this method and describes its implementation in detail in the context of the case study.

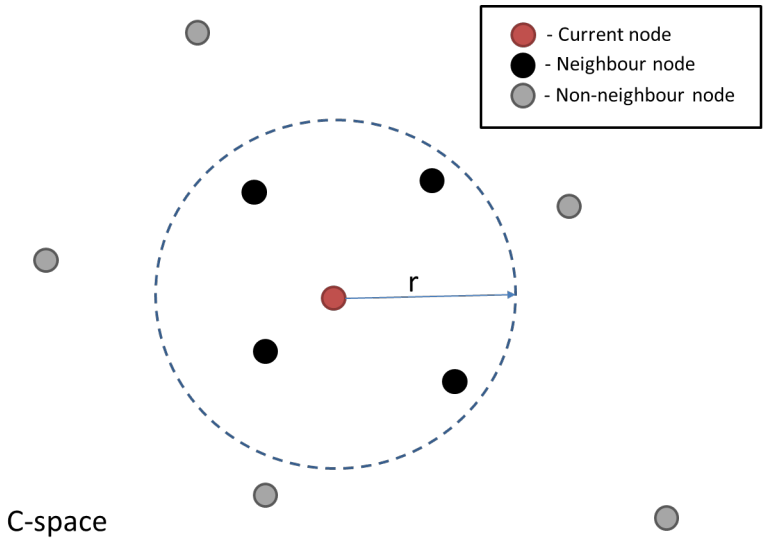


Figure 3.4: Neighbourhood region and corresponding neighbour nodes (shown in black) for an evaluated node at the centre of the neighbourhood region (shown in red). [155]

3.4.1 Pre-processing Phase

3.4.1.1 C-space Sampling

In the normal dynamic roadmap procedure, the first step is to sample a robot's entire C-space such that the robot is capable of reaching any desired goal state, q_{goal}, from a starting configuration, q_{start}, by traversing the resulting roadmap. To form this roadmap, these sampled configurations (also referred to as nodes) are connected to neighbouring samples to form edges. A path from any point in C-space is formed by traversing any number of these edges. Neighbouring nodes are defined as all those that lie within a predefined radius, r, from an evaluated node, as illustrated in Fig. 3.4. For the standard dynamic roadmap method, this sampling procedure would only need to be performed once for a particular robot as it considers only the system parameters of the robot and not its environment. However, this is a very costly process both in terms of time and memory, especially when the required resolution is high as it explores the entire reachable workspace of the robot without consideration for the actual task.

In this case study, two changes are made to reduce the computational resources required for this process. Firstly, instead of sampling from the entire C-space of the robot, only the configurations in which the end effector lies within a pre-defined Cartesian workspace are accepted into the roadmap. Here forward kinematics is used as a fast check to determine which C-space samples meet this

criterion. By doing so, irrelevant configurations that the robot would never reach for the given application are ignored. Secondly, the *spherical wrist assumption*[3] is adopted. By assuming that the robot's 4^{th}, 5^{th} and 6^{th} joints do not affect the Cartesian configuration of the robot (neglecting the orientation of its end effector), it is possible to plan collision-free paths by considering only the first three joints of the KUKA robots. Consequently, the last three joints of the robot are only taken into consideration for the generation of motion paths when the goal orientation is known. By adopting this assumption the dimension of the problem is lowered from six to three, significantly reducing the number of samples required to generate a dense roadmap that effectively explores the search space [66].

3.4.1.2 Workspace Discretization

The purpose of the dynamic roadmap pre-processing phase is to create a mapping between the C-space roadmap and the Cartesian workspace. In order to form this mapping, the workspace must be discretized into uniform cells of a predefined size, n. The smaller the value of n, the higher the resolution, but at the expense of more cells being needed to represent the entire workspace. As the number of cells increases, the computation time required for the pre-processing phase increases exponentially. Hence the parameter n must be carefully chosen to ensure that good quality solutions can be obtained without incurring impractical computational costs.

3.4.1.3 C-space to Workspace Mapping

The mapping between C-space and task space is achieved by recursively performing a *collision check query* between each configuration in the roadmap (corresponding to both nodes and edges) and every cell within the workspace discretisation. This collision check query determines which cells in the workspace would collide with the robot at the evaluated node or edge when occupied with an obstacle. To achieve this, each robot link is modelled as a rigid body represented by point clouds. The spatial arrangement of these links for a given configuration is determined by forward kinematics, enabling the transformation of the point

[3]The spherical wrist assumption states that the 4^{th}, 5^{th} and 6^{th} joint of a 6 DoF robot, where the axes intersect at a common point, only adjust the orientation of the end effector and causes zero translation.

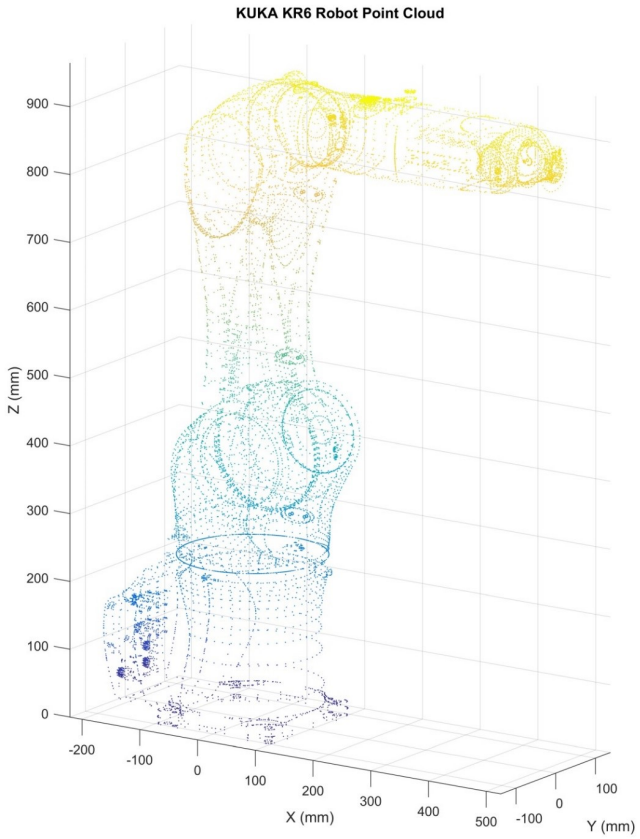


Figure 3.5: Point cloud representation of the KUKA KR6 robot in the home configuration.

clouds to model any configuration of the robot. An example of the point cloud representation of the KUKA KR6 is shown in Fig. 3.5.

To check for collision along a roadmap edge, an interpolation procedure is performed to obtain multiple discrete configurations between the two node configurations that form the edge, with each configuration being tested individually for collision. While linear interpolation could be used to obtain these configurations, it would be difficult to determine an appropriate step size that would not generate an excessive number of configurations nor miss any cells that would be in collision with a short segment along the edge. Instead, the interpolated configurations are obtained as follows. In the first step, a collision check query is performed on the midpoint of an edge. If at this configuration the robot collides with cells not previously encountered at the two nodes that form the edge, then further samples are obtained along the edge by taking the two midpoints from

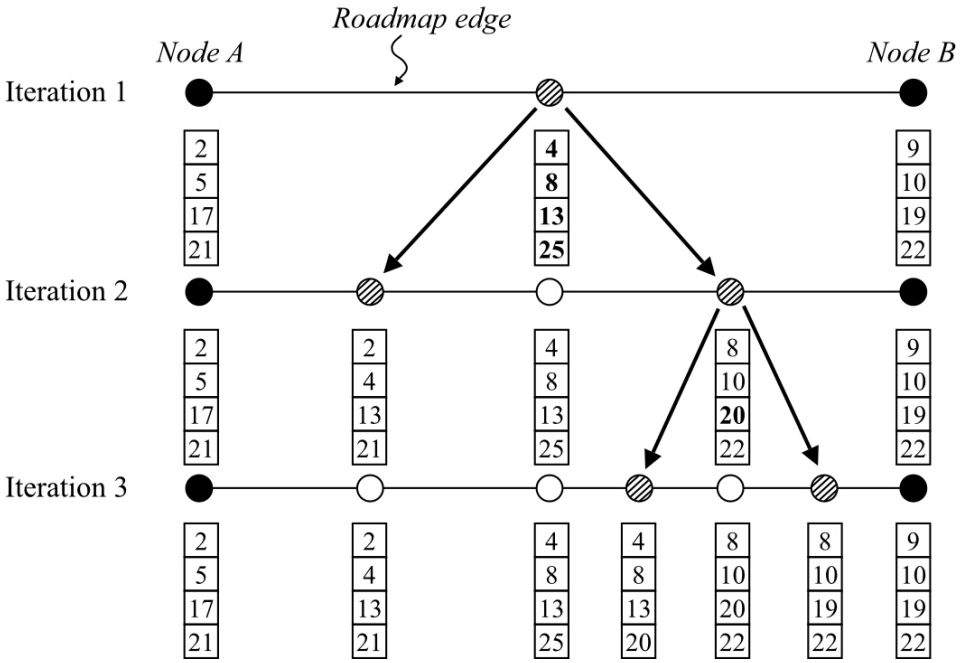


Figure 3.6: Illustration of the roadmap-to-cell mapping procedure for roadmap edges. Nodes with hatch lines correspond to new points along the edge that are evaluated for collision. Squares underneath each point represent a collision between the corresponding configuration and the i^{th} cell indicated by the internal number. Bold numbers show cells that had not been in collision with the configurations checked in the previous iteration. Arrows show the chain of configurations that lead to further evaluation. The procedure ends when no new cells in collision with configurations along the edge are found.

the bisections formed by the evaluated midpoint configuration. These steps are recursively performed until no new cells in collision with the robot are found. Any cell that is in collision with at least one midpoint configuration or with the two node configurations are considered to be in collision with a roadmap edge. An illustration of this procedure is shown in Fig. 3.6.

Performing a collision check query between every cell in the workspace and every configuration[4] in the roadmap has a complexity of $\mathcal{O}(kn^3)$, where k is the total number of configurations in the roadmap and n represents the granularity of the discretization along each Cartesian axis. Thus at high resolutions this mapping procedure is very computationally costly. To improve the efficiency of this mapping, the number of cells checked for collision for any single configuration is restricted to those that are contained within the bounding box of the robot point cloud, as all cells outside this region are guaranteed not to be in collision

[4]When referring to the number of configurations in the roadmap, I consider both node configurations and the interpolated configurations along each roadmap edge.

with the robot at the evaluated configuration.

Every cell, node and edge is given an identifier number. Each time a collision between the robot and a cell is detected, the current node or edge being examined is referenced in a cell database under the index of the cell. By doing so, updating the roadmap during execution becomes a simple process of temporarily removing all nodes and edges belonging to the cells that are occupied by an obstacle in the workspace. Thus at the end of the pre-processing phase, the database would contain a list of all the cells in the workspace along with the edges and nodes that would become invalidated by a cell should it be occupied by a foreign object.

3.4.2 Online Planning Phase

In the online planning phase, the motion planner uses the database and roadmap generated in the pre-processing phase to obtain a collision-free path between any given start configuration q_{start} and goal configuration q_{goal} in real-time. The software architecture for online motion planning is shown in Fig. 3.7. The remainder of this section discusses each component of the software architecture in detail.

3.4.2.1 Updating the Roadmap

For any given motion planning query, the planner first updates the roadmap by invalidating nodes and edges that would be in collision with the detected obstacle in the workspace. The planner requests the current spatial and geometric information of the obstacle represented by a centroid XY position and the dimensions of the unoriented bounding box in the X-Y plane. In this way the planner treats the obstacle as a non-oriented rectangular object, which indeed may not accurately represent the geometry of the true obstacle. However, this over-approximation of the obstacle provides added clearance between the robot and the obstacle, which, for the purpose of collision avoidance, increases the safety of the system.

The set of obstacle-occupied cells in the workspace is determined from the centroid and bounding box of the obstacle. By exploiting the C-space and task space mapping generated offline, all nodes and edges of the roadmap in collision with the approximated obstacle can be retrieved from the database. Updating the roadmap then involves temporarily invalidating these nodes and edges such that during the search phase of the planner, any path through invalid nodes or edges would not be considered. When new obstacle information is retrieved,

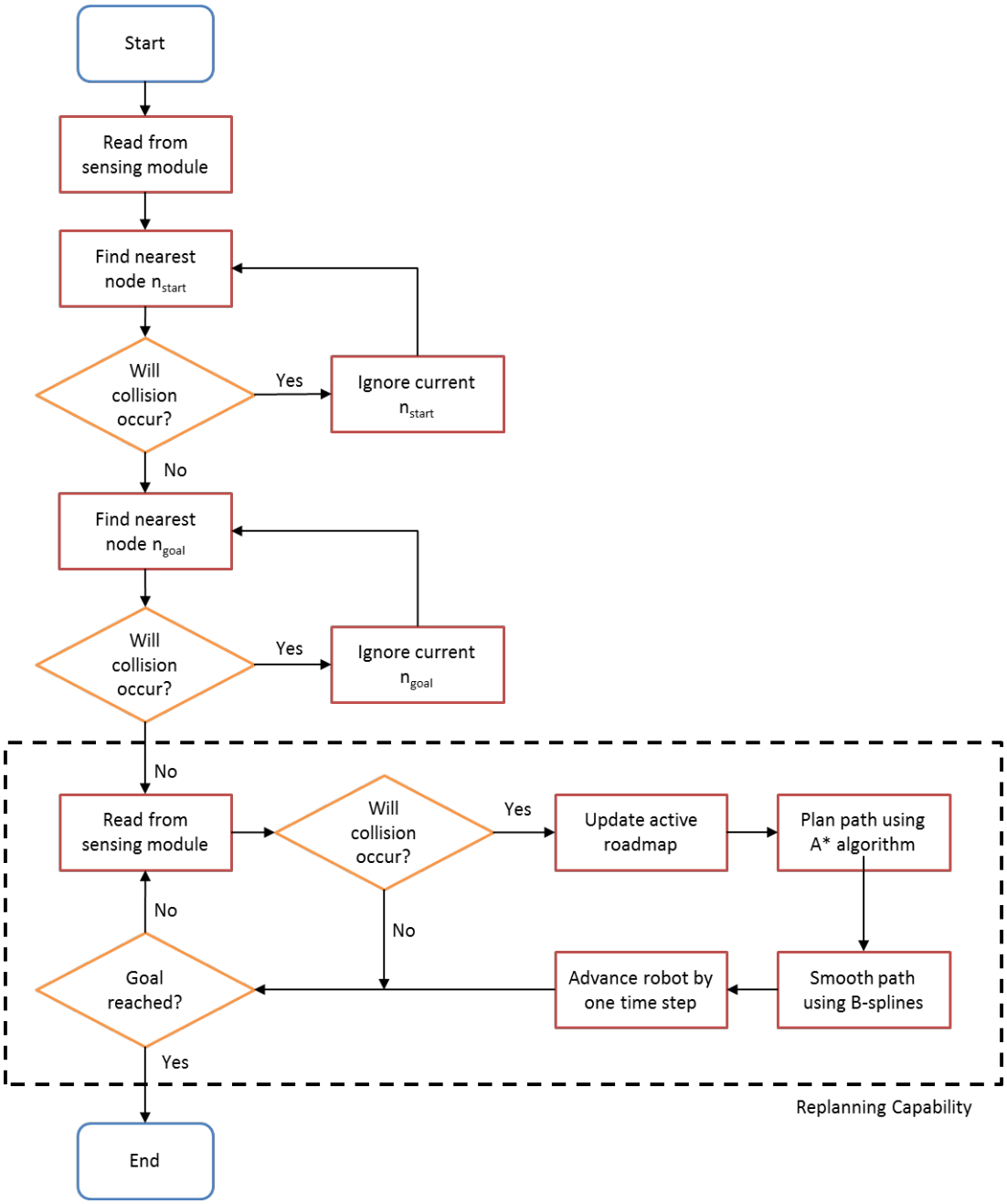


Figure 3.7: Program flowchart for dynamic path planning. Dashed box indicates portion of code responsible for replanning capabilities.

all previously invalidated nodes and edges are returned to a valid state prior to updating the roadmap.

Finally, the roadmap requires the insertion of q_{start} and q_{goal} to enable a complete search from a start to goal state. This insertion involves a search for the nearest neighbour nodes in the roadmap based on the Euclidean distance in C-

space. Using the same neighbourhood region defined in Fig. 3.4, connections between the neighbour nodes and q_{start} (or q_{goal}) are made subject to satisfying a collision check query for each candidate connection.

3.4.2.2 Path Planning

Using the nodes q_{start} and q_{goal}, an A* search is performed on the active roadmap to find an optimal (shortest length) path through the roadmap. Recalling that A* is resolution complete, if the algorithm reports failure to find a solution in the roadmap, then there is no solution that exists within the combined sampled C-space and granularity of the workspace. When this occurs on an initial motion planning query before execution, the algorithm simply reports a failure and terminates.

Dynamic obstacle collision avoidance behaviour is realised through the planner's capability to plan paths in real-time. During execution of an initially planned path, the planner retrieves the updated obstacle information from the machine vision subsystem at each interpolation cycle and determines whether the obstacles in the environment has changed. If a change has indeed occurred, the planner performs collision check queries for the remaining segments of the sent path. In the condition that a collision is detected along the remaining path, re-planning is performed by updating the roadmap and searching for a new path to q_{goal} using the robot's current configuration as q_{start}. If during this re-planning query no solution is found from the A* search, the robot is put into *idle* mode where it would wait for an updated path. The planner would then make recursive attempts to find a new feasible path. Indeed this means that the robot would wait indefinitely if the obstacle continued to obstruct the robot, preventing any further advancement towards the goal.

3.4.2.3 Path Smoothing Using B-splines

Since the A* algorithm searches through discretely sampled configurations, the resulting path generally possesses sharp changes in joint angle velocities at the switch-points between any two edges. This is highly undesirable as it leads to inefficient motion and places unnecessary burden on the motors of each joint. **B-splines** are used to remove these effects by providing a smooth *continuous* path, eliminating any discontinuity in the derivatives (velocity and acceleration)

of the original solution. In the following, I briefly introduce the core concepts of B-splines.

Given a set of points, B-splines generate a smooth curve that approximates these points using a series of n-order curves that connect smoothly to each other. A key characteristic of B-splines is their insensitivity to round-off errors and numerical uncertainty. B-splines achieves this efficiently by employing the use of weighted averages to construct geometrical curves in nature and are conveniently written in terms of basis functions.

Curves created through B-splines always lie within the convex hull formed by control points - these are typically the points that must be smoothed. Smooth curves are produced from convex combinations of polynomial curves through the definition of knots – a way of controlling the weighted average and smoothing of curves. With the right set of control points and knots, it is possible to construct curves of almost any shape.

Letting $\mathbf{M} = (m_i)_{i=1}^{n}$ be the set of n control points and $\mathbf{t} = (t_i)_{i=1}^{n+d+1}$ be the knot vector (a non-decreasing sequence of real numbers), a spline curve $\mathbf{f}$ of degree d is given by:

$$f(t) = \sum_{i}^{n} m_i B_{i,d}(t) \tag{3.1}$$

Where t lies in the interval of $[t_i, t_{i+d+1}]$, and $B_{i,d}$ is the basis function given by the recurrence relation:

$$B_{i,d}(t) = \frac{t - t_i}{t_{i+d} - t_i} B_{i,d-1}(t) + \frac{t_{i+d+1} - t}{t_{i+d+1} - t_{i+1}} B_{i+1,d-1}(t) \tag{3.2}$$

For a detailed derivation of Eqs. 3.1 and 3.2, I refer the reader to [160], which provides a comprehensive description of splines methods.

3.5 Robot Control

So far in my discussion of motion planning I have only described the aspects of path planning. However, to realise the execution of a planned path on a physical robot, the time evolution of the planned geometric path must be specified. This trajectory planning sub-problem can be solved by either the host PC running the machine vision and motion planning algorithm, or locally in the robot controller

that manages the path tracking and actuator control capabilities of the system. In this case study the latter approach was taken to realise *real-time* trajectory generation through an adaptive control approach developed as part of the *ITRA* toolbox.

ITRA provides a library of high-level functions to enable control of physical KUKA robots compatible with the KUKA KR C4 controller. Through this toolbox, three different external control approaches are made available to manage the path tracking behaviour of the robot in response to sent commands. In all three approaches, commands can be sent in either Cartesian or axial (joint angle) coordinates.

In the KUKA Robot Language **(KRL)-based** approach, sent commands are directly translated to target points to reach through a point-to-point (PTP) or linear path (LIN) motion.[5] When multiple commands are sent sequentially using the KRL approach, the controller ensures that each preceding command is achieved before executing the next successive command, forming a first-in-first-out queue that prevents any changes to the robot's actions after the commands are sent. For example, if a robot is commanded to go from point A to point B, and mid-way during its motion a second command to move to point C is given, the robot will first arrive at point B with zero velocity, then execute the motion to move from point B to point C. In this approach, the robot is able to operate at interpolation cycles of either 4 ms or 12 ms.

The second approach is the **computer-based** approach. This approach requires trajectory planning to be performed at the host PC such that a sent command contains the complete trajectory of the target motion. Rather than receiving a single target point, the controller parses through a text file containing a sequence of configurations interpolated at 12 ms (this approach does not support 4 ms interpolation cycles). The velocities and accelerations of each joint (or the end effector when target points are sent in Cartesian coordinates) are implicitly defined based on the interpolation of the geometric path. Like the KRL approach, any preceding commands are always fulfilled before a succeeding

[5]PTP and LIN can often be mistaken to mean the same. This is not true. PTP refers to an optimal movement between any two points, where all joints move synchronously with the velocity profiles determined by a leading axis. This generally translates to a curved path traced by the end effector. Conversely, LIN constrains the path of the end effector to a straight line between two points. While this minimises the distance traveled by the end effector, it is generally not optimal relative to the actuation of the motors.

command is executed. Thus if a second trajectory is sent to the controller while an earlier command is being executed, the trajectory is appended to the end of the previous command.

Both the KRL and computer-based approaches do not support real-time adaptation of a robot path during execution. This means that even if a new path was planned to avoid a dynamic obstacle, it would not be possible to overwrite the previous commands sent to the robot. The Robot Sensor Interface **(RSI)-based** approach was developed to overcome this limitation. This approach exploits the real-time system of the KR C4 (operating at either 4 ms or 12 ms interpolation cycle) to compute smooth trajectories to a target point while satisfying maximum velocity and acceleration constraints. Importantly, the approach is able to handle any arbitrary values of initial velocities and accelerations (granted that they do not exceed the maximum permitted velocity and acceleration). This was achieved by implementing the second-order trajectory generation algorithm developed in [161] within the KUKA controller's RSI system software. With this approach, a new command can be sent to the robot while it is static **or moving**. Any new commands sent while the robot is in motion automatically overwrites all previous commands. Following the earlier example, suppose a robot is executing a command to move to point B from point A. If, mid-way during the motion it receives a new command to move to point C, the controller directs the robot to follow a new trajectory towards point C without reaching point B. This new trajectory is smooth and continuous, which avoids any discontinuity in the executed velocity and acceleration profiles. An illustration is provided in Fig. 3.8.

Based on these capabilities, the RSI-based approach was employed for the execution of geometric paths planned by the motion planner. Once an initial plan is found by the A* search, the path is sent for execution at 4 ms interpolation cycle. During this time, the motion planner retrieves a new observation from the machine vision subsystem and determines whether a re-planning query is required. Whenever a new plan is obtained, the resulting path is directly sent to the robot. The RSI-based control then manages the fast online modification of the robot trajectory to maintain smooth robot motion while tracking the updated path.

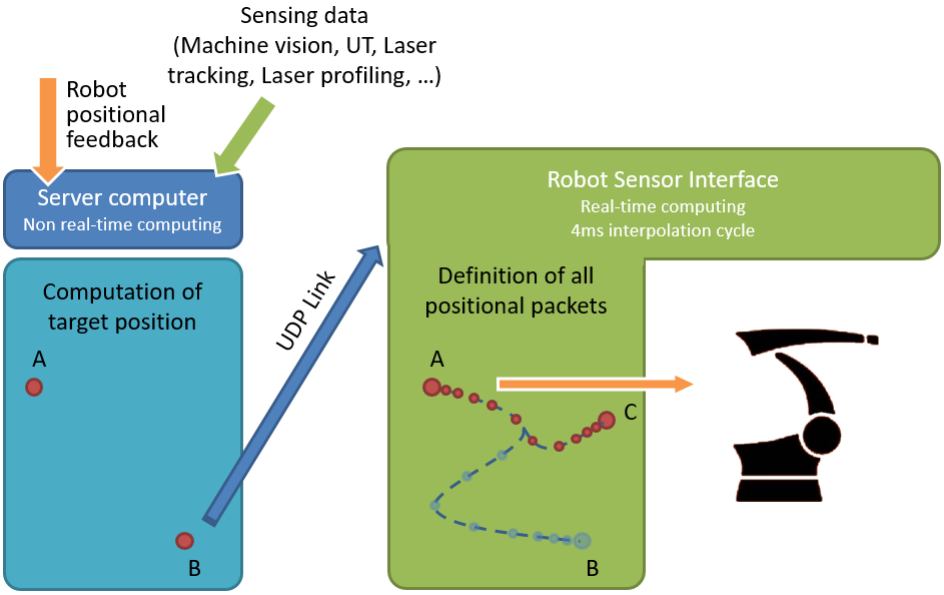


Figure 3.8: The RSI-based external control approach from the ITRA toolbox. When a new target point C is sent to the robot during execution of an existing command to move to point B, a smooth continuous trajectory is obtained in real-time to adapt the motion of the robot on-the-fly. [155]

3.6 Experimental Setup

The integrated system comprising of the described components was implemented on a physical setup within an industrial-like environment. Using the KUKA QUANTEC KR90 R3100 industrial manipulator, a pick and place task was performed to transfer a box between two locations on a table. The KUKA KR90 robot is a heavy-duty serial-link manipulator commonly deployed for traditional industrial applications. A summary of the robot's specification is given in Table 3.1, while Fig. 3.9 describes the geometric properties and work envelope of the robot.

The pick and place task assigned to the robot involved the transportation of a laser-cut box object using a hook end effector, as shown in Fig. 3.10. A number of grasp and placement locations were chosen on the surface of a table measuring 160 cm $\times$ 110 cm $\times$ 85 cm (length $\times$ width $\times$ height) located near the base of the robot. The dimensions of this table defined the workspace of the task used to inform the configuration sampling procedure in the offline roadmap generation phase of the motion planner. On the surface of the table was a radio-controlled car used to simulate a dynamically-moving object. An extended marker was used

Table 3.1: Robot Specification for the KUKA QUANTEC KR90 R3100

Property	Value
Working envelope	66 m^3
Weight	1121 kg
Axis 1 joint limits (max speed)	±185° (105°/s)
Axis 2 joint limits (max speed)	-140°to -5° (101°/s)
Axis 3 joint limits (max speed)	-120°to +155° (107°/s)
Axis 4 joint limits (max speed)	±350° (292°/s)
Axis 5 joint limits (max speed)	±125° (258°/s)
Axis 6 joint limits (max speed)	±350° (284°/s)

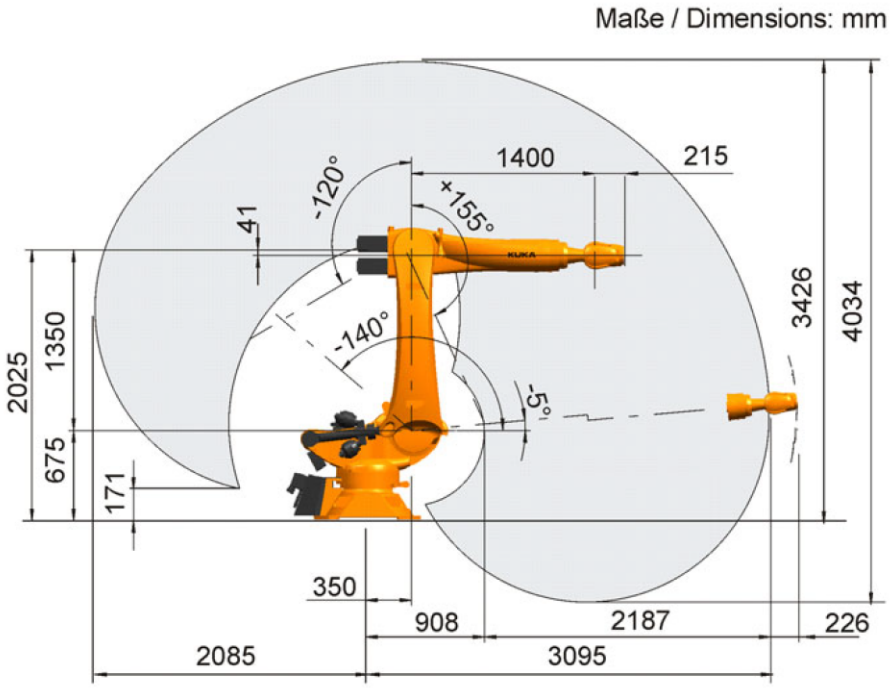


Figure 3.9: The working envelope and geometric description of the KUKA KR90 robot. [162]

to artificially increase the height of the object such that it would act as a moving obstruction as the robot transports the box object between placement goals on the table. After this modification, the dynamic obstacle measured 20 cm × 10 cm × 35 cm (length × width × height) and was capable of travelling at an estimated speed of 1 m/s. Fig. 3.11 shows the table setup and a photograph of the dynamic obstacle used in the experiment.

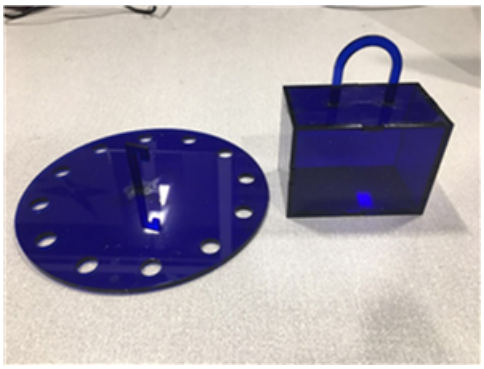


Figure 3.10: The hook end effector and box object used for the pick and place task. Both parts were produced through laser-cutting.

(a)

(b)

Figure 3.11: The workspace of the pick and place task, (a) Table setup showing a grasp location of the box object and the dynamic obstacle placed on top of the table, (b) Dynamic obstacle dressed with a yellow coloured cover and an extended marker for virtually increasing the height of the object.

The machine vision setup consisted of two HD Pro AWCAMHD15 optical cameras possessing a fixed frame rate of 30 fps and a resolution of 640 $\times$ 480 pixels. The first camera (labelled *Cam-1* in Fig. 3.12) was placed 3 meters above the surface of the floor to provide a near top-down view of the workspace. The second camera (*Cam-2*) was used to provide an orthogonal view of the workspace relative to *Cam-1*. The arrangement of these two camera placements enable each camera to view the blind spots of the other camera.

The setup of the experiment described in this section was purposely designed to simulate the harsh, noisy and dirty conditions common in real-world industrial environments. Neither lighting conditions nor reflections were controlled during the experiments. This meant that the ambient lighting, which changed according to the time of the day, introduced varying lighting conditions throughout the

Figure 3.12: Location of two camera installations within the experimental environment. Each camera provides an orthogonal view relative to the other to cover for occlusions caused by the robot invading the view of the cameras.

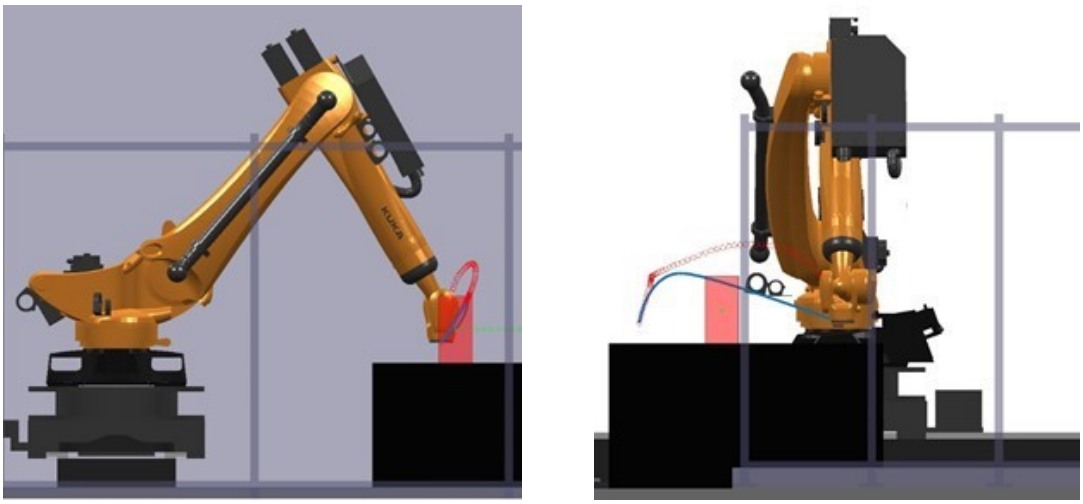
experiments. Furthermore, a non-uniform textured tablecloth was used to cover the surface of the table. This tablecloth introduced speckle reflections that appear as noise to the machine vision subsystem. These considerations are important when evaluating the behaviour of an adaptive robotic system that is expected to perform in environments that are dynamic in nature.

3.7 Experimental Findings

3.7.1 System Performance

The performance of the system was evaluated according to its *reactiveness* to a dynamic obstacle during the pick and place task. Using an Inspiron 15 7000 laptop with a quad-core Intel i7 CPU and 16 GB RAM as the host PC, the computational efficiency of the motion planner is first evaluated in isolation from the machine vision and robot control components through simulation. Afterwards, the computational performance of the complete system is evaluated on the physical setup described in Section 3.6.

A simulation-based environment of the physical setup was modelled within MATLAB to enable virtual simulations of motion planning. The simulation consisted of the KUKA KR90 robot, a table-like object and a box obstacle controlled externally by the user to emulate random motion. The simulation enables the validation of dynamic re-planning through visualisation of the original and updated end effector paths in response to the dynamically-moving obstacle. Fig. 3.13 shows an example motion planning query where the dynamic obstacle (shown as



(a) **(b)**

Figure 3.13: Example of a dynamic motion planning trial, where the blue trajectory represents the initial planned path, while the red trajectory represents the time evolution of the true executed motion. (a) Side view - the path of the red box obstacle is shown by green crosses, (b) front view.

a red box) obstructs the original planned trajectory. The motion planner finds a new collision-free path that directs the robot to move above the obstacle. Notice that the final trajectory is longer than the original trajectory, as the motion planner always seeks an optimal solution. Certainly in the absence of obstacles, this would correspond to a PTP motion.

The computational performance of the motion planner was evaluated using three planning problems consisting of different start and goal configurations selected from within the workspace of the pick and place task, as reported in Table 3.2 (see Fig. 3.14 for a visualisation of the trajectories). Five trials for each planning problem was conducted to obtain a statistical representation of the planner's performance. A breakdown of the CPU time required to perform each high-level procedure within the motion planning algorithm is provided in Table 3.3. Note that in all of these trials, the dynamic obstacle was kept static. Thus the reported CPU time corresponds to the **minimum** required time to compute a single solution (no re-planning took place).

Notice that the *Update Roadmap* procedure consumed the most resources across the complete motion planning workflow. In fact the most computationally heavy tasks correspond to the multiple queries to collision checking for connecting the start and goal configurations to the roadmap (as we will see in the physical

Table 3.2: Planning problems used to evaluate the computational performance of the motion planner

ID	Start x (mm)	Start y (mm)	Start z (mm)	End x (mm)	End y (mm)	End z (mm)	Solution length (mm)
1	10500	-6400	975	10500	-5200	975	1643.5
2	11000	-6400	975	10300	-5200	975	1994.9
3	10200	-5700	975	11200	-5700	975	1981.9

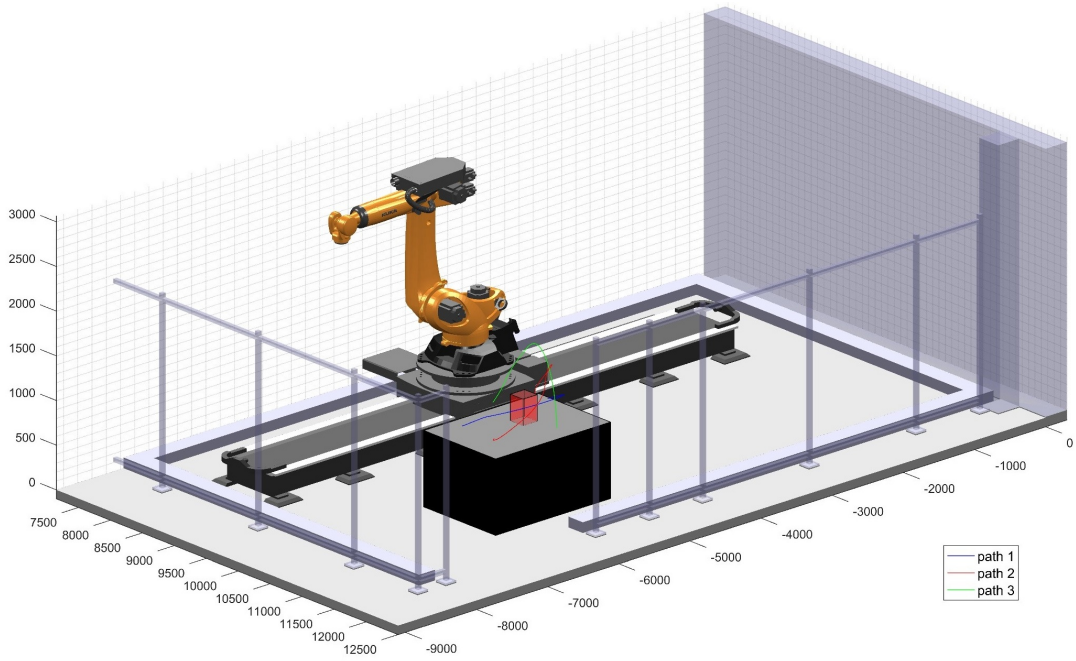


Figure 3.14: Solutions to three different motion planning trials used to evaluate the computational performance of the motion planner under static conditions (obstacle does not move during these evaluations).

experiment). Despite this, by exploiting the offline generated roadmap to explore the environment, the DRM-based motion planner was able to achieve planning times under 50 ms for all three test cases. These results suggest that even under dynamic scenarios the planner would be capable of producing new collision-free paths in well under the average human reaction time of 180 ms. Conversely, if a motion planner that did not involve a pre-processing stage was used, the computation time required to obtain a solution would likely lie in the range of seconds due to the many collision-checking queries involved.

Table 3.3: Breakdown of CPU time for solving a motion planning query (no dynamic re-planning)

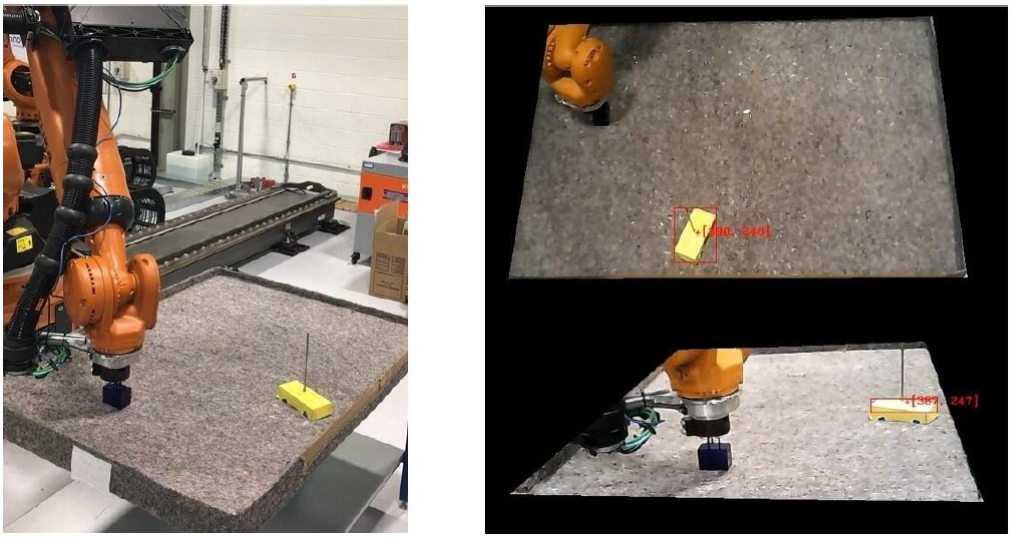
ID	Update Roadmap	A* Search	Path Smoothing	Total
1	30.3 ± 1.6	1.64 ± 0.4	3.06 ± 0.5	35.0 ± 2.4
2	17.8 ± 1.3	8.00 ± 0.3	3.62 ± 0.4	29.4 ± 2.1
3	38.5 ± 3.3	1.36 ± 0.6	2.42 ± 0.2	42.3 ± 3.8

Note: *Update Roadmap* comprises of updating the roadmap using the pre-processed mapping between C-space and workspace, and the connection of the start and goal configurations to the roadmap.

We now move on to the experiment conducted on the physical setup to evaluate the performance of the entire integrated system. While it has been shown that the motion planner is capable of planning a single path in real-time, the simulation results are insufficient for proving the system's response under dynamic scenarios. Let us define the system's **reaction time** as the time from when a change in the environment occurs to when the robot begins to execute corrective action. By this definition, the reaction time takes into consideration the cycle time of the machine vision subsystem, the computation time of the motion planner, and the latency in the robot control module, where latency corresponds to the delay between a command being sent to the robot and the robot beginning to execute the command.

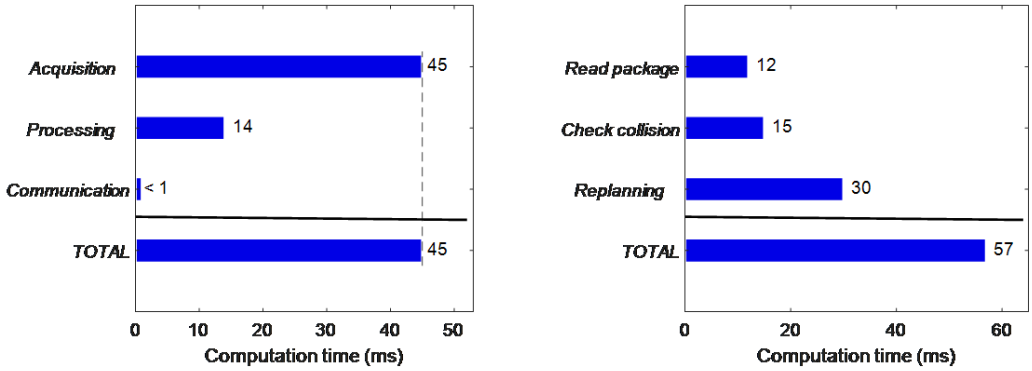
Multiple runs of the pick and place task were conducted on the physical setup. In these trials, the obstacle was radio-controlled by a user externally and, as such, the system did not know the true location of the obstacle at any moment in time. At the beginning of each pick and place task, the obstacle was placed at the outer edge of the table to maximise its clearance from the robot during the initial planning phase (see Fig. 3.15). Once the robot began execution of an initially feasible path, the obstacle was actively driven to obstruct this path, forcing a dynamic re-planning query. Fig. 3.16 reports the approximate CPU times for each high-level procedure observed in machine vision and motion planning for **dynamic re-planning** averaged across multiple runs of the experiment.

The machine vision subsystem consists of image acquisition, image processing and communication. The reported CPU time for image acquisition is a *worst-case*



(a) (b)

Figure 3.15: Still video frames captured during a pick and place task involving a dynamic obstacle, (a) image of experimental setup captured from an external camera (not integrated into the physical system), (b) images from the two optical cameras superimposed with the obstacle coordinates computed by machine vision (as seen by the user on the host PC).



(a) (b)

Figure 3.16: Average CPU times obtained in the pick and place task performed on hardware for: (a) machine vision, comprising of image acquisition, image processing and data communication, (b) motion planning, considering only the dynamic re-planning queries and comprising of reading the packages sent by the machine vision, checking collision along existing path, and motion re-planning.

value determined by the maximum time required to obtain an image from the cameras, which is limited by the frame rate of the hardware. While the cameras used in this system had a rated frame rate of 30 fps, during physical tests it was found that up to 45 ms could pass between two image frames. *Communication*

corresponds to the time required to send the spatial information of the obstacle, packaged as 8-bit units [155], to the motion planner (this does not include the time required to receive the package, which is instead taken into consideration in the motion planning side). The reported *TOTAL* CPU time corresponds to the working cycle time of the machine vision subsystem. Here the three sub-components of machine vision run in parallel threads, providing a working cycle time determined by the most time-intensive component (image acquisition). However, when considering the reaction time of the system, the individual CPU times must be summed to give the worst-case reaction time (assuming 45 ms was required to detect the latest change in the environment, subsequently followed by image processing and communication to deliver this information to the motion planner).

Conversely, each procedure within the motion planner runs in sequence. Thus the TOTAL CPU time for dynamic motion planning is computed as the sum of each individual component. Here, *check collision* corresponds to the time required to test whether the remaining segments of the existing solution is in collision with the obstacle at the currently perceived location, while *re-planning* consists of the steps considered in Table 3.3.[6]

Finally, let us consider the system reaction time taking into account each of the individual components. As mentioned earlier, the reaction time measures the delay between a change taking place in the environment and the robot beginning to execute corrective action. For us humans, the average reaction time is heavily dependant upon the nature of the stimulus. For example, according to Grice et al. [163], the average human reaction time can vary between 150 ms for haptic stimuli, 170 ms for auditory stimuli, and 250 ms for visual stimuli. While the use of machine vision more closely relates to visual stimuli, this case study takes the more general average reaction time considered in [157] as the criterion for satisfying real-time performance requirements.

In [159], the latency in robot control for a KUKA robot controlled through the ITRA toolbox's RSI-based approach is 30±3 ms. This value was obtained by recording the timestamp for sending a command to the robot and measuring a second timestamp corresponding to the first robot positional feedback that corre-

[6]During a dynamic re-planning query, the roadmap update step does not connect the goal configuration to the roadmap since re-planning will always be preceded by a regular motion planning query (assuming that the goal configuration does not change). Thus the re-planning time is generally less than the total time for a single regular motion planning query

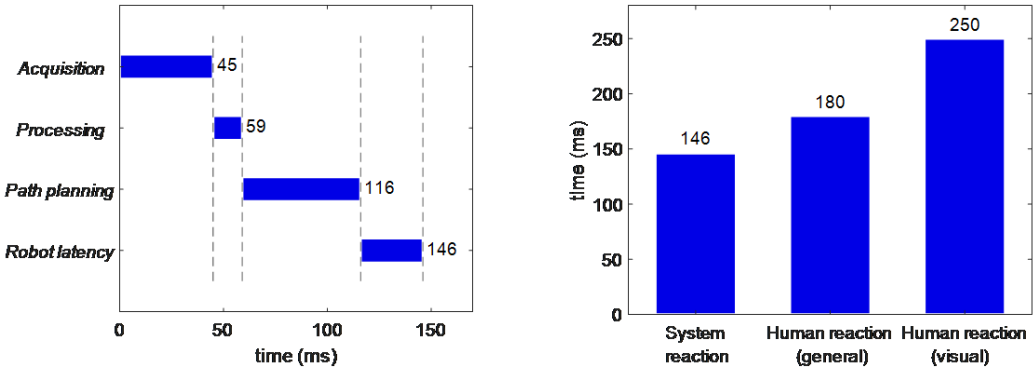


Figure 3.17: (a) Breakdown of the average *worst-case* reaction time for the integrated system accounting for image acquisition, image processing, motion planning and latency in robot control, (b) comparison of system reaction time against the average reaction time considered in [157] and the average reaction time for visual stimuli [163].

sponds to a deviation of at least 0.01 mm from the original *home* position of the end effector. This was repeated 100 times to obtain the reported latency. Combining this with the previously presented results, the average *worst-case* reaction time of the system was measured to be 146 ms, which lies within the threshold to be considered as real-time response (see Fig. 3.17).

3.7.2 Failure Cases

A number of practical difficulties were encountered when conducting experiments to assess the performance of dynamic motion planning. First of all, currently no standard benchmarking experiment has been developed by the research community to enable consistent and accurate testing of adaptive motion planning algorithms. This is due to the stochastic nature of dynamic planning problems that make this difficult - dynamic obstacles do not in general move in predictable ways. In this case study, a design decision to manually control the dynamic obstacle was made to simulate the unpredictability of dynamic obstacles in the real world. However, this heavily reduced the repeatability of experiments as the obstacle does not move in exactly the same way nor at the same time across repeated trials. On some occasions, dynamic re-planning was not induced across the entire trajectory due to the obstacles failing to obstruct the initially planned path within the short duration of the robot's motion. In other trials, dynamic

re-planning was successfully induced but the planner failed to find a new path as the obstacle had either obstructed the placing location of the object, preventing the robot from reaching that point safely, or moved too closely to the robot. The latter scenario occurs due to the enlargement of the virtual obstacle considered in the planner as a result of discretising the environment, which can indicate to the planner that collision occurs between the robot and obstacle when in reality no collisions had occurred. In either of these situations, the planner continued to search for a new path whilst the robot remained stationary, allowing the system to recover once the obstacle no longer interfered with the pick and place task.

The results reported in Section 3.7.1 contains all successful re-planning trials where the obstacle successfully interfered with the planned path without obstructing the final placing task.

3.8 Discussion

First of all, it is necessary to recognise that the responsiveness of a robotic system to adapt to its environment depends not only on the computational efficiency of the underlying planner. Any planner has a high dependency on having a high accuracy representation of the environment and the objects that the system interacts with. However, this accuracy can often come at the cost of additional processing time in the sensing subsystem prior to making the information available to the planning system. Furthermore, in this case study, a rather significant amount of time was spent receiving packages from the machine vision subsystem (12 ms or 21% of the total re-planning time) when passed through TCP/IP. This latency in communication is highly undesirable, and spending time to identify the most effective means of enabling communication between components within the system can reduce the amount of computational resources spent on completing the task.

On the other end, the capabilities and performance of the robot controller also affects the real-time performance of the system. In this case study, the controller based on the ITRA toolbox enabled trajectory generation to take place in a real-time system at the controller side. While this can reduce the CPU time required to convert a path to a trajectory, the optimality of the time parameterization suffers as the controller generates trajectories at a local level (it produces a smooth trajectory between the current robot configuration and the latest target

configuration without taking into account all future points along the remaining path). Moreover, latency in the robot control can limit the responsiveness of the system to change. In most circumstances, the duration of this latency is considered insignificant. However, for those applications seeking extremely high responsiveness (less than 100 ms), the latency can become a limiting factor, yet is often neglected.

Focusing on the aspects of planning, the results of the experiments conducted have shown that based on the MATLAB-based implementation used in this case study, a substantial amount of computation time was spent on collision checking. A common drawback of all sampling-based approaches is the large number of collision checks that must be performed to evaluate the validity of sampled configurations (including the edges that connect between them). One way of reducing this cost is to improve the collision-checking algorithm itself, which has indeed been the focus of much research [40, 164–166]. The flexible collision library, an open-source collision detection and proximity query project, deserves noteworthy mention as it provides a particularly fast, ready-to-implement tool for collision checking queries.

However, even with these tools available for fast collision checking, meeting the demands of real-time motion planning generally requires some means of retaining existing knowledge about the environment. Methods that involve a pre-processing phase benefit from having this knowledge available from the beginning, while methods that do not involve an offline graph generation procedure generally requires to incrementally build up this knowledge (this is equally true from the perspective of finding optimal solutions). Algorithms belonging to the latter that are able to find a feasible solution quickly and incrementally improve a solution over time are considered *anytime* algorithms, and may be particularly beneficial to task planners that seek to find a feasible solution quickly and continue to improve the solutions of future actions during execution without incurring significant computation costs in an offline pre-processing stage.

While solving the above challenge can enable a planner to find a solution more quickly (possibly in real-time for the case of motion planning), this does not necessarily solve the more fundamental problem of *adapting* an existing solution to changing conditions during execution. Since the primary drawback of planning in C-space is the difficulty to represent obstacles in the same space, seeking to efficiently update the validity of sampled configurations in relation to changes in

the obstacle regions of the search space (i.e. ascertain that they are still collision-free) is not a trivial task.[7]

The DRM method implemented in this case study is indeed a valid approach to addressing the challenges described above. However, its use comes with a number of caveats. While extended pre-processing offline can improve the generality of the planner during online execution, it comes at a high computational cost. As mentioned earlier in this chapter, if an appropriate granularity is chosen for the discretization of the workspace, accompanied by sufficient sampling for roadmap generation, the pre-processing phase would only be required to be performed once for a given robot. The resulting mapping would be sufficient for many applications. However, to achieve this generality would require very long pre-processing times, which may not always be available. The computational resources consumed is not only limited to the aspect of computation time, but the memory required to store the subsequent map can also be a limiting factor. Lastly, the capability of the planner to find high quality solutions is largely dependant on how well the offline pre-processing phase explores the search space. Poor granularity or insufficient sampling would lead to sub-optimal solutions when run online. This is in contrast to incremental methods described earlier than continually improves a solution over time, which may be capable of converging to the optimal solution as planning time tends towards infinity.

3.9 Summary

This case study serves to provide an insight into the additional challenges faced when addressing planning problems subject to dynamic changes during robot operation. In the work presented in this chapter, I have demonstrated one approach to enabling adaptive robot behaviour for low-level motion planning in dynamic environments through a number of considerations made in the areas of perception, planning and control. In the rest of this thesis, I narrow down on some of the challenges introduced so far from the context of planning and present a number of techniques that have been developed to extend the capabilities of adaptive planning towards the level of task planning.

[7]Some exceptions to this exist. E.g. for an omni-directional mobile robot on a 2D plane, an identical representation could be used for the Cartesian space and C-space. Under this assumption obstacles do not need to be transformed between the two spatial representations, making the problem simpler for adaptive planning.

Chapter 4

Multi-Goal Path Planning for Continuous Cost Spaces

In these next two chapters I focus specifically on task and motion planning for mobile wheeled robots (MWRs). Chapter 4 primarily deals with the motion planning sub-problem, where I present developments that advance the state-of-the-art in multi-goal path planning and subsequently demonstrate how these developments can be integrated with off-the-shelf task planners to obtain high-quality solutions to static planning problems efficiently. Chapter 5 builds upon the work presented in this chapter and addresses the dynamic task and motion planning problem, where the environment is partially-known or dynamic, and re-planning online is required to maintain safe yet optimal plans.

The developments reported herein stem from the observations of key considerations identified in Chapter 3 for planning in dynamic environments. While the planning domain investigated in this chapter differs from robotic pick and place and focuses on the fixed, static environment, key considerations for improving the efficiency of planning remain relevant and are discussed throughout this chapter.

4.1 Introduction

Task planning plays a fundamental role in autonomous mobile robotic systems across many important applications. For example, in plant inspection and surveillance applications, the concept of multi-goal path planning **(MTP)** is crucial for formulating an optimal plan to visit a defined number of goal regions using the

most efficient route available. Here the definition of *most efficient* depends upon the optimisation criteria. Commonly used types of cost functions include:

- path length, the default cost function considered in most literature relating to path planning
- total curvature, which correlates to the duration taken to execute a path (just like cars, robot may need to slow down to make large curvature turns)
- energy consumption or effort, determined by the amount of work required to act against gravity to traverse up sloped surfaces
- obstacle clearance, which considers how safe a path is by evaluating the proximity of obstacles along the robot path.

Generating optimal solutions to these kinds of task planning problems cannot therefore be obtained by estimating the distance between goals using simple metrics such as the Euclidean or Manhattan distance function. Rather, explicit considerations for the geometric and spatial nature of the task is necessary to determine a good estimate of the true costs between goals.

As another example, consider the application of planetary exploration, where a robot is assigned a long-term mission involving multiple different objectives such as sampling soil, capturing images of key landmarks or monitoring environment conditions within specified regions. Achieving a single objective of this nature require the execution of multiple primary actions in a particular order. For example, taking an image requires navigating to the landmark, adjusting a pan-tilt unit carrying the camera, and then saving an image. The dependency of some actions on other actions being achieved first is described as task precedence. Furthermore, long missions often require considerations for replenishing the battery life of the system. The majority of mobile robots achieve this through docking to a charging station. Task planners will therefore need to account for these secondary actions that do not directly achieve any single objective, but nevertheless is essential for accomplishing the mission.

In these kinds of circumstances, application-specific cost functions may additionally be required. Returning to the example of planetary exploration, existing Mars rovers have relied upon solar energy captured by on-board solar panels to maintain battery levels for extended deployment. Here the solar influx received

along a path may provide a significant contribution to the cost function for path optimization.

For applications similar in complexity to the above, the planning task is no longer a simple sequencing problem. While a poor sequence would only result in a sub-optimal plan in MTPs, for the more general task planning problem the addition of precedence constraints can mean that an ill-conditioned problem would result in an infeasible solution. This necessitates the use of symbolic task planning, but like before, *optimising* a solution requires inference from some form of geometric and spatial reasoning. This type of reasoning is conveniently addressed through motion planning.

In this chapter, I investigate the problem of motion planning involving multiple goals. Solutions to these problems provide the necessary geometric information required for optimal task planning in the MWR domain. While many motion planning algorithms exist in literature (see Section 2.1), exhaustively applying a single-query planner to compute a path between every possible combination of goals is expensive. Conversely, the use of multi-query planners carry the disadvantage of long pre-processing times and high memory requirements. As the literature review in Chapter 2 and the findings from the case study presented in Chapter 3 have shown, the concept of reusing past state space exploration information to solve new instances of motion planning can provide higher efficiency planning. In this work I expand on this concept to simultaneously solve multiple path planning queries by sharing the same exploration information between multiple problems. This chapter therefore introduces the Multi-T-RRT* algorithm, an efficient method for computing high-quality paths between all pairs of goals within an MTP-like problem. The algorithm enables fast computation without the requirement for pre-processing while possessing probabilistic completeness and asymptotic optimality guarantees through the adoption of sampling-based algorithms[1]. The algorithm also supports general cost functions through explicit considerations for continuous cost spaces.

Returning to the problem of task planning, I go on to validate the viability of integrating the Multi-T-RRT* algorithm with PDDL task planning (the de facto standard of symbolic task planning, as discussed in Section 2.2.1.5) to solve

[1]I chose to adopt sampling-based algorithms due to their superior reliability and generality as identified in the literature-based evaluation of motion planning methods reported in Chapter 2.

optimal planning problems in the MWR domain. The algorithm is benchmarked against baseline planners to quantify its performance in terms of plan quality and planning efficiency relative to existing methods, showing that the algorithm: (i) scales well with the dimensionality of the problem, (ii) provides low-cost solutions that adequately accounts for general cost spaces, and (iii) reduces planning time by several fold compared to a single-query motion planning approach.

The rest of this chapter is organised as follows: in Section 4.2 I give a formal definition for the path planning problem and describe the planning domain considered in this chapter. Section 4.3 then introduces the Multi-T-RRT* algorithm and provides a description of its components. Sections 4.4 and 4.5 presents the results of a series of simulation studies and benchmarks, respectively, conducted to evaluate the behaviour and performance of the algorithm. Finally, Section 4.6 gives a discussion on the relevance of the Multi-T-RRT* algorithm in the wider context of robotic planning, while Section 4.7 summarises the chapter.

4.2 Problem Statement

4.2.1 Path Planning Formulation

The path planning problem is defined in the following way. Let C be the robot configuration space in d-dimensional $\mathbb{R}^d$, where all infeasible regions due to collision is denoted as $C_{obs} \subset C$ and all valid free space is denoted by $C_{free} \subset C \setminus C_{obs}$. A single configuration within C is denoted by q. The standard path planning problem is then given by $\{C, q_0, q_g\}$, where $q_0 \in C$ is the starting configuration and $q_g \in C$ is the goal configuration. Thus we define the feasible path planning problem as follows:

Problem (feasible path planning)**.** *Given a standard path planning problem of the form $\{C, q_0, q_g\}$, find a collision-free path $\sigma_{0,g}$ such that $\sigma_{0,g}(\tau) \in C_{free}$ for all $\tau \in [0, 1], q_0 = \sigma_{0,g}(0)$ and $q_g = \sigma_{0,g}(1)$. If a collision-free path exists, the problem is feasible.*

Now let $c : C \to \mathbb{R}_+$ be the cost function that maps all configurations within the configuration space to a real positive value within a continuous cost space. By denoting $c_p(\sigma) = f(c, \sigma)$ as the total cost of a feasible path, the optimal path planning problem can be defined.

Problem (optimal path planning)**.** *Let* Σ *be the set of collision-free paths to a path planning problem* $\{C, q_0, q_g\}$*. Given the cost function* c*, find an optimal path* σ^* *such that* $c_p(\sigma^*) = \min_{\sigma \in \Sigma} c_p(\sigma)$*.*

4.2.2 Task Planning Domain

The general task considered both in this chapter and in Chapter 5 consists of a single mobile robot free to move within an $\mathbb{R}^2$ environment. Let us define a set of landmarks L as the set of goal configurations where the robot must visit to achieve a prescribed task. A basic *MoveTo*(l_i, l_j) action for any pair of landmarks l_i, $l_j \in L \mid_{i \neq j}$ describes a movement action from goal configuration l_i to goal configuration l_j whose cost is determined by $\sigma_{i,j}$ (obtained through path planning). The robot starts from a root landmark l_0 (which I will refer to as the *robot base*) and must end at a goal landmark l_g. Task precedence constraints may additionally be specified to determine which tasks cannot be completed before certain sub-goals are met. The objective of the planning problem is to find a valid task plan and the corresponding collision-free motions to complete a set of prescribed tasks while satisfying any task precedence constraints and minimizing the total incurred cost of motion (according to the optimal path planning definition).

For a problem with no task precedence and where no actions are specific to any landmarks (e.g. where the same action is performed at every goal or the robot continuously performs a secondary action during motion), the planning task reverts to an MTP. In this scenario the solution to the planning problem is an optimal visiting sequence for L and the corresponding shortest motions for each movement action. An example of a general planning problem is illustrated in Fig. 4.1.

This chapter primarily focuses on the path planning component of the task planning problem (though some preliminaries are provided in relation to the task planner). Chapter 5 gives further attention to the integrated planning architecture while addressing the aspects of adaptive task and path planning for the task domain described here.

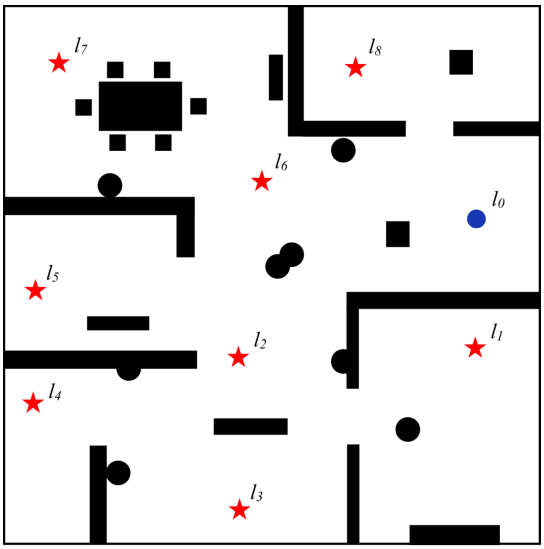


Figure 4.1: A conceptual illustration of the task planning domain, whereby the robot must visit a set of landmarks (indicated by red star markers) starting from a robot base landmark (shown as a blue marker). Collision with obstacle regions shown in black should be avoided.

4.3 Multi-T-RRT* Algorithm

4.3.1 Overview

Unlike the standard path planning problem where we would seek to find a path between a single pair of start and goal configurations, I address a multi-goal path planning problem where we seek to find all the paths between every possible permutation pair of start and goal configurations. These start and configurations are made up of all possible combinations between the set of L landmarks in the task planning domain.

The path planning algorithm presented in this section, henceforth referred to as Multi-T-RRT*, adopts a similar approach to the multi-tree extension of the T-RRT algorithm [53], but further extends it by enabling optimal paths (as opposed to purely feasible paths) to be found through the use of RRT* algorithm's rewiring procedures [43]. The efficiency of the algorithm is further improved by the use of sorting procedures inspired by the Bi-RRT* algorithm in [51].

General cost spaces are accounted for through the use of a transition test first proposed in [48]. Given this continuous cost space from which a cost value can be derived for all robot configurations, we define the cost function for total path cost as a discrete, weighted sum of **integral cost** and **path length**:

$$c_p(\sigma) = length(\sigma)\left(\frac{w_a}{\omega}\sum_{k=1}^{\omega} c\left(\sigma\left(\frac{k}{\omega}\right)\right) + w_b\right) \tag{4.1}$$

where ω is the number of subdivisions of σ and w_a and w_b are weights such that $w_a + w_b = 1$, $w_a \gg w_b$. Here the left term in Eq. 4.1 corresponds to the discrete approximation of the integral cost of path σ derived from the continuous cost space c, while the right term corresponds to the length of path σ. According to this equation, the objective is to **minimize** the integral cost with a bias towards shorter paths. Note that the heuristic nature of this cost function means that it is a dimensionless quantity.

The path planner terminates when a termination criteria is met. This could be defined based on a bounded computation time, maximum number of iterations, or the existence of a feasible solution for all possible movement actions. The planner then returns the best set of paths and corresponding costs found for each of these possible movement actions, if one is found.

Once the path planner terminates, the obtained path costs along with the original task planning objectives are used to generate a PDDL problem file for a predefined PDDL domain. Any movement actions for which a feasible path had not been obtained (as a result of the termination criteria used) is assumed to carry an infinite cost (the action is infeasible)[2]. The PDDL problem can then be solved using a compatible planner. In this work I apply openly-available planners, namely the successful Metric-FF and LPG-*td*, which are able to handle numerical expressions for the optimisation of plan costs relative to the cost function used to compute individual path costs. Example PDDL domain and problems files for a small MWR task planning problem, configured for compatibility with the LPG-*td* planner, can be found in Appendix A.

In the remainder of this section I devote the attention to the Multi-T-RRT* algorithm. I refer readers to Chapter 5 for further details on the high-level task and path planning architecture.

[2]Note that it is possible to solve the task planning problem without finding a feasible solution for all movement actions. In fact, for a given planning domain consisting of a set of landmarks L, a solution for any domain-specific problem can be solved if all landmarks form a fully-connected graph such that any landmark $l_i \in L$ can be reached from every other landmark $l_j \in L \mid i \neq j$ by traversing the graph.

4.3.2 Algorithm Description

The Multi-T-RRT* is summarised as pseudo-code in Algorithm 1. The algorithm begins by creating m trees, one rooted at each landmark in L and only containing the root as a node in the tree. At each iteration, a tree is selected for expansion in a round-robin fashion. Then, like the T-RRT* algorithm [49], a sample q_{rand} is drawn from the configuration space and a nearest node q_{near} from the tree is found. A configuration q_{new} is generated by using a steering function to guide q_{near} towards q_{rand}. The transition test is performed on q_{new} and poor quality configurations are rejected according to its cost value $c(q_{new})$. If accepted, the algorithm searches for the set of neighbours Q_{near} that lie within an adaptive radius r [43] from q_{new}.

A neighbour sorting procedure, adopted from [51], is applied to sort a list of path cost, configuration and path triplets $u = \{c_p(\sigma), q, \sigma\}$ obtained from the neighbour nodes in ascending order according to the quantity c_p. This step reduces the number of computations required for finding the best parent node by always assessing the best candidate node first. Subsequently, the algorithm iterates through each neighbour node until a collision-free connection with q_{new} can be achieved. The corresponding neighbour node is then assigned as its parent node.

The list of neighbour nodes are then checked for rewiring, a characteristic step of the RRT* algorithm. Nodes whose path from the root can be improved by passing through the newly added node q_{new} are rewired as a child of q_{new}. This necessary step ensures that paths obtained by the algorithm is asymptotically optimal. The algorithm then searches for a feasible connection from q_{new} to every other tree by finding a collision-free path from q_{new} to neighbouring nodes of foreign trees. Where such a path exists, a complete path between the roots of the two corresponding trees is found. The global set of path solutions is updated if this local solution is of higher quality than any existing solution for the corresponding pair of landmarks. Finally, a shortcutting procedure is applied at the end of the tree expansion phase to further improve the quality of final solutions.

Each of the sub-functions in Algorithm 1 are described in further detail below:

initTree - initialises a tree structure consisting of nodes, V, and edges, E, such that $T = \{V, E\}$. Initially the nodes list contains only the root of the tree and an empty set of edges.

sampleConfiguration - Goal biasing is adopted for sample generation. This

Algorithm 1 Multi-Tree T-RRT*

Input: The configuration space C, the cost function $c \rightarrow R_+$ and landmarks $l_k \in L \mid k = 0...m$

Output: Trees $T_k, k = 0...m$ and path solutions Σ_{best}

1: **for** $k = 1$ **to** m **do**
2: $T_k \leftarrow initTree(l_k)$
3: **end for**
4: $Cost_{best} \leftarrow \infty; \Sigma_{best} \leftarrow \emptyset$
5: **while not** $stoppingCriteria(T_k, k = 0...m)$ **do**
6: $T' \leftarrow nextTreeToExpand()$
7: $q_{rand} \leftarrow sampleConfiguration(C)$
8: $q'_{nearest} \leftarrow findNearestNeighbour(T', q_{rand})$
9: $q_{new} \leftarrow steer(q'_{nearest}, q_{rand})$
10: **if** $transitionTest(T', q_{nearest}, q_{new})$ **then**
11: $Q'_{near} \leftarrow findNearNeighbours(T', q_{new})$
12: $U_{near} \leftarrow sortNeighbours(Q'_{near})$
13: $q'_{parent} \leftarrow getParent(U_{near}, q_{new})$
14: $addNodeAndEdge(T', q'_{parent}, q_{new})$
15: **for all** $(c_p(q_{new}), q'_{near}, \sigma_{near}) \in U_{near}$ **do**
16: **if** $c_p(q_{new}) + c_p(\sigma_{near}) < c_p(q'_{near})$ **then**
17: $rewire(T', q'_{near}, q_{new})$
18: **end if**
19: **end for**
20: **for all** $T_k \neq T'$ **do**
21: $(\sigma_{sol}, c_p(\sigma_{sol})) \leftarrow connectTrees(T', T_k, q_{new})$
22: **if** $c_p(\sigma_{sol}) \neq \emptyset$ **then**
23: $Cost_{best|T',T_k} \leftarrow c_p(\sigma_{sol}); \Sigma_{best|T',T_k} \leftarrow \sigma_{sol}$
24: **end if**
25: **end for**
26: **end if**
27: **end while**
28: $\Sigma_{best} \leftarrow shortcutting(\Sigma_{best})$

function returns a random configuration sampled from C, with a small bias towards returning one of the other tree roots, each with equal probability. In this way the algorithm biases the growth of a tree towards all other landmarks.

findNearestNeighbour - Returns the nearest node to a configuration q from a tree T_k in configuration space C.

steer - Given two configurations q_A, $q_B \in C_{free}$, the steer function produces configuration $q_C \in C_{free}$ which is a result of advancing configuration q_A towards

q_B with a maximum step distance of η_{step}. The value of η_{step} is normally set according to the maximum dimensions of the environment and the degree of clutteredness (smaller step sizes enable the tree to explore narrow passages). By adjusting this quantity according to the size of the environment, the algorithm scales efficiently across different units. Thus in the evaluations presented in this chapter I consider environments in arbitrary units as the algorithm's performance is unaffected by this.

transitionTest - Let us define c_{max} and c_{min} as the maximum and minimum cost space values of the nodes in the tree T_k, respectively. With a tree node q_i and a sampled node q_j as input, the transition test filters configurations according to the following rules. If $c(q_j)$ exceeds the maximum threshold c_{max}, it is automatically rejected. If on the other hand $c(q_j) < c(q_i)$, it is automatically accepted. If neither of the first two statements are true, then q_j is accepted with the probability defined by:

$$p_{ij} = \exp\left(- \frac{\Delta c_{ij}}{(K \cdot H)} \right) \tag{4.2}$$

where K is a normalizing constant determined by the average cost of query configurations, Δc_{ij} is the slope of the cost between q_i and q_j, and H is the temperature parameter that is adapted by a factor h_{rate} in each function call based on acceptance or rejection. Letting α be the growth factor defined by Eq. 4.3, the tuning of the temperature parameter H is given by:

$$H = \begin{cases} \frac{H}{2^{\alpha}}, & \text{if } q_{new} \text{ is accepted} \\ H \cdot 2^{h_{rate}}, & \text{otherwise} \end{cases}$$

$$\alpha = \frac{c_j - c_i}{c_{max} - c_{min}} \tag{4.3}$$

Thus the parameter h_{rate} directly controls the rate of exploration across high-cost regions.

findNearNeighbours - The nearest neighbours to a node q within the same tree is returned based on the concept of a shrinking ball radius centred on q, which was introduced in the RRT* algorithm [43]. The equation defining this radius is given by:

$$r_n = \min\left\{\left(\frac{\gamma}{\zeta_d}\frac{\log n}{n}\right)^{1/d}, \eta\right\} \tag{4.4}$$

where d is the dimension of the problem, ζ_d is the volume of a unit ball in $\mathbb{R}^d$, γ is a constant greater than a certain lower bound γ_L, and η is an upper bound constant to limit the length of r_n.

sortNeighbours - Arranges neighbour nodes Q_{near} into a list of triplets consisting of $u = \{c_p, q, \sigma\}$ and sorted in ascending order according to path cost, given by equation 4.1.

getParent - Given a set of sorted neighbour triplets U and the sampled configuration q_{new}, *getParent* iterates through each triplet $u \in U$ and performs a collision check along a connecting edge between $q_{neighbour}$ and q_{new}, terminating when either a feasible edge is found or when all neighbours are evaluated. The purpose of sorting the set of triplets u is to minimise the number of collision checks required, which speeds up computation, while guaranteeing that the first feasible parent node found always provides the lowest path cost to q_{new}.

addNodeAndEdge - This function expands a tree T_k by adding the new node q_{new} to the tree with a connecting edge from its parent node q_{parent}.

rewire - The rewire procedure is a crucial component of the RRT* algorithm, providing the algorithm with asymptotic optimality guarantees. After a new configuration q_{new} is added to a tree T_k, the rewire procedure attempts to reduce the cost to reach any of the neighbour nodes q_i by reassigning q_{new} as the parent node. Acceptance for rewiring is subject to the satisfaction of two conditions: (i) the cost of the new path σ_{new} to q_i through q_{new} is less than the cost of the current path σ_{prev} to q_i, and (ii) the connecting edge between q_i and q_{new} is collision-free. When these two conditions are met, the original edge connecting q_i to its parent is updated to reflect the accepted rewiring. As the number of expansions in the algorithm tend towards infinite, the shortest found path from the root of the tree to any point in the free C-space corresponds to the optimal path.

connectTrees - For $T_i, T_j \in T \mid i \neq j$, and a configuration q_i in T_i, the function finds a nearest node $q_j \in T_j$ to q_i. If $||q_j - q_i|| < \eta_{step}$, the function proceeds to attempt a connection between T_i and T_j through q_i. Lines 11-19 in Algorithm 1 are re-applied with T_j and q_i as inputs. If a parent is successfully found, $q_i' = q_i$ is added to T_j. Subsequently, the path between the corresponding roots of T_i and T_j is given by $\sigma = \big(\sigma(q_i), flip\big(\sigma(q_i')\big)\big)$ and the total path cost is

$c_p = c_p(q_i) + c_p(q_i')$.

shortcutting - Although the Multi-T-RRT* is characterised by asymptotic optimality guarantees, in practice a substantial number of iterations is required to converge towards a high-quality solution. In other words, further improvements can generally be made to paths found after a moderate number of iterations. The shortcutting procedure refines the quality of a path σ by attempting to remove redundant nodes along the path through a local search procedure. Unlike other variants of RRT algorithms, where shortcutting only takes into consideration the path length and collision avoidance criteria, the implementation here adopts the segment cost, computed using equation 4.1, into consideration to account for general cost functions. The shortcutting procedure iterates through each triplet of consecutive points $\{q_A, q_B, q_C\}$ and removes q_B if $c_p([q_A, q_B]) + c_p([q_B, q_C]) > c_p([q_A, q_C])$. Evaluations across each consecutive triplets of points are recursively performed until no further local shortcuts can be made to σ.

4.4 Simulation Study

A series of simulations was conducted to validate the performance of the path planning algorithm. These simulations were conducted on an Intel Core i5 3320M 2.6 GHz processor with 8 GB RAM, and in all cases the results from the path planner were used to formulate a PDDL problem, which was subsequently solved using the LPG-*td* planner.[3]

Simulation 1 considers an MWR within a cluttered environment, where the cost function c is given by the inverse of the shortest distance to the nearest obstacle. Minimising this cost function translates to increasing the obstacle clearance along the planned path. Using a set of 5 landmarks (inclusive of the robot base) distributed in a 100×100 environment, the Multi-T-RRT* algorithm was applied to the task with the following parameters: the algorithm was set to terminate after 5000 successful iterations (i.e. when a total of 5000 nodes were added across all search trees), $w_a = 0.99$, $w_b = 0.01$, $\eta_{step} = 1$, $h_{rate} = 0.01$ and $H = 10^{-6}$.

Fig. 4.2 shows the complete solution set obtained using the Multi-T-RRT* algorithm for Simulation 1. In Fig. 4.2a, a visualisation of the five trees at the end of the search is provided. These trees originally consisted only of the corre-

[3]Detailed benchmarking to assess the performance of two different planners, namely LPG-*td* and Metric-FF, is provided later in Section 4.5.2.

sponding landmark nodes (represented by colored markers), and were iteratively grown to explore the search space. Although no individual tree by itself provided effective coverage of the entire map, the algorithm is able to obtain a continuous path from any one landmark to all other landmarks across the environment by joining together multiple trees. This means that any one tree contributes to the solution of several paths by making information obtained by exploration available to all of the planning instances being considered simultaneously. This somewhat resembles a multi-query planning algorithm by being able to support the search of multiple solutions, but differs in that it does not require a time-consuming pre-processing stage and each **new** planning query requires exploring the search space from scratch.

Figs. 4.2b-4.2f show the entire set of continuous paths that are generated by finding the shortest connections between the trees in Fig. 4.2a, where each plot is organised to show the subset of paths that start or end at the same landmark. Observe that in all the solutions obtained, whenever a path passes through narrow regions where obstacles lie on either side, the path lies close to the medial axis of the environment, where obstacle clearance is at its maximum. Finally, Fig. 4.2g shows the final route obtained by applying the LPG-*td* planner to solve the corresponding PDDL problem.

In **Simulation 2** I consider the application of the algorithm to an outdoor navigation task involving the traversal of rough surfaces. Here I define the cost function as the elevation of the robot, such that paths that require ascending steep slopes incur particularly high costs. I perform two planning queries for solving this problem. In the first query, the algorithm is set to terminate once a collision-free path for all pairs of landmarks is obtained (i.e. an initial solution is found for all path planning sub-problems). In the second run, I set the algorithm to run for exactly 3000 iterations. This gives the planner additional time to explore the search space. In both trials $w_a = 0.01$, $w_b = 0.99$, $\eta_{step} = 1$, $h_{rate} = 0.008$ and $H = 10^{-6}$. The resulting solutions are shown in Fig. 4.3.

In this simple problem, the optimal visiting order for the 5 landmarks is quite intuitive. We can see that by providing further planning time, the algorithm is able to find a better quality solution to the task planning problem. This optimisation takes place not only in the low-level paths, but at the high-level ordering task, which is informed by the path costs derived from the continuous cost space.

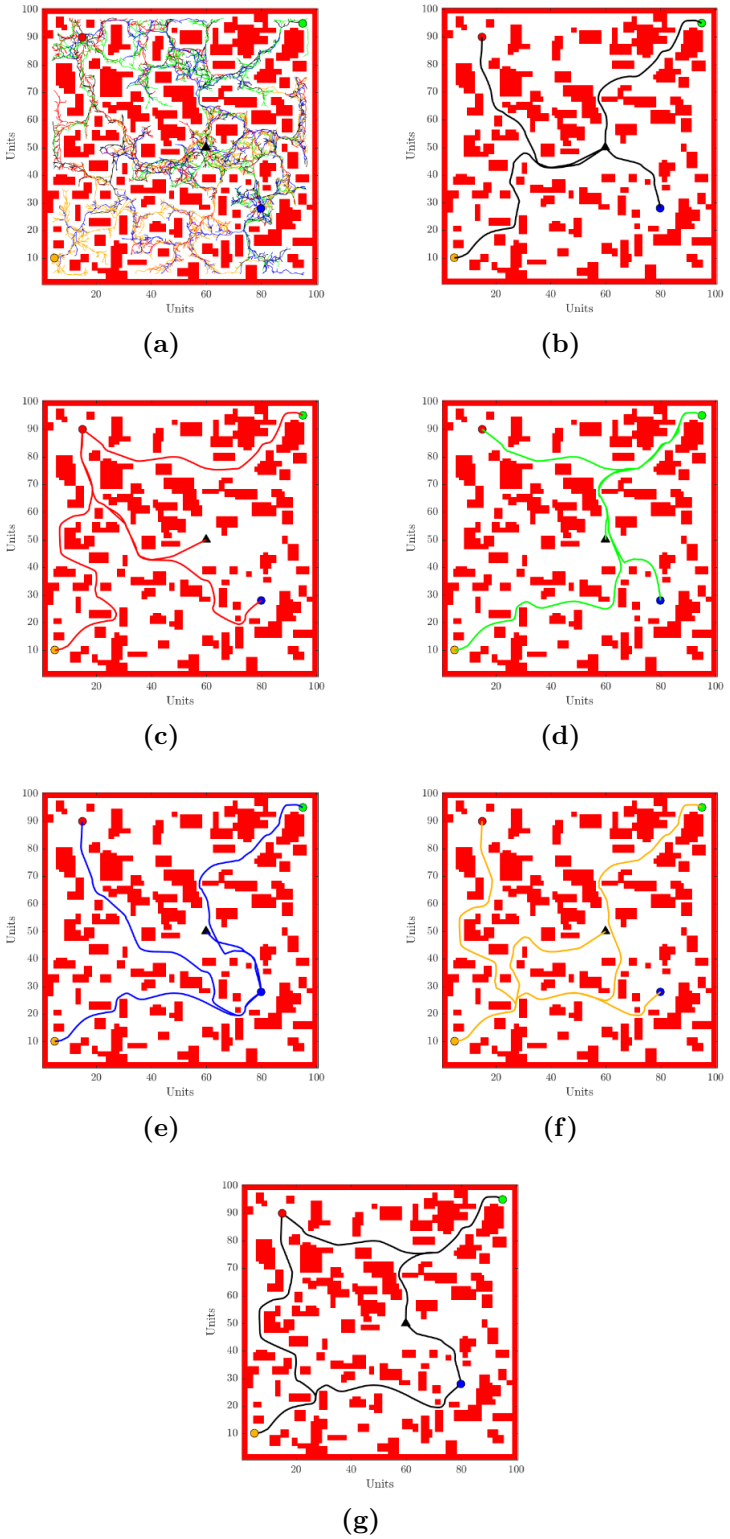
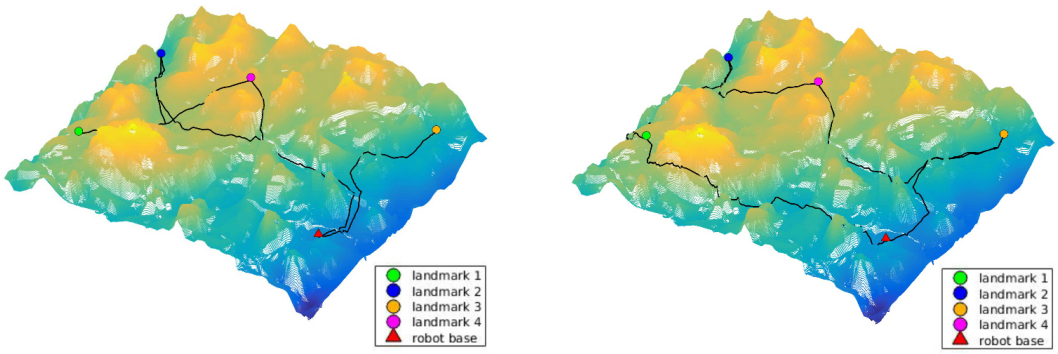


Figure 4.2: Sample sets of solutions to a Multi-T-RRT* planning query. Triangle marker represents the robot base, while circular markers represent a regular landmark.



(a) **(b)**

Figure 4.3: Incremental improvement to quality of solution over increasing iterations in the mountain environment, (a) initial solution obtained after 1188 iterations of the path planning algorithm, (b) solution after 3000 iterations of the path planning algorithm.

These simulation studies have demonstrated the capability of the planner to meet different planning needs based on the priority between plan quality and planning speed. In simulation 1, extended planning time enabled the path planner to obtain high quality paths by further exploring the search space after an initial solution was obtained, while in simulation 2 I showed that a feasible solution (but not necessarily a *near*-optimal solution) was retrieved with minimal iterations. These behaviours are inherently controlled by the termination criteria that dictate the required planning time of the algorithm.

4.5 Benchmarking

In the previous section I have shown that the integration of the Multi-T-RRT* algorithm with PDDL planning is a viable approach to solving task planning problems for mobile robots. In this section I provide a number of evaluations conducted to benchmark the performance of the integrated planner.

4.5.1 Benchmarking Path Planners

To evaluate the quantitative advantage of applying the Multi-T-RRT* to compute paths between all possible pairs of landmarks, I compare the performance of the algorithm against a standard approach of performing multiple instances of the T-RRT* algorithm [49] to individually compute each landmark-to-landmark path.

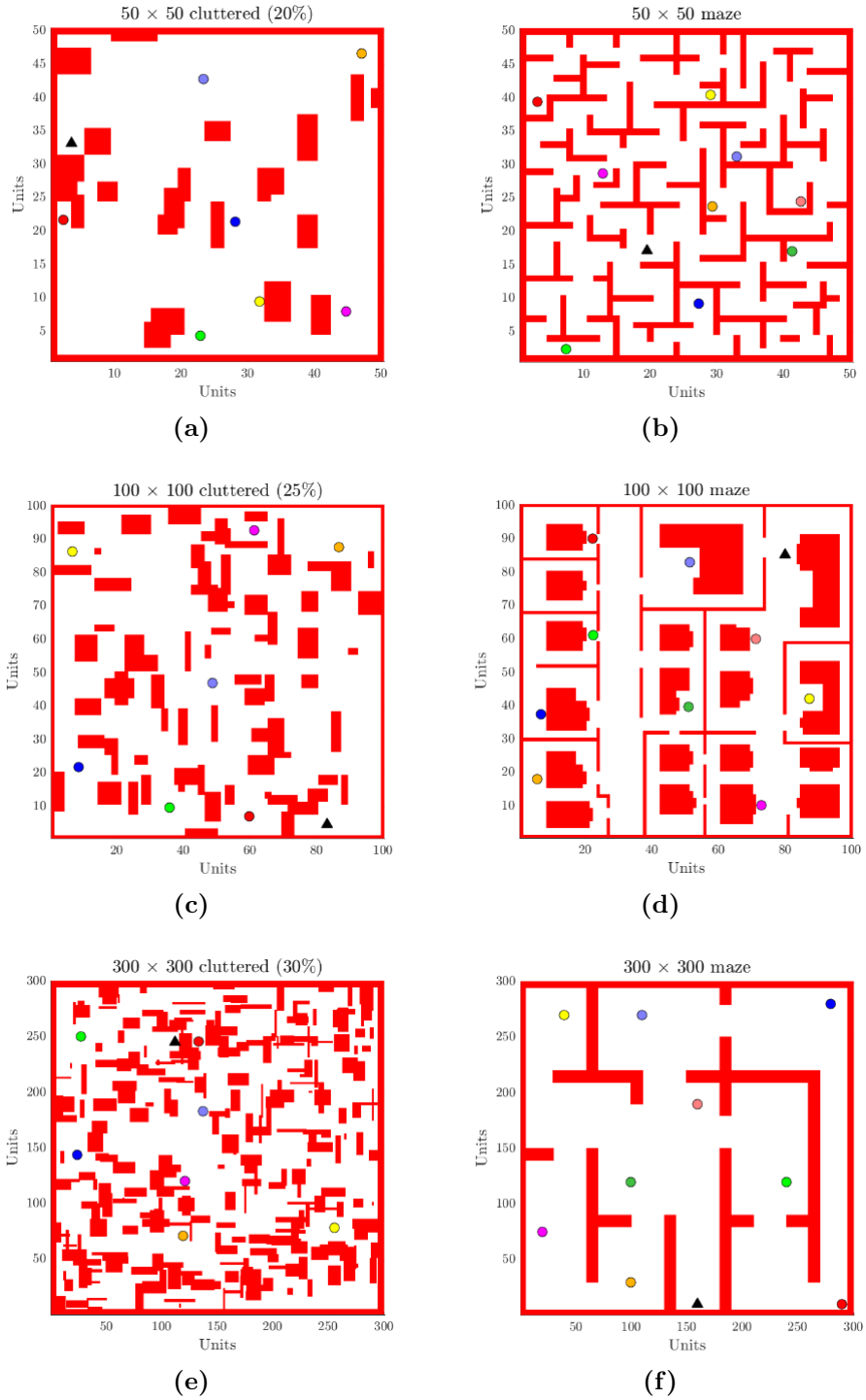


Figure 4.4: Example environments of various sizes for benchmarking Multi-T-RRT*. (a), (c) and (e) show cluttered environments consisting of randomly generated obstacles as a percentage of the C-space, while (b), (d) and (f) show examples of structured maze-like environments. Triangle markers indicate the robot base position and circular markers indicate landmarks.

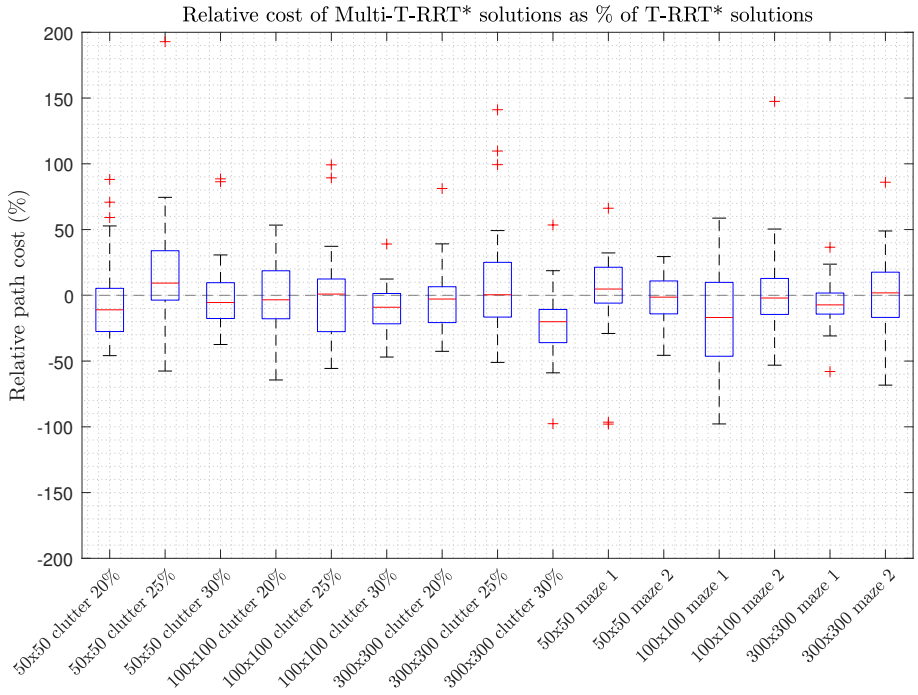
Fifteen different 2D environments were generated to conduct these trials, with the cost function defined as maximising obstacle clearance as considered in Simulation 1. These environments can be broken down into two groups: a cluttered environment comprising of randomly generated obstacles and a maze-like environment. Both groups contain a number of 50 × 50, 100 × 100 and 300 × 300 environments to enable the evaluation of each method's scalability. Within the cluttered environment group, three degrees of clutteredness were defined for each environment size: 20%, 25% and 30%, where the percentages describe the percentage of the C-space occupied by obstacles. This gives a total of 9 environments in the first group. As for the second group, two different structured maze-like environments were developed for each environment size (giving a total of 6 environments) to evaluate the effectiveness of the algorithms for exploring the search space. A subset of these environments are provided in Fig. 4.4 as examples along with the corresponding landmarks and robot base.

Fig. 4.5a reports the relative path costs of solutions obtained using the Multi-T-RRT* algorithm and the T-RRT* approach for each environment, given as a percentage cost difference computed using Eq. 4.5.

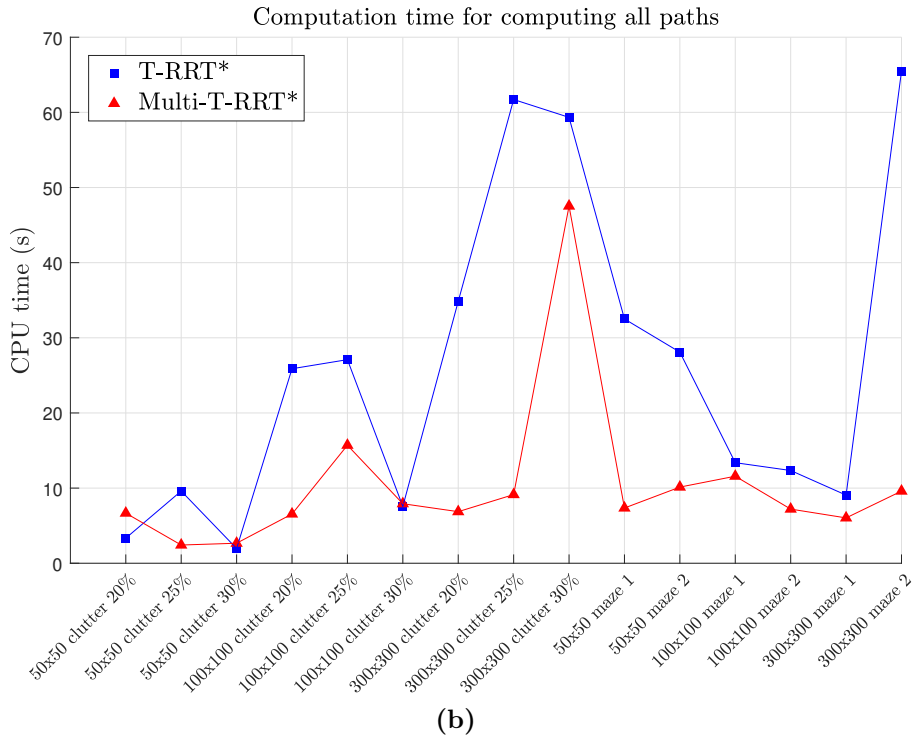
$$\Delta c_p = \frac{c_p(\sigma_{M-T-RRT^*}) - c_p(\sigma_{T-RRT^*})}{c_p(\sigma_{T-RRT^*})} \tag{4.5}$$

where Δc_p is the path cost difference, $c_p(\sigma_{M-T-RRT*})$ is the cost of the path obtained by Multi-T-RRT* and $c_p(\sigma_{T-RRT*})$ is the cost of the path obtained by T-RRT*. Each box plot captures the path cost differences between every corresponding landmark-to-landmark path in the given environment, with a negative percentage difference indicating a higher quality solution obtained by Multi-T-RRT*. As the box plots show, the mean cost difference falls within the vicinity of 0% across all environments, demonstrating that the solutions obtained by Multi-T-RRT* are indeed comparable to that of T-RRT*. Since Multi-T-RRT* is a multi-tree variant of the T-RRT*, the results show that the extensions made to Multi-T-RRT* preserve the properties of the T-RRT* with respect to the quality of solutions obtained.

Fig. 4.5b reports the total computation time required by each approach to compute a solution for all path planning problems within the same environment. Here we can observe more significant difference in performance between the two methods. In all but one case, the Multi-T-RRT* algorithm was able to solve the



(a)



(b)

Figure 4.5: Performance evaluation of Multi-T-RRT* algorithm against T-RRT* [49] for multi-goal path planning (a) Total path cost as a percentage of solutions obtained by T-RRT* (b) CPU time required to compute paths between landmark-to-landmarks pairs.

path planning problems faster than by solving each problem individually using T-RRT*. In the best case, Multi-T-RRT* was able to reduce the computation time required by approximately 85% (corresponding to the *300 × 300 maze 2* environment). The one case in which Multi-T-RRT* performs poorly in comparison corresponds to the *30 × 30 clutter 20%* environment. On this occasion the planning problem was simple (only a small percentage of obstacles existed within a small environment). Hence the overhead introduced by applying a multi-tree search approach dominated the computation time on this instance.

These results indicate that the strategy of reusing the same sampled information to simultaneously solve multiple path planning problems within the same environment proves advantageous, since a potential time-saving in computation time of up to 85% could be achieved with insignificant effect on the quality of solutions obtained when compared to a standard single-query planning approach. Additionally, it does not require the expensive pre-processing of multi-query methods and substantially reduces the memory complexity of the algorithm by preserving only the information relevant to the planning task (whereas multi-query planners must adequate explore the entire C-space to adequately cope with all possible path planning problems).

4.5.2 Benchmarking PDDL Solvers

A key component of the presented approach is the PDDL planner used to find a high quality task plan given a PDDL problem file containing the true costs provided by the Multi-T-RRT* algorithm. In the previous simulation studies described in Section 4.4, I applied the LPG-*td* to obtain the final task plans. To evaluate how the performance of the integrated approach might be influenced by the choice of the PDDL planner used, a comparison between LPG-*td* and Metric-FF, two planners that have proven particularly successful in past IPCs, was conducted using the same PDDL problem file generated after applying the Multi-T-RRT*.

Three different planning instances were considered for this benchmark, each corresponding to a cluttered environment but with the number of landmarks varied between 5, 7 and 9 (inclusive of the robot base), respectively. The planning problem required that all landmarks were visited and that the robot returned to the robot base on completion of its tasks. In each of these planning instances,

Table 4.1: Benchmarking PDDL Planners

Planner	**5 landmarks**		**7 landmarks**		**9 landmarks**	
	Plan cost	CPU time (s)	Plan cost	CPU time (s)	Plan cost	CPU time (s)
LPG-*td*	**185.8**	0.58	318.8±13.1	**0.60**	275.9±13.6	**0.60**
Metric-FF	**185.8**	**0.03**	**275.1**	4.73	**249.6**	429.62

the Multi-T-RRT* algorithm was used to compute all path costs for every combination of landmark pairs required to generate the PDDL problem file. Each planner was then called to solve the same PDDL problem for 20 runs.

The open-source implementation of the LPG-*td* used a random start to obtain the initial action graph required to solve the planning problem on each run. During a single planning instance, each time a solution to the problem was found, the planner restarted the state-space search with a new action graph in an attempt to find a better quality solution. This continued until a minimum predefined search time was reached. In this benchmark, the planner was requested to conduct a search for 0.5 seconds in each run. As a result, the LPG-*td* planner provided solutions of a stochastic nature.

Table 4.1 reports the total plan cost (given by the sum of the path costs for each movement action in the final solution), total path length (obtained as the sum of the length of each individual path that makes up the final solution) and the CPU time. A $\pm 2 \times SD$ is given in the table for the plan cost obtained by LPG-*td* as it varied across the 20 runs.

In all planning instances considered, the Metric-FF planner consistently found the lower cost solution between the two planners (for the simplest case of 5 landmarks, LPG-*td* was able to achieve a solution of the same quality). For the problem involving 5 landmarks, Metric-FF was additionally able to converge to a solution in significantly less time that the 0.5 seconds allocated to LPG-*td*. However, as the size of the problem grew to 7 landmarks and 9 landmarks, the efficiency of the Metric-FF deteriorated rapidly (requiring over 400 seconds for the latter scenario). Conversely, LPG-*td* was able to converge to a solution in under a second for all cases, with the potential to improve the solution quality further if more time was allocated for the search.

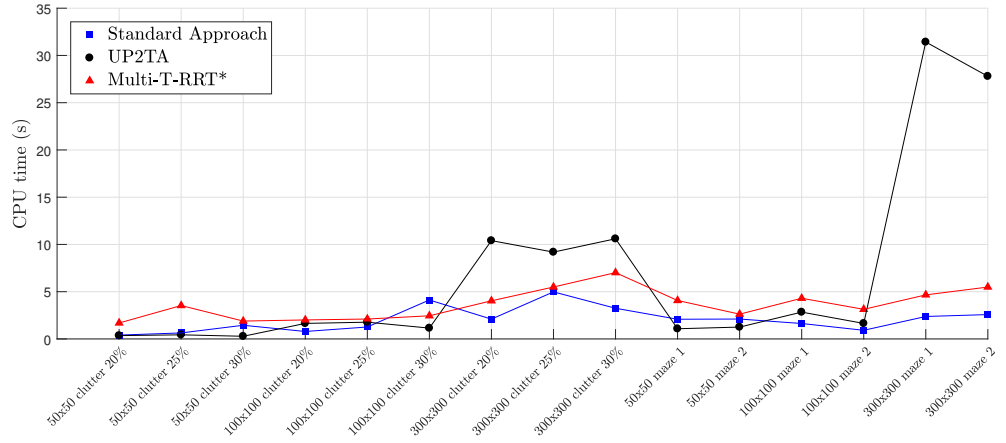
We can conclude from these results that the type of PDDL planner used drastically affects the performance of the overall system. Even when we consider only LPG-*td* and Metric-FF, we can find notable differences in the characteristics of the system. The benchmarking results have shown that LPG-*td* can reliably find a solution quickly (less than 1 second), but it does not provide any guarantee for optimal solutions. Conversely, the Metric-FF can consistently provide a high-quality solution, but potentially at the cost of reduced efficiency for large search spaces. The choice of the PDDL planner is therefore largely dependent upon the application. For example, for applications where anytime characteristics is desirable, LPG-*td* proves advantageous as it can provide an initial solution quickly for execution, but subsequently improve the later parts of a plan while early actions are performed, while Metric-FF is suited for applications requiring highly optimised plans for reasons such as safety and long-term productivity.

4.5.3 Benchmarking against UP2TA

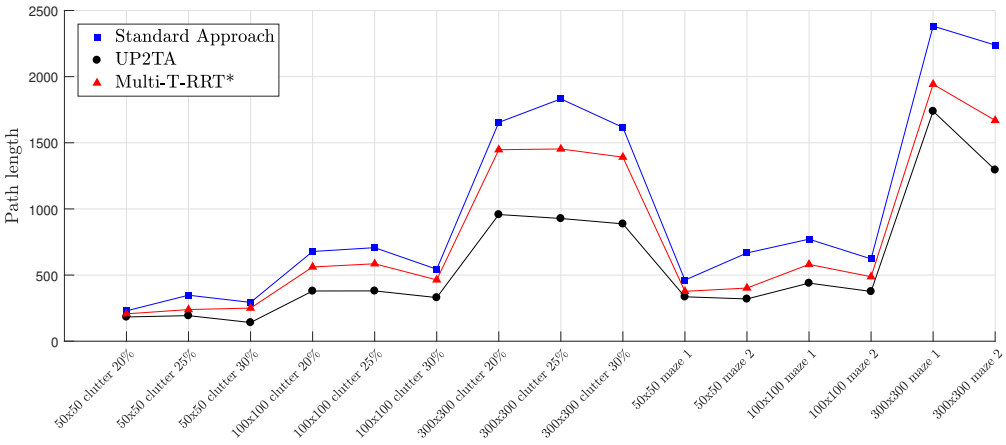
The proposed integrated planner was benchmarked against two alternative approaches to solving the task and motion planning problem for MWRs: (i) a **standard approach** that consists of estimating the cost of motion between all pairs of landmarks using the Euclidean distance metric, combining the costs with PDDL task planning to obtain an estimated optimal sequence of motions and subsequently computing the true path for each of the motions in the sequence sequentially using a bi-directional implementation of the T-RRT* algorithm, (ii) the state-of-the-art **UP2TA framework** presented in [113], which consists of employing a greedy search algorithm to approximate the cost metrics for each possible movement action, solve the corresponding task planning problem, and finally apply the Theta* algorithm to individually obtain the true paths for each required motion in the planned sequence. For consistency, the LPG-*td* planner was used for all three planning approaches to solve the task planning problem.

The set of environments described in Section 4.5.1 (see Fig. 4.4) were again used in this comparison, where each planning approach was used to solve the problem in each environment 50 times. The average total planning time, total path length and total path cost are shown in Fig. 4.6.

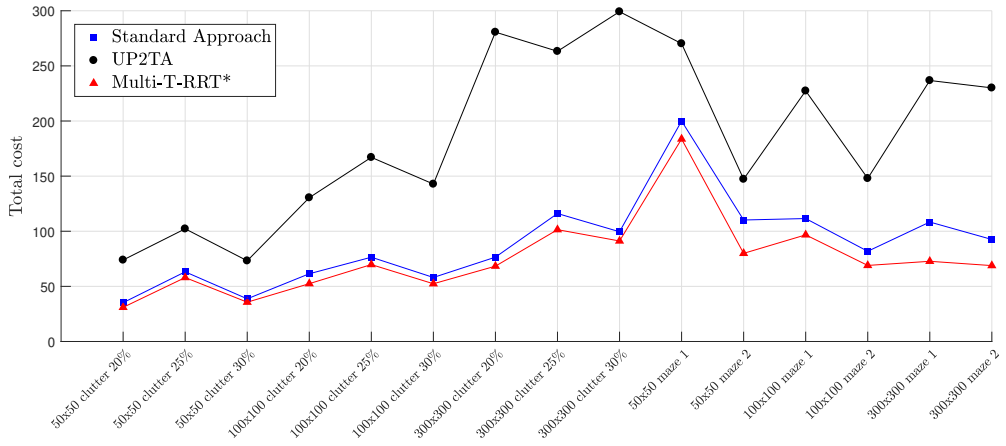
Let us first consider the performance between the standard approach and Multi-T-RRT*. Perhaps unsurprisingly, the standard approach requires less com-



(a)



(b)



(c)

Figure 4.6: Benchmarking the DA-TPP against a simple planning approach and the UP2TA method, (a) total computation time, (b) total path length of initial solution, (c) total path cost of initial solution.

putation time that the Multi-T-RRT*. This is due to the use of the simple Euclidean metric to estimate motion costs prior to planning. Thus the number of path planning problems solved in the standard approach is limited to the number of movement actions in the resulting task sequence. However, as Fig. 4.6b and Fig. 4.6c show, both the total path length and total path cost suffer, as Multi-T-RRT* outperforms the standard approach in all trials. This is a result of the poor motion cost estimations that lead to sub-optimal task sequences that are fixed by the time motion planning is performed to compute true paths. The largest difference in solution quality was obtained for *300 × 300 maze 1*, where Multi-T-RRT* achieved a 32.9% reduction in total path cost compared to the standard approach.

As for UP2TA, we can observe from Fig. 4.6a that the approach scales poorly with the the size of the environment. While UP2TA had the shortest computation times for all 50 × 50 environments, it also required a substantially longer computation time for all 300 × 300 environments (over 30 seconds for *300 times 300 maze 1*). In contrast, the Multi-T-RRT* algorithm consistently required less than 10 seconds for all trials, with the longest computation recorded at 7.6 seconds for *300 times 300 cluster 30%*. This finding agrees with the observations made in Chapter 2 that sampling-based algorithms scale well with the dimensionality of the problem, while the majority of other methods do not. Furthermore, while UP2TA was able to obtain solutions with the lowest path length for all trials, it also returned solutions with the highest total path cost among the three approaches (Multi-T-RRT* returned solutions up to 75.7% lower in total path cost). This is due to the inability of Theta* (the motion planner used in UP2TA) to account for additional cost criteria beyond path length, resulting in shortest paths that lie very close to obstacles in the environment. Thus not only does Multi-T-RRT* prove superior in terms of scalability, it is also able to account for general cost criteria that are difficult to include in methods such as UP2TA.

Overall, this comparison has shown that the Multi-T-RRT* proves superior in terms of solution quality compared to other existing approaches to task planning and that it is capable of achieving computational performance that lies close to the planning efficiency of naive strategies. Furthermore, the method scales well with the size of the C-space and is able to consolidate different cost functions for optimisation across general cost spaces.

Table 4.2: Benchmarking Path Planning Algorithm

Planner	Env. type	**Initial Plan**		**5000 Iterations**	
		Mean plan cost	CPU time (s)	Mean plan cost	CPU time (s)
Multi-T-RRT	mountain	750.35	**14.71**	638.49	**43.49**
	cluttered	196.14	**7.17**	189.30	**13.38**
Multi-T-RRT*	mountain	**712.85**	14.95	**614.78**	44.14
	cluttered	**190.97**	7.96	**181.49**	15.78

4.5.4 Benchmarking against Multi-T-RRT

Finally, I compare the performance of the Multi-T-RRT* against the Multi-T-RRT, a similar multi-tree variant of the T-RRT presented in [53], through the task planning problems considered in Simulations 1 and 2 (Section 4.4). The fundamental difference between the Multi-T-RRT and my proposed algorithm is the absence of rewiring procedures that provide the asymptotical optimality guarantee present in Multi-T-RRT*. Through this comparison, I demonstrate that the sorting procedures described in Section 4.3 enable the presented algorithm to find higher quality solutions with minimal increase in computation time.

Using the environments in Simulation 1 (cluttered) and Simulation 2 (mountain) for benchmarking, 50 trials were performed using both planners to first obtain an initial solution, and subsequently an improved solution over 5000 iterations. For both methods the LPG-*td* planner was used to obtain the final task-level plan. Planning parameters were assigned the same values used in Simulations 1 and 2. Table 4.2 reports the average plan cost and CPU time.

Observe that the proposed algorithm always obtains a higher quality solution, both initially and after extended planning. Although the computation time is always lower using the Multi-T-RRT, the Multi-T-RRT* falls behind by a very small margin (generally less than 1 second). Arguably, this is an acceptable trade-off for the resulting quality gained in the solutions. The key difference in performance is the algorithm's asymptotic optimality gain, which guarantees that the algorithm will converge to optimal solutions when given sufficient planning time.

The quality of the Multi-T-RRT solution does improve over time as the ex-

tended growth of trees allow the algorithm to better explore the search space (subsequently opening up shorter paths through more direct routes). However, generally the resulting path from Multi-T-RRT contain significant oscillation-like motions due to the rough paths that arise from the lack of rewiring. Hence Multi-T-RRT* is able to obtain better quality solutions. This can be observed particularly in the initial plans obtained by the two methods, where both algorithms have only minimally explored the environment to find a feasible solution. Despite this limited information, Multi-T-RRT* makes better use of the sampled configurations by updating the tree structure to reflect the shortest path that can be obtained given the current level of exploration across the environment.

4.6 Discussion

Based on the benchmarking and simulation study reported above, I wish to highlight a number of key features of the Multi-T-RRT* algorithm introduced in this chapter.

Firstly, the presented algorithm supports the use of any general cost criteria that can be represented as a continuous cost space. This means the algorithm can be directly applied to different applications without modification, which is unlike the majority of path planning algorithms in literature that by default only search for a shortest length path. Extending the use of these algorithms to encapsulate other costs often require custom-designed modifications based on the application.

As described in Section 3.8, collision checks are generally a costly low-level function. Eliminating the frequency of these calls can provide algorithms with better planning efficiency. The Multi-T-RRT* achieves this by using sorting procedures to order neighbour nodes when searching for a parent node so that the algorithm only performs collision checks for a neighbour node when it is the current best candidate. In Simulation 2, we saw that the planner is able to incrementally improve the quality of solutions when given further planning time (e.g. by running the algorithm for further iterations after an initial solution is obtained). The results presented in Table 4.2 show how the algorithm is able to find an initial solution to the cluttered planning problem in approximately 7 seconds while a notably better solution was obtained after 15 seconds. Similarly, in the mountain environment an initial solution was found at approximately 15 seconds, and when queried after 44 seconds a significantly improved solution

was obtained. This behaviour closely resembles an anytime algorithm, which describes an algorithm that can return a valid solution when interrupted (or queried before termination), but continues to find better solutions as further computation time is permitted. Through iterative sampling and rewiring, the algorithm is guaranteed to converge to an optimal solution as the computation time tends towards infinity if a solution to the planning problem exists (i.e. the algorithm is also probabilistically-complete). From a practical perspective, this means an initial solution can be requested early to enable fast execution, while the planner continues to run in the background to improve later steps in the plan as the robot begins execution.

Another key feature of the algorithm is the way in which it maximises the gain from every sampled configuration by using it to find solutions to multiple path planning problems simultaneously. This provides the algorithm with two key advantages: (i) it does not require a costly pre-processing phase offline prior to planning, and (ii) unlike the DRM studied in Chapter 3, the Multi-T-RRT* is a purely sampling-based algorithm that does not use any form of discretisation, eliminating problems relating to granularity. The latter has important implications, as the use of discretisation generally restricts the scalability of an algorithm (hence why deterministic algorithms such as A* and Theta* that discretise the search space do not scale well with the size of the search space, as observed in Section 4.5.3). Thus the Multi-T-RRT* can be effectively applied to robots beyond the 2-dimensional search space considered in this chapter (e.g. it can be applied to 6-DoF manipulators directly).

I also wish to highlight a key observation at the level of task planning: the choice of the planner is a critical consideration that depends upon the nature of the application. As shown in the benchmarks, LPG-*td* provides no guarantee for an optimal solution but can find feasible solutions quickly, making it very suitable for fast applications. Metric-FF on the other hand requires extensive planning times even for moderately sized problems (e.g. from 9 landmarks onward in the mobile robot task planning domain), which limits its use to offline planning. However, Metric-FF excels in applications where it is important to obtain the highest quality solutions available (e.g. for maximising safety of a robot system under uncertainty).

Finally, while this chapter has shown the use of the Multi-T-RRT* algorithm in simulations only, in Chapter 5 I will go on to describe an experiment that

involves the deployment of the paths found by this algorithm on a physical Pioneer 3-DX differential drive mobile robot.

4.7 Summary

In this chapter I have presented the Multi-T-RRT* algorithm for solving multiple path planning problems simultaneously and have shown how it can be used to provide task-level symbolic planning with spatial and geometric reasoning.

The path planning algorithm provides a more efficient way to compute the true motion costs of optimal paths and supports any type of continuous cost space, which improves the generality of the planner compared to many existing path planners that only consider minimising path length by default. By integrating the resulting path costs with a PDDL planner, I have shown that task-level plans can be optimised according to the cost spaces considered at the low-level planning domain. Finally, the algorithm also inherits the probabilistic completeness and asymptotic optimality guarantees from the RRT* by preserving key sampling and rewiring routines in the algorithm.

The benefits of applying the Multi-T-RRT* for solving multiple path planning problems simultaneously has been quantified through a comparison with the standard approach of solving multiple instances of path planning separately, showing that comparable solution quality can be achieved with up to 85% reduction in computation time. An evaluation of the behaviour of the algorithm when integrated with the LPG-*td* and Metric-FF planners shows that there is indeed a trade-off between maximising planning efficiency and minimising plan cost, as first identified from literature in Chapter 2. The LPG-*td* has subsequently been identified as a suitable planner for fast computational performance. The overall performance of the resulting integrated planner has been benchmarked against the UP2TA framework, showing that the proposed approach scales better with the dimension of the problem and is capable of consolidating cost functions not accounted for in standard path planning methods. Finally, I show that through the adoption of sorting procedures, which minimise the number of collision checks performed by the algorithm, solving task planning problems with Multi-T-RRT* can enable higher quality solutions to be found with minimal increase in computation time when compared to the original Multi-T-RRT algorithm.

In Chapter 5 I extend the work presented here to show how the Multi-T-

RRT* algorithm can be used to enable adaptive planning behaviours for anytime, dynamic applications.

Chapter 5

Adaptive Task and Path Planning Framework for MWRs

5.1 Introduction

So far we have investigated the problem of static task planning for MWRs, where I have presented an efficient path planning algorithm for integration with a PDDL task planner. This integrated approach enables a fast way to compute optimal solutions to task planning problems without relying upon offline pre-processing. In the previous chapter, preliminary benchmarking results have demonstrated the effectiveness of this approach for two different application scenarios, the first being representative of a cluttered environment, while the second mimics an outdoor navigation problem.

While Chapter 4 has primarily focused on the details of the path planning layer comprising of the Multi-T-RRT* algorithm, this chapter devotes further attention to the higher-level planning architecture. I first briefly revisit the integrated task and path planner, which consists of a PDDL task planning layer embedded with plan costs derived from the Multi-T-RRT* path planner (I shall refer to this integrated architecture as the *base planner* throughout this chapter). As shown in previous analysis, this method is capable of generating high quality solutions to general MWR task planning problems. However, the performance of a system relying on this base planner is generally sub-optimal and inadequate when we take into account real-world dynamic considerations. This is due to the ever present phenomena of change and uncertainty in the real world. Solving a task

planning problem generally assumes the world is unaffected by external influences or that future changes to the state of the world are known (and can therefore be accounted for). However, uncertainty and dynamics mean that infeasibility due to such things as the non-existence of a collision-free path between landmarks or the existence of obstruction along specific paths are difficult or impossible to determine offline.

Since the term 'uncertainty' encapsulates many different types of real-world phenomena (such as noise, geometric variations, knowledge uncertainty, state uncertainty etc.), let us narrow our considerations down to the uncertainties relating to obstacles, which directly influence the feasibility of a path or configuration in motion planning and the feasibility of movement actions in task planning. It is not uncommon that at the start of a planning task, only a partial representation of the obstacles in the environment is known a priori. The location and exact geometry of some obstacles may be uncertain, while freely movable and possibly dynamically-moving objects capable of entering and exiting the considered environment could be unknown altogether. For those applications that fall under this category and where reliable performance is required, existing implementations generally require a remote operator to provide manual override of commands when unexpected events occur or are predicted. These are generally labour intensive, difficult to detect remotely (as operators must interpret information obtained remotely from sensors in real-time) and are prone to human errors. As a result, there is a growing demand for more adaptive systems in complex, modern applications beyond carefully-designed industrial cells.

Indeed, as discussed in Section 2.2.4.1, various adaptive algorithms for general task planning have been developed to cope with dynamic changes in the environment. These methods focus on finding feasible solutions to planning problems given new observations, usually through planning one action at a time while updating the state of the world in between each action. However, according to the findings derived from the literature review presented in Chapter 2, it remains an ongoing challenge to maintain optimal solutions under these conditions as fast, explicit planning is required to consider all the goals of the problem.

Let us return to the integrated base planner introduced in Chapter 4. In addition to the above considerations for adaptive planning, another factor that can limit the performance of a planner is the trade-off between solution quality and planning time. While the Multi-T-RRT* algorithm provides asymptotic

optimality guarantees at the level of motion planning, this usually means in practice that extended planning times are required to obtain high-quality solutions. Where near-optimal solutions are specifically needed but planning time is an equally important factor, it can be unclear when to terminate the path planning algorithm to begin solving the task-level problem. A more desirable behaviour is that of *anytime*-like characteristics, which describes algorithms that compute a feasible solution to a problem in minimal time and then improve the solution using any further allocated planning time until interrupted or a termination criteria is met. When a solution is requested, these algorithms return the best solution found at that moment in time. In the context of task and motion planning, anytime planning would enable fast computation of an initial solution for execution. Once the plan is deployed for execution, the planner may then continue to improve the quality of the task plan and corresponding motions for actions not yet executed. In this way a planner could incrementally develop optimal solutions for finite-horizon planning problems.

The rest of this chapter is devoted to exploring the concepts of anytime and adaptive planning to extend the capabilities of the base planner towards online planning in dynamic environments. Using the base planner and the Multi-T-RRT* path planning algorithm as the core architecture, I introduce two extensions to provide the planner with anytime-like characteristics and the capability to re-plan both at the level of task planning and at the level of motion planning. The corresponding planning architectures are collectively referred to as the Dynamic Anytime Task and Path Planning (**DA-TPP**) framework. The contributions herein are summarised as follows:

1. I present an anytime extension to the base planner such that an initial feasible solution is obtained when a solution to all path planning queries has been computed, or when an initial termination criteria is met. Once obtained, any further planning time allocated to the planner is used to improve the quality of the existing solution by continued iterations of the Multi-T-RRT* algorithm. Experimental results are provided for the evaluation of the behaviour of the anytime task and path planner.

2. I present a dynamic re-planning extension to the base planner that enables a solution to be adapted at both the task and path planning levels to maintain an optimal solution on detection of changes in the environment. Local path

correction procedures re-plan low-level paths to avoid potential collisions, while a global re-planning procedure updates the task plan to maintain a high-quality solution after updating the costs of all feasible motions. Both of these procedures have been developed to retain existing trees generated by the Multi-T-RRT* algorithm to maximise the use of previously sampled nodes for fast computation.

3. I introduce a modification to the Multi-T-RRT* algorithm to enable the pruning of nodes in trees, which limits the total number of nodes used to explore the search space. This modification maximises the computational efficiency of the dynamic re-planning routines by placing an upper bound on the memory complexity for maintaining the search trees and limiting the time complexity of any individual re-planning query.

The rest of this chapter is organised as follows. In Section 5.2 I provide formal definitions to the task planning problem, while in Section 5.3 I introduce the software architectures for the base planner, the anytime extension and the dynamic re-planning extension. In Section 5.4, I describe the path planning routines for dynamic re-planning and the modification to the Multi-T-RRT* for tree pruning. Section 5.5 reports the experimental results conducted to evaluate the various aspects of the DA-TPP framework, and finally Section 5.6 provides a summary of the chapter given in the broader context of MWR task planning.

5.2 Problem Formulation

In this section I give formal definitions to the *feasible* task planning problem and the *optimal* task planning problem. I then combine this with the notation used in the formal definitions of feasible and optimal path planning as described in Section 4.2 to formally define the integrated task and path planning problem (ITPP).

5.2.1 Task Planning

As explained in Section 2.2, different notations can be used to represent the task planning domain. In the development of the DA-TPP framework, the PDDL representation was chosen due to its wide acceptance as the de facto standard for

AI planning. Thus I adopt the notation used in this language for the implementation of task planning described in this chapter. Nevertheless, it is important to note that alternative representations can be adopted according to the approaches used for solving the task planning problem (see Chapter 2 for details).

Five components are necessary to define a standard task planning problem: the finite state space S, the initial world state $s_0 \in S$, a finite set of actions A that are applicable within the domain, the finite set of state transitions E that result from each action, and the goal state $s_g \in S$. [79] An action $a_i \in A$ can only be applied at step k if $s_k \in pre(a_i)$, where $pre(a_i)$ is the precondition set of a_i. Here all states in $pre(a_i)$ satisfy the preconditions required to perform action a_i. The new state obtained at step $k+1$ is given by $eff(a_i)$, the effect set of a_i, such that $s_{k+1} = a_i(s_k)$[1] where $s_{k+1} \in eff(a_i)$. This state transition mapping from $pre(a_i) \rightarrow eff(a_i)$ is denoted by $e_i \in E$. Thus, the standard task planning problem consists of the tuple $\{S, so, sg, A, E\}$. Subsequently the feasible task planning problem is defined as follows:

Problem (feasible task planning)**.** *Given a standard task planning problem of the form $\{S, so, sg, A, E\}$, find a valid action sequence $p = (a_{[0]}, a_{[1]}, \ldots, a_{[n]})$ such that $s_0 \in pre(a_{[0]})$ and $s_g \in eff(a_{[n]})$. If a valid action sequence exists, the problem is feasible.*

Suppose in addition to the standard task planning problem $\{S, so, sg, A, E\}$ a finite set of action costs is defined as $G_A : A \rightarrow \mathbb{R}_+$ such that every action a_i is mapped to a real cost value $g_i \in G_A$. The cost of a given action sequence p is then given by the summation $h = g_{[0]} + g_{[1]} + \ldots + g_{[n]}$, where h is the total cost of an action sequence. Using this notation, the optimal task planning problem is defined as such:

Problem (optimal task planning)**.** *Let P be the set of all feasible action sequences for a task planning problem $\{S, s_o, s_g, A, E\}$. Given the finite set of action costs G_A, find an optimal action sequence $p^* \in P$ such that $h(p^*) = \min_{p \in P} h(p)$.*

5.2.2 Task and Path Planning

The planning domain for the ITPP problem consists of both the symbolic state space S in the task planning problem and the geometric C-space C in the path

[1]Throughout this chapter I let $a_{i[k]}$ be the short-hand representation of $a_i(s_k)$.

planning problem. Movement actions defined in the symbolic state space must be associated with feasible paths in the geometric configuration space. Thus a means of linkage is required to translate task states and movement actions to robot configurations and path plans, respectively, and vice-versa. Accordingly, a standard ITPP problem consists of an extended tuple $\{S, C, \delta_o, \delta_g, A, E\}$, where $\delta_0 = \{s_0, q_0\}$ and $\delta_g = \{s_g, q_g\}$ are state-configuration pairs. Letting $\psi_i = \{a_i, \sigma_i\}$ represent an action-motion pair, the feasible ITPP problem is defined as:

Problem (feasible ITPP)**.** *Given an ITPP problem* $\{S, C, \delta_o, \delta_g, A, E\}$*, find a valid sequence of action-motion pairs* $\Psi = (\psi_{[0]}, \psi_{[1]}, \ldots, \psi_{[n]})$ *such that* $s_0 \in pre(a_{[0]})$*,* $s_g \in eff(a_{[n]})$*,* $q_0 = \sigma_{[0]}(0)$*,* $q_g = \sigma_{[n]}(1)$*,* $\sigma_{[k]}(1) = \sigma_{[k+1]}(0)$ *and* $\sigma_k(\tau) \in C_{free}$ *for all* $\tau \in [0, 1]$*. If a valid action-motion sequence exist, the problem is feasible.*

Note that for all non-movement action-motion pairs in Ψ, $\sigma_{[k]}(0) = \sigma_{[k]}(1) = q_k$. Finally, the optimal ITPP problem is defined by re-introducing the finite set of action costs G_A (see Problem 5.2.1) and the path cost function c_p (see Problem 4.2.1). Additionally, recall that an optimal path is denoted by σ^*, where $c_p(\sigma^*) = \min_{\sigma \in \Sigma} c_p(\sigma)$. Then $\forall g_i \in G_A$:

$$g_i = \begin{cases} c_p(\sigma_i^*), & \text{if } a_i \text{ is a movement action} \\ 0, & \text{otherwise} \end{cases}$$

With a slight abuse of notation, let $h(\Psi) = g_{[0]} + g_{[1]} + \ldots + g_{[n]}$ denote the total cost of the action-motion sequence Ψ. Finally, the optimal ITPP is defined as:

Problem (optimal ITPP)**.** – *Let* Ω *be the set of feasible action-motion sequences for the standard ITPP problem* $\{S, C, \delta_o, \delta_g, A, E\}$*. Given the finite set of action costs* G_A*, find a feasible action-motion sequence* Ψ^* *such that* $h(\Psi^*) = \min_{\Psi \in \Omega} h(\Psi)$*.*

5.2.3 Task and Path Planning Domain

The task and path planning problem addressed in this chapter builds upon the description given in Section 4.2. Previously the objective of the problem was to find an optimal task plan and the corresponding collision-free motions to achieve a set of prescribed tasks while satisfying any task precedence constraints and minimising the total incurred cost with respect to a motion planning cost criteria. The considerations in Chapter 4 were limited to offline planning problems where

the environment was fixed and completely known a priori. This chapter extends these considerations to partially-known and dynamic environments.

Recall from Chapter 4 that C_{obs} represents the obstacle-occupied regions of the C-space where collision between the robot and the environment occurs. In this chapter, let us consider $C_{obs}(t)$ as the *time-variant* obstacle-occupied regions of the C-space as perceived by the robot. Thus, C_{free}, the collision-free regions of the C-space, varies according to new observations made in the environment. The objective of the planning problem in this chapter is therefore to adaptively compute and **maintain** an optimal task plan and the corresponding set of collision-free motions according to perceived observations of the environment until the completion of all tasks while satisfying all previous requirements.

Since we are interested in defining the reaction time of the robot in response to newly detected obstacles, a secondary objective of this work is to maximise the planning efficiency of the base planner to adequately cope with dynamic changes in the environments online.

5.3 Software Architecture

5.3.1 Base Planner

Let us revisit the base planner introduced in Chapter 4. Unlike the majority of planners described in Section 2.2.2, a distinguishing property of the base planner is the computation of all path plans prior to solving the task-level planning problem. Even in the CTMP literature discussed in Section 2.2.1.3, motion planning queries are generally interleaved with the task planner to determine the feasibility of actions as they are considered. Indeed for general robotic task planning problems (often involving the manipulation of objects) the search space of the problem can be too large to compute all motion plans in practical times. However, this work exploits the reduced search space unique to MWRs, which provides the opportunity to enable optimal task planning using similar concepts common to CTMP.

Using the Multi-T-RRT* algorithm, the execution costs of each movement action is computed according to the function given in Eq. 4.1. An interfacing layer then takes these cost values along with the original task-level planning problem and generates a problem file that meets the convention laid out in PDDL, where

movement-type actions are assigned the cost values from motion planning. The task-level planning problem is then solved by applying a PDDL-compatible AI planner to the PDDL domain file (presumably available beforehand given the robot's capabilities and application domain) and the newly-generated problem file. Finally, a parser is used to extract the sequence of actions from the solution and associate each movement action with the corresponding motion paths previously computed in the motion planning layer. It is worth noting here that when the base planner is applied to MTP problems, the extracted sequence of actions can be translated into a visiting sequence of goal points/landmarks. In the ITPP problems considered in this chapter, let us assume that the solution returned is a **visiting sequence of landmarks** together with the set of paths planned between successive landmarks (recall the task domain definition presented in Chapter 4).

While any PDDL-compatible AI planner can be used to obtain a solution to the problem in the task planning layer, the choice of planner matters in terms of the performance of the overall base planner. Firstly, to address the problem as an optimal ITPP problem requires the use of a planner that supports **fluents** (recall from Chapter 2 that these are numerical state variables), such as the Metric-FF or LPG-*td* planners. The use of other planners that are not able to support these features would result in finding feasible solutions only. Secondly, as discussed in Section 4.6, the use of different planners on the same problem can lead to differences in planning behaviour. In this work, I choose to adopt the LPG-*td* planner for solving the task-level planning problem in the DA-TPP framework due to its adjustable planning speed while being able to support fluents. While it is true that other planners such as the Metric-FF outperforms LPG-*td* from a solution quality perspective, the DA-TPP framework is specifically concerned with the problem of enabling fast adaptiveness to changes in an environment during execution. Thus to achieve this, an effective balance between solution quality and planning efficiency is highly important. Nevertheless, alternative planners can be used within the DA-TPP framework according to the demands of the application.

A high-level illustration of the described base planner architecture is shown in Fig. 5.1.

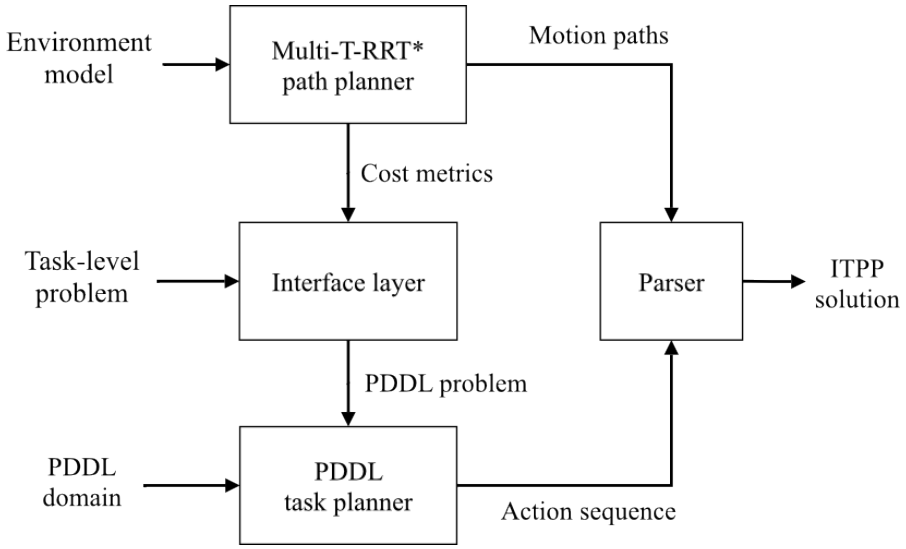


Figure 5.1: Base planner architecture

5.3.2 Anytime Planning

The anytime planning component of the DA-TPP framework is developed as an extension to the base planner to overcome the limitations of the Multi-T-RRT* algorithm's asymptotic optimality guarantee. Although algorithms with this guarantee can provide high quality solutions, they generally require extensive planning times in order to converge to these solutions. The anytime extension enables a system to first determine an initial feasible solution (where the cost is generally sub-optimal) such that the robot may begin to execute the set of tasks, and during this time the planner continues to reduce the cost of execution for pending steps in the plan.

A flow diagram of the anytime planning routine is shown in Fig. 5.2. An initial solution is first obtained using the previously described base planner with an initial termination condition that could be specified as a fixed allotted time or when connectivity between all landmarks has been achieved. This solution is saved in memory, and the planner continues to iterate through the tree expansion procedures of the path planning layer until a final termination criteria is met (e.g. when the execution of tasks has been completed). A task planning query is made to update the task plan whenever the cost of any path $c_p(\sigma_{i,j})$ between two landmarks l_i and l_j, $l_i, l_j \in L$ decreases below an upper cost bound C_s^+ defined by:

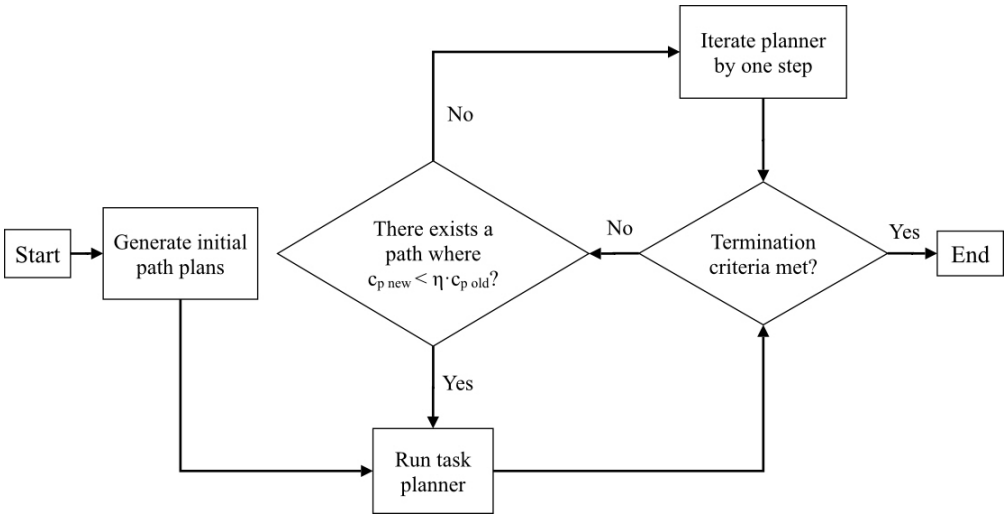


Figure 5.2: Anytime planner flow diagram

$$C_s^+ = \eta_a \cdot c_p(\sigma_{i,j})_{prev}. \tag{5.1}$$

Where η_a is the *anytime bound constant* with a value less than or equal to 1, and $c_p(\sigma_{i,j})_{prev}$ is the cost of $\sigma_{i,j}$ found at the time of the previous task planning query. That is to say, whenever a new solution to a path planning problem between two landmarks with a minimum guaranteed cost reduction is found, a new attempt to find a better task sequence is made. By defining an appropriate value for η_a, the number of task planning queries can be controlled to avoid unnecessarily performing high-level task planning in instances where the planner has likely converged to the optimal sequence of actions (i.e further improvements to the path plans would not affect the ordering of the action sequence p).

Note that as the robot achieves each task during execution, the task planning sub-routine first updates the PDDL problem file to reflect the changes to the initial state of the world (within the scope of the task planning query) prior to solving it, such that any updates to the task plan considers only those tasks that have not yet been made. Thus as the planner continues to search for a better solution, the current best plan for the remaining set of incomplete tasks is stored as a pending output to any subsequent queries made to the anytime planner.

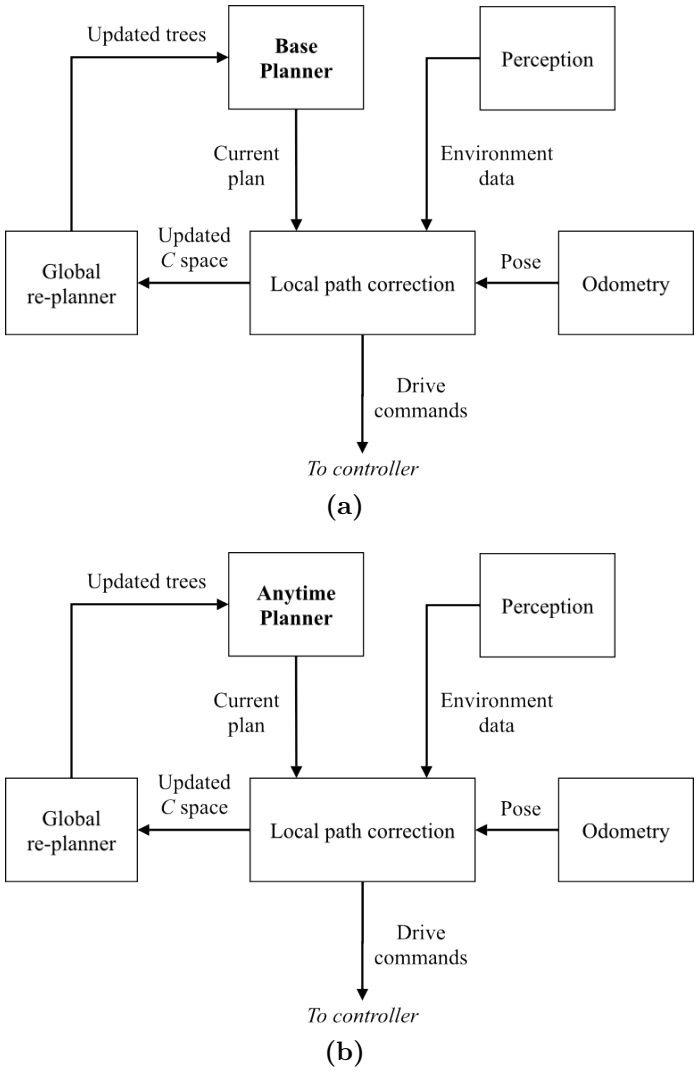


Figure 5.3: Dynamic planning extension, (a) Re-planning sub-routines integrated with the base planner, (b) Re-planning sub-routines integrated with the anytime planning architecture.

5.3.3 Dynamic Re-planning

Dynamic re-planning capabilities can be achieved through the extensions to the base planner architecture as shown in Fig. 5.3. These extensions are made up of two key components: a local path correction sub-routine and a global re-planning sub-routine (a detailed description of the underlying algorithms for these two sub-routines are provided in Section 5.4). In addition to these two additions, a source of external perception is required to observe the robot's environment (represented by the block *perception*).

These extensions to the base planner provide the following dynamic re-planning

behaviour. During execution, whenever unknown obstacles are detected by the sensing system, the planner evaluates whether the new obstacle region in the C-space would result in collision with the currently executed path. If no collision is detected, the planner ignores the obstacle and no re-planning is performed. In the reverse case, if the current path would lead to collision with the obstacle, the local path correction sub-routine is called to find a new collision-free path to the robot's current goal landmark l_j. Once this path is obtained, the planner compares the cost of the new re-planned path $c_p(\sigma_j)_{new}$ against the cost of the previous path to the current goal $c_p(\sigma_j)_{prev}$. An instance of global re-planning is subsequently performed if $c_p(\sigma_j)_{new}$ is greater than a minimum cost bound C_s^-, defined by:

$$C_s^- = \eta_d \cdot c_p(\sigma_j)_{prev}. \tag{5.2}$$

where η_d is the *dynamic bound constant* with a value greater than or equal to 1. The global re-planning sub-routine seeks to find a new optimal task plan by treating the new obstacles as permanent obstacles in the environment. All path plans are checked and updated if necessary to ensure no collisions occur as a result of the changes detected in the environment (new paths are also obtained by finding connecting paths between the robot's current location to all landmarks). A task planning query is then performed on the updated set of path costs to determine a new solution to the optimal ITPP problem.

Note that when $c_p(\sigma_j)_{new}$ does not exceed C_s^-, global re-planning does not occur. The planner resumes execution using the updated local path with the existing task plan intact. In this case, the detected obstacle information is discarded after the local path correction sub-routine.

The purpose of Eq. 5.2 is to limit the number of unnecessary global re-planning queries requested during execution (as this is an expensive sub-routine that results in noticeable idle time in real-time applications). Generally, small hazards (such as rocks, potholes or even other moving subjects) detected along a robot's path do not significantly impact the quality of an existing high-level action sequence, though the low-level path must still be corrected to avoid such things as collision. In these situations we would not expect the cost of the corrected path to be drastically more expensive than the original path. In contrast, bigger obstacles may require the robot to take large detours in order to reach its original goal landmark (consider driving as an analogy, where significant diversions can

result from a single roadblock). In these situations, it may be less costly to perform a reordered sequence of tasks. However, this requires the updating of all paths to ensure the new task plan takes into account the newly detected obstruction for all actions. The relative balance between these two behaviours is controlled by the value of η_d. Smaller values favor performing global re-planning to optimise the task plan at the cost of longer idle times, while larger values favor performing only the local path correction sub-routine for better real-time performance but at the cost of potentially lower quality plans.

The dynamic re-planning extension to the base planner can be integrated with the anytime planning extension by simply replacing the base planner module in Fig. 5.3a with the anytime planning procedures described in 5.3.2 (see Fig. 5.3b). Using this combined architecture, the planner continues to improve the quality of solutions by expanding the trees of the path planning layer during execution. When new obstacles are encountered, the re-planning sub-routines of the dynamic re-planning extension are called to update the currently executed path and subsequently the entire task plan (when the condition in Eq. 5.2 is met) using the most up-to-date set of trees expanded by the anytime component.

5.4 Extensions to the Multi-T-RRT* Algorithm

This section describes the sub-routines that are essential to the anytime and dynamic components of the DA-TPP framework. I first present the algorithm for the local path correction sub-routine, which re-plans purely in the low-level motion planning layer. I then describe the global re-planning sub-routine, which performs re-planning procedures both in the task planning layer and in the motion planning layer. Several functions used in the local path correction sub-routine is adopted here. Finally, I introduce a tree pruning extension to the Multi-T-RRT* algorithm that periodically deletes nodes that are deemed not useful for the search of an optimal solution to any path planning problem. This enables the size of the trees to be capped at a specified upper limit, which significantly improves the memory complexity of the planner and the time complexity of the aforementioned re-planning sub-routines.

Note that in the following, the notation used in Chapter 4 for the description of the Multi-T-RRT* algorithm is preserved.

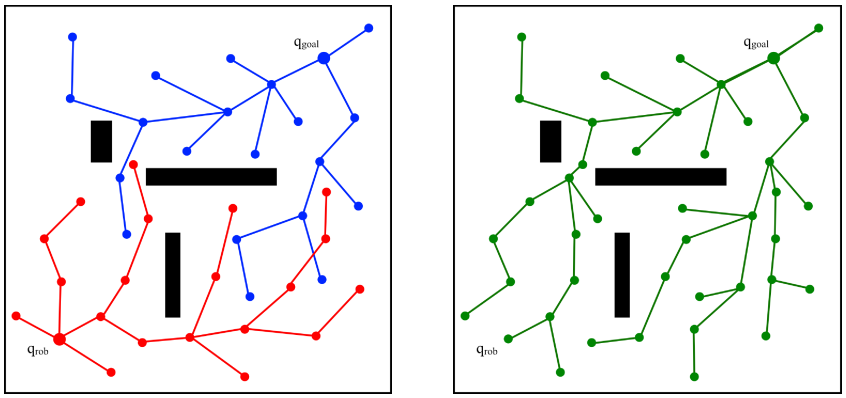
5.4.1 Local Path Correction

The local path correction sub-routine can be considered a **dynamic path planning** extension to the Multi-T-RRT* algorithm, which enables a new optimal path to be found as new obstacle information is provided to the motion planner. It involves a set of RRT tree update procedures that closely resembles a number of functions in the RRT$^{\text{X}}$ algorithm presented by Otte and Frazzoli [58].

To enable the use of the local path correction routine, the representation of relevant sampled configurations in the search space must be adjusted as follows. Taking the RRT tree T_0 rooted at the start landmark l_0 and the RRT tree T_g rooted at the goal landmark l_g, a new tree T_{merge} is generated by combining the nodes in T_0 and T_g into a single tree, rooted at l_g. As previously noted in Section 2.1.2, setting the root of an RRT tree as the goal location provides better support for dynamic re-planning as the root of the tree does not need to be updated to reflect the changing robot configuration. Without merging T_0 and T_g, any dynamic re-planning procedures would be required to update T_0 at each query to take into account the new configuration of the robot. Furthermore, additional complexity would be introduced to the re-planning procedures as new connections between the two trees must be established. Instead, by using a single tree T_{merge}, we can avoid the aforementioned difficulties. To perform this merge, a new tree is created with a single node at l_g. An iterative expansion process is applied to find the neighbours of l_g from among T_0 and T_g, and choosing the node with the lowest travel cost from l_g to add to the tree. This new node becomes the next node to expand, with the algorithm continuing to select the next neighbour node with the lowest cost to l_g through the iteratively-constructed T_{merge}. The result is an RRT*-like tree where every node in the tree is connected to the branch that provides the lowest cost to l_g from among the sampled configurations. An example of T_0 and T_g before merging and the resulting tree T_{merge} after the merge function is shown in Fig. 5.4.

Now consider a path σ due for execution. As the robot advances along σ, a collision checking query is conducted on the remaining segments of the path each time a new set of obstacles O is detected. If collision is found at any point along the path, the algorithm proceeds with the remaining procedures for path re-planning. Otherwise, O is discarded.

The remaining path re-planning procedures is as follows (see Algorithm 2 for the pseudo-code of the local path correction sub-routine). An *invalidateNodes*



(a) **(b)**

Figure 5.4: Illustration of the tree merge function, (a) Trees T_0 (red), rooted at q_{rob}, and T_g (blue), rooted at q_{goal}, grown across the configuration space. (b) The resulting tree T_{merge} (green) resulting from the merge function applied on T_0 and T_g.

function (Line 5 of Algorithm 2) obtains all the nodes in T_{merge} that comes into collision with any obstacle $o \in O$ and marks these as invalid (which is equivalent to removing them from T_{merge}). This results in a set of disconnected nodes and branches that can no longer be reached from the root of T_{merge}. The *updateOrphans* function (Line 6 of Algorithm 2) identifies all of these **orphaned** nodes in the tree, removes any connections with parent nodes and assigns an infinite travel cost (from the root of the tree) to these nodes. These two functions collectively resembles the *propagateDescendants* function of the RRT$^{\text{X}}$ described in [58]. Afterwards, the algorithm establishes new connecting edges between the orphans and the remaining valid nodes in the tree by iterating through a queue of nodes, called the **rewire queue**, consisting initially of the neighbours of the orphaned nodes. This rewire queue is ordered according to the travel cost of the nodes from the root of the tree (lowest first). At each iteration, the front node of the rewire queue is dequeued and an instance of rewiring is performed on its neighbour nodes in an attempt to reconnect orphaned nodes. Any neighbour node that is successfully rewired is added into the rewire queue. The algorithm continues to iterate through the rewire queue until it is empty, at which point the algorithm advances to the next step in the sub-routine. These procedures are encapsulated in the *rewireTree* function (Line 7 of Algorithm 2) and resembles the *reduceInconsistency* function in the RRT$^{\text{X}}$ algorithm. Subsequently, the robot's current configuration q_{rob} is added to T_{merge} using a standard tree expansion procedure. A new path σ_{new} from q_{rob} to l_g is then obtained by searching the T_{merge}.

Algorithm 2 localPathCorrection

Input: Merged tree T_{merge}, set of all planning trees T and current path σ
Output: Updated path σ_{new}

```
1: O_new ← getObstacles()
2: q_rob ← getRobotPose()
3: if collision(σ, O_new) then
4:    c_p(σ) ← pathCost(σ, q_rob)
5:    T_merge ← invalidateNodes(T_merge, O_new)
6:    T_merge ← updateOrphans(T_merge)
7:    T_merge ← rewireTree(T_merge)
8:    σ_new ← updatePath(T_merge, q_rob)
9:    c_p(σ_new) ← pathCost(σ_new, q_rob)
10:   if c_p(σ_new) > (1 + η_d) · c_p(σ) then
11:      globalReplanning(T, O_new, q_rob)
12:   end if
13: end if
```

Once a new path is obtained, the new path cost $c_p(\sigma_{new})$ is compared with the lower cost bound C_s^- as defined in Eq. 5.2. If $c_p(\sigma_{new})$ exceeds C_s^-, the global re-planning sub-routine is called to permanently propagate the current perceived state of the world to the task-level planner. Otherwise, the algorithm discards O such that their existence in the world is ignored until collision with these obstacles are again detected. This behaviour offers the advantage of delaying re-planning procedures for future steps as the effect of these obstacles are minimal to the overall cost of the executed set of motions. In this way, small dynamically-moving obstacles that are likely to have changed states when the robot returns to the same area do not induce expensive global re-planning procedures.

5.4.2 Global Re-planning

The global re-planning sub-routine is comprised of a global update to the set of path solutions in the motion planning layer and a subsequent task planning query using the new action costs derived from the updated motion planning layer. It takes as input the set of path planning trees T, the set of newly detected obstacles O and the robot's current configuration q_{rob}.

With reference to Algorithm 3, the global update to the motion planning layer consists of finding new optimal paths between every pair of landmarks in the Multi-T-RRT* path planner. When called, every tree $T_k \in T$ is updated by

Algorithm 3 globalReplanning (Motion Planning Layer)

Input: Set of all planning trees T, set of new obstacles O_{new} and the current robot configuration q_{rob}
Output: Set of updated planning trees T and set of path solutions Σ_{best}

1: **for all** $T_k \in T$ **do**
2: $T_k \leftarrow invalidNodes(T_k, O_{new})$
3: $T_k \leftarrow updateOrphans(T_k)$
4: $T_k \leftarrow rewireTree(T_k)$
5: **end for**
6: $\Sigma_{best} \leftarrow updatePaths(T)$
7: $\Sigma_{best} \leftarrow pathsFromRobotToLandmark(\Sigma_{best}, T, q_{rob})$

applying the same procedures in the local path correction algorithm (Lines 5-7 in Algorithm 2) to invalidate nodes in collision with O, update orphaned nodes, and re-establish connections through rewiring in conjunction with the rewire queue. The optimal paths between landmarks are obtained by finding the connecting nodes between associated trees and selecting the lowest cost path (the **set** of best paths for all the ITPP problem is represented by Σ_{best}).

In general, the robot's current configuration q_{rob} does not coincide with any of the landmarks in L. In order for the task planner to find a valid action sequence that starts from the robot's current configuration, q_{rob} is inserted into the ITPP problem as a virtual landmark l_{rob} (subsequently a new landmark object is also added when generating the PDDL problem file in the task planning stage). Using the updated trees associated to each of the original landmarks, attempts are made to connect l_{rob} to the original set of landmarks by testing connections from the neighbouring nodes of each tree to l_{rob}. Successful connections lead to the addition of a continuous path from q_{rob} to the corresponding landmark, while unsuccessful connections indicate that a path has not been found (i.e. infinite cost as assigned to the corresponding action during task planning). In this way the integrated planner attempts to solve a new task planning problem involving a different robot starting position, while all goals in the task remain the same as the initial planning problem.

Finally, a new action-motion sequence Ψ_{new} is obtained by generating a new PDDL problem file and conducting another instance of task planning.

5.4.3 Tree Pruning

A consequence of the Multi-T-RRT* algorithm's asymptotic optimality property is the requirement for extensive sampling to effectively explore the search space. This means that converging to the set of optimal solutions during path planning requires many nodes to be added to the set of trees, T. However, as the number of nodes n in the trees increase, the computational resources consumed by the planner also increases (in terms of time and memory). This is universally true for all RRT*-based algorithms that involve a rewiring procedure. In fact in a complexity analysis of the RRT* algorithm conducted by the original authors, it was found that the time complexity of the RRT* was $\mathcal{O}(n \log n)$. [43]

While this time complexity is tolerable for static applications, the same cannot be said for dynamic re-planning, where planning efficiency is particularly important if near-real-time performance is desired. Both the local path correction and global re-planning sub-routines involve rewiring procedures and are therefore computationally sensitive to n.

Karaman et al. [61] had proposed the use of a branch-and-bound technique for the standard RRT* to enable more efficient expansion in their anytime implementation of the algorithm. Branch and bound is a popular technique in the fields of AI and optimisation, where candidate solutions whose costs are worse than the current best solution are automatically disregarded. In the context of the RRT* algorithm, this refers to sampled nodes in the tree where the cost of any path from the start to the goal that passes through the node will always be larger that the current best path. Karaman et al. proposed the periodic deletion of these nodes to limit the size of the tree as these nodes cannot be part of a shorter path. Similarly, recall from Section 2.1.2 that Otte and Correll had proposed the use of tree pruning to remove redundant nodes from a parallel set of RRT* trees [54]. This was performed each time a new best solution was found, allowing all nodes that do not provide better solutions even at best (according to the same criteria as branch-and-bound used by Karaman et al.) to be identified and removed. Both of these techniques were shown to reduce the memory consumption and rate of convergence of their respective algorithms by minimising the number of nodes stored in memory and limiting the number of rewiring operations required in a single iteration of tree growth. However, in both cases authors had only considered the use of this concept for a single path planning query.

In general, it is difficult to apply a tree pruning technique directly in the context of solving multiple path planning problems simultaneously. To explain why, consider three landmarks l_A, l_B and l_C. Suppose we inspect a sampled node q_a that belongs in tree T_A. Using an admissible estimate of the cost to go to l_B, let us suppose that no solution through q_a can provide a lower cost solution to l_B. However, in order to conclude that q_a cannot offer better solutions universally, we must also prove that q_a does not have the potential to offer better solutions for a path to l_C. As a result, the likelihood of a sampled node being removed falls as it must now meet two criteria. Following this analysis then, as the number of landmarks in the problem increases, the likelihood of a node being deleted tends towards zero (that is, for a problem with many landmarks, a node might never be completely irrelevant as it will always advance towards at least one landmark).

In this work, I extend the concept of tree pruning to the Multi-T-RRT* algorithm to place an upper bound on the number of nodes across the entire set of T-RRT* trees while preserving the capability of the algorithm to improve solutions over time. This enables a substantial reduction in the computation time required for re-planning (as less rewiring procedures are required in each instance due to the upper bound on number of nodes) and minimises the memory consumption of the algorithm. To address the problem of multiple goals as described above, a heuristic procedure was developed to evaluate the usefulness of nodes within the context of multiple trees. The method works as follows. Let N_{max} be a predefined upper bound on the number of nodes n across the set of trees T. When n is less than N_{max}, the expansion of the Multi-T-RRT* continues as normal. Once n reaches N_{max}, the addition of a new node to a tree can only be accepted under the condition that an existing node of less usefulness within the same tree has been found. Using this method, the number of nodes in the Multi-T-RRT* is capped at N_{max} without preventing the algorithm from continuing to explore the search space for better solutions.

Letting q_{new} represent a newly sampled node, the conditions for searching for an existing node of less usefulness are described below in chronological order:

Condition (Condition 1)**.** *A search within the neighbours of q_{new} is conducted to find a node with only one child, and where the path cost from the tree root to the child is less by passing through q_{new} instead. If no nodes that meet this criteria are found, move to condition 2.*

Condition (Condition 2)**.** *A search within the neighbours of q_{new} is conducted to find any nodes without children that possesses a higher* ***cost space*** *value than q_{new} (recall that the Multi-T-RRT* algorithm samples nodes using a transition test that accounts for a continuous cost space). If no nodes that meet this criteria are found, move to condition 3.*

Condition (Condition 3)**.** *A search across the tree is conducted to find a node that has no children and has a lower usefulness value ρ than q_{new} (explained below). If no nodes that meet this criteria is found, then q_{new} is rejected.*

Note that in addition to each of the 3 conditions above, the selected node must also not be a connecting node to other trees, otherwise it is considered as a useful node. Once a node for deletion is found, the existing node is removed from the tree and the new node q_{new} is added. Additionally, if the deleted node was found in step 1, a connecting edge is formed between the deleted node's child and q_{new}. Finally, the algorithm resumes with the normal procedures by performing rewiring on the neighbours of q_{new}. The complete process for tree-pruning is summarised in the flowchart shown in Fig. 5.5.

In condition 3 of the search procedures listed above, a heuristic usefulness value ρ was introduced to enable a comparison of existing nodes with q_{new}. This value was computed in the following way. Let i be the index of the tree for which q_{new} was sampled for, and let *costToGo(A,B)* be an admissible estimate of the cost from A to B (e.g. a straight segment that connects A to B, where the distance is given by the Euclidean metric). ρ is then given by:

$$\rho = \min_{k=[1:m],k\neq i} \frac{c_p(\sigma(l_i, q_{new})) + costToGo(q_{new}, l_k)}{c_p(\sigma_k)_{best}} \tag{5.3}$$

Where m is the number of landmarks (and trees) in the ITPP problem, $\sigma(l_i, q_{new})$ is the path from the root of T_i to q_{new} and $c_p(\sigma_k)_{best}$ is the cost of the current best solution path for landmark k. In other words, the usefulness value measures the best case solution quality of any path passing through a node relative to the current best solution for the corresponding path goal. Thus in step 3 of the search procedures, rather than deleting a node with no potential to improve any solution path (which may not exist), the algorithm chooses to delete a node with lower potential to improve the path quality of a solution compared to the newly sampled node.

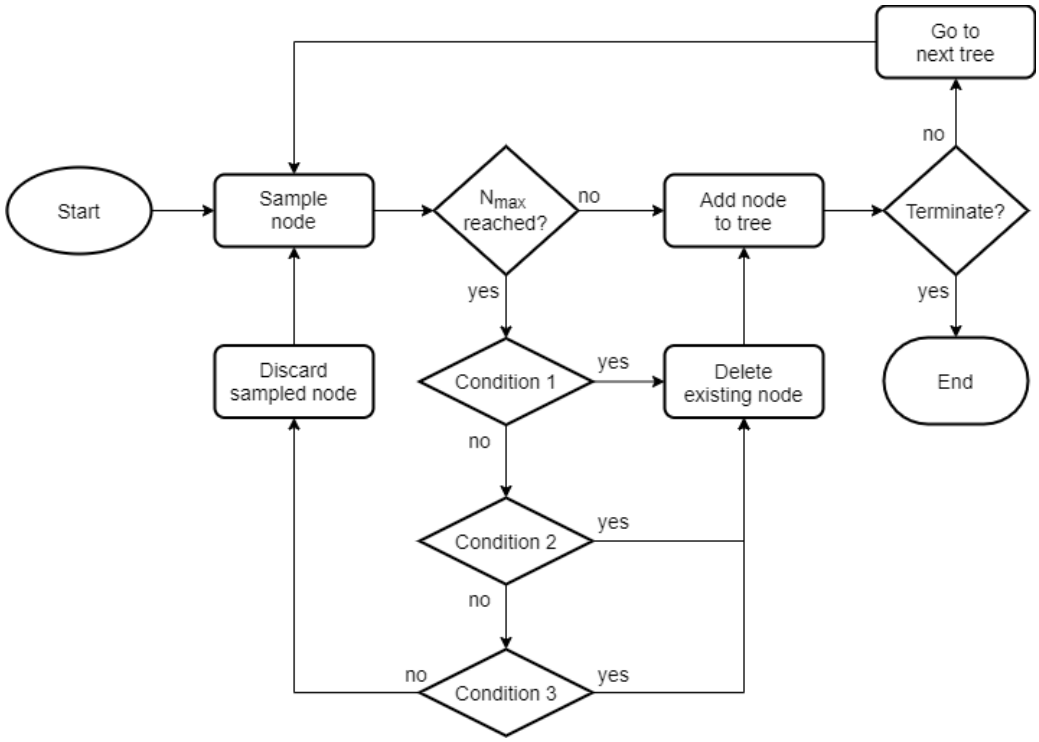


Figure 5.5: Flowchart showing the tree pruning procedure for adding nodes into a tree while limiting the maximum number of nodes.

The tree pruning procedures described in this section can be implemented to the base planner, anytime planner or dynamic planner of the DA-TPP framework. When searching for high quality solutions, the use of tree pruning places an upper limit on the computational resources of the planner (in terms of memory) and improves the overall efficiency for converging towards optimal solutions. For dynamic planning applications on the other hand, tree pruning improves the near-real-time performance of the re-planning sub-routines by limiting the size of trees that must be updated while enabling the search for high quality solutions.

I will show in Section 5.5.2 that the use of tree pruning allows solutions of similar quality to be found with less nodes across the entire set of trees compared to extended sampling without tree pruning.

5.5 Experimental Evaluation

A number of experiments were conducted to evaluate each of the components of the DA-TPP framework. In the first experiment I demonstrate the behaviour of the anytime planner and present an evaluation of its performance for various η_a

values. Through the second experiment I show that the tree pruning technique introduced as an extension to the Multi-T-RRT* algorithm enables the path planner to find solutions of comparable quality to the same algorithm without tree pruning, but with a smaller number of total nodes across the entire set of trees. Finally, I present a third experiment conducted on a physical robot to demonstrate the behaviour of dynamic re-planning in an office environment with unknown obstacles placed across the environment.

In all three experiments reported in this section, a PC carrying an Intel® Xeon® CPU E3-1270 v3 (3.50 GHz) was used to run the DA-TPP algorithms implemented on Matlab version R2016b software.

5.5.1 Anytime Planning

To demonstrate the behaviour and performance of the anytime planner, a set of trials was conducted on the ITPP problem shown in Fig. 5.6 using the anytime planning procedures described in Section 5.3.2. In this experiment, the value of the anytime bound constant η_a was varied between 0.80, 0.85, 0.90, 0.93, 0.95 and 0.97. For each η_a value, 50 trials were conducted for statistical significance with an allotted planning time of 200 seconds. The task planning layer was allocated a search time of 0.5 seconds for the LPG-*td* planner. Furthermore, to eliminate the effects of randomness as much as possible in the evaluations (the Multi-T-RRT* and LPG-*td* are both random in nature), two additional precautions were taken. Fistly, the random number generator in Matlab was reset for each trial. Since Matlab's implementation of random number generation is actually based on a deterministic algorithm, by resetting the random number generator we can guarantee that the same sequence of randomly generated numbers are used in each trial. This means that the only true unpredictability that leads to variation between each run is the computation speed, which varies with each run as the PC is not a real-time system. Secondly, the initial plan obtained by the base planner in the very first trial was used as a starting point for all subsequent trials. This removes variability in the initial solution, which minimises the effect of randomness on the final comparison.

Fig. 5.8 shows the plan cost obtained by the anytime planner after 200 seconds for each of the η_a values, with the distribution of results over the 50 trials represented by a box plot. Immediately we can see that better quality solutions

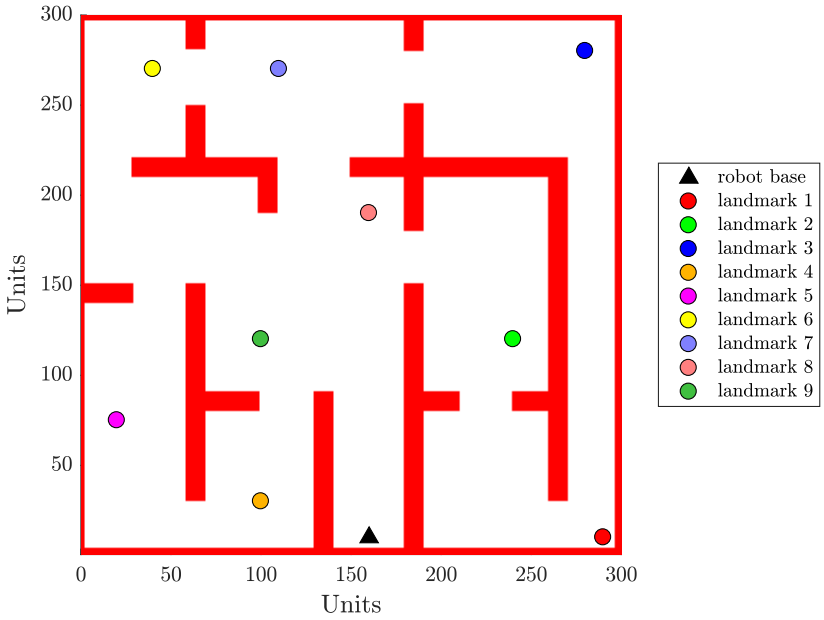


Figure 5.6: Environment and corresponding ITPP problem used for the anytime planner evaluation.

are consistently found by the planner with η_a values closer to 1. Note, however, that the lowest plan cost achieved across the 50 trials are comparable across all values of η_a (inclusive of outliers). These observations are unsurprising, since for the limit case of $\eta_a = 1$, the planner would perform an instance of task planning every time a better path solution was found, which maximises the **likelihood** of finding the optimal action sequence. Nevertheless, it is possible for the planner to converge to the optimal action sequence across all of these η_a values if sufficiently high quality paths are found in the path planning layer at the time when the upper bound criteria is met.

In Fig. 5.8 the number of task-level re-plans performed by the end of the 200 seconds planning duration is reported for each η_a value, where the distribution over 50 trials is again represented by box plots. Notice this time that as the value of η_a approaches 1, more task planning queries are performed by the planner (the variance over the 50 trials also increases with η_a). This is an indication of the computational efficiency of the planner for converging to a stable action sequence (though as observed above, this may not be the **optimal** sequence). For smaller η_a values, a larger improvement in the quality of path solutions is required to fulfill the upper cost bound criteria for task re-planning. In the extreme case of $\eta_a = 0.80$, this consistently occurs once only over a period of 200 seconds. On

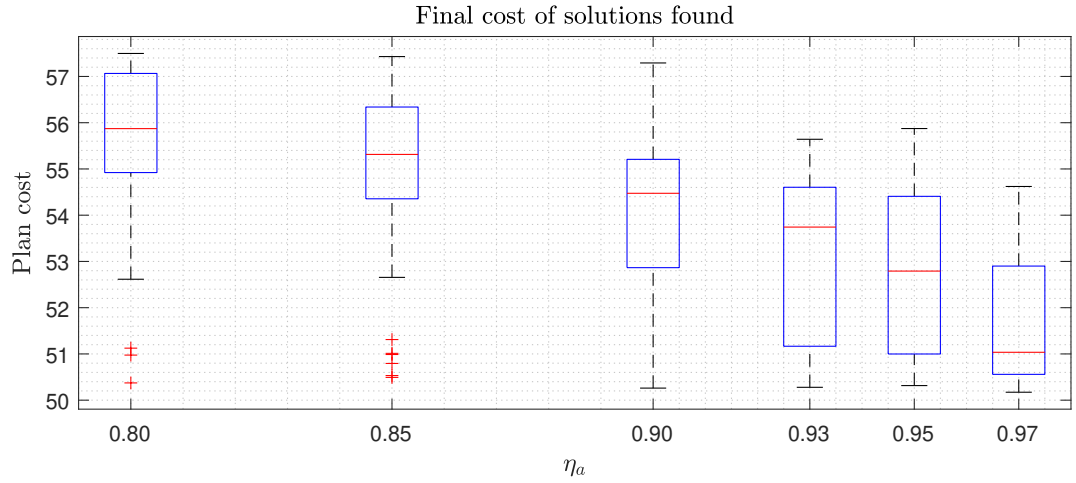


Figure 5.7: Plan cost of solution found over 50 runs for various η_a values.

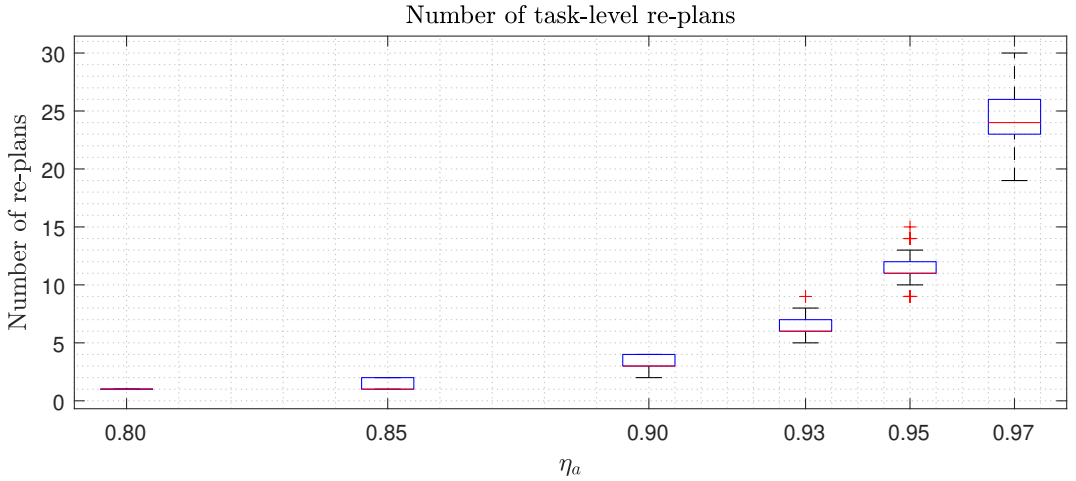


Figure 5.8: Number of re-plans performed in anytime planning across 50 runs for various η_a values.

the other extremity, a much larger number of task re-planning instances (up to 30) was performed within the 200 seconds duration for $\eta_a = 0.97$. Consequently, while the likelihood of finding the optimal action sequence is higher for these trials, less time is devoted to the expansion of the Multi-T-RRT* algorithm to improve the quality of path solutions (a minimum of 15 seconds is required for 30 calls to task planning assuming the minimum search time of 0.5 seconds taken for each query to the task planner).

Let us further examine these observations in Fig. 5.9, where the plan cost improvements for a single trial against time is shown for each η_a value.

As we have already seen, the quality of the final plan reached by the end of

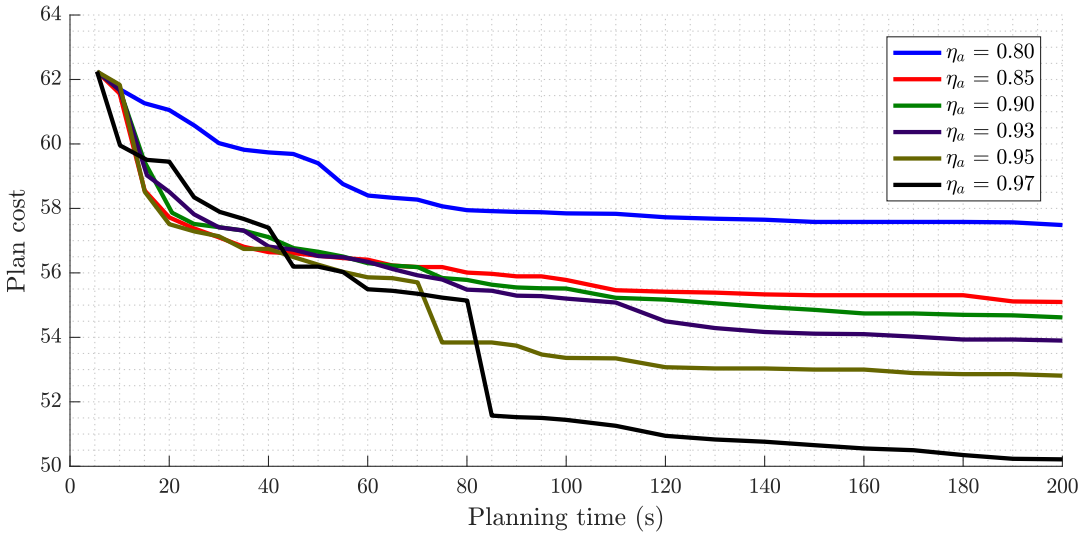


Figure 5.9: An example of the improving plan costs over 200 seconds for varying η_a values.

the 200 seconds is generally better for larger values of η_a. In addition to this, the implication of the increasing number of task planning queries performed by the planner is observable as a slower convergence towards the solution found in each trial. While the quality of the overall solution continues to improve slowly due to refinements made at the path planning level, we can see that the solution cost of trials involving smaller η_a values generally flatten out faster than the trials involving larger η_a values. This is a consequence of the increasing number of task planning queries performed, which reduces the overall efficiency of the planner to reach a stabilised action sequence in the task planning layer.

From a practical point of view, this is generally not a major consequence if anytime planning is performed in parallel with task execution. Since MWRs are normally not fast moving robots (compared to robots such as drones or manipulators), the time required to complete a task is generally at least several minutes long. This provides sufficient time for the anytime planner to continue searching for better plans even for large η_a values.

5.5.2 Tree Pruning

Let us now examine the effectiveness of the tree pruning technique for reducing the computational resources required by the Multi-T-RRT* to improve the quality of a solution. Using the same ITPP problem shown in Fig. 5.6, the base planner was used to obtain an initial solution with the termination criteria of the Multi-

T-RRT* algorithm set to return the solution when the number of nodes across all trees reach n_a. This initial solution was saved for later re-use. The base planner was then allowed to continue iterating through the expansion procedures of the Multi-T-RRT* algorithm without tree pruning until a second termination criteria was met such that an improved solution was returned when the number of nodes reach n_b. I refer to this as *standard tree expansion*. I then revert back to the initial solution obtained with n_a nodes. Iterations of the expansion procedures were again performed but this time the tree pruning technique described in Section 5.4.3 was applied to the Multi-T-RRT* algorithm with the number of nodes fixed at n_a. Planning continued for the same length of time as what was required for standard tree expansion to reach n_b nodes. I refer to this as *expansion with tree pruning*.

50 trials were conducted for n_b set to 10000, 20000 and 30000 nodes, while n_a was fixed at 3000 nodes for all three instances. Fig. 5.10 reports the plan costs for the initial solution, the solution obtained by standard tree expansion, and the solution obtained by expansion with tree pruning. Like before, the distribution across 50 trials are represented by box plots. In Fig. 5.10a ($n_b = 10000$), we find that both methods of further expansion provides solutions of better quality than the initial solution, though on average using standard expansion provides a slightly higher quality solution. In Fig. 5.10b ($n_b = 20000$), the average cost of solutions obtained by both approaches for further expansion were approximately identical. However, on two occasions the expansion with tree pruning found solutions that were more costly than the maximum cost achieved with the standard tree expansion. Finally, in Fig. 5.10c ($n_b = 30000$), in addition to finding approximately equal average plan costs, the expansion with tree pruning always found a solution with a cost within the range achieved by standard tree expansion across all 50 trials.

Table 5.1 reports the memory required to store the set of T-RRT* trees at the end of expansion in each instance. Clearly, with tree pruning the storage memory required is close to that of the initial solution, while the memory required for standard tree expansion grows according to the number of nodes in the tree. Importantly, the increase in memory is non-linear relative to the number of nodes in the tree. This is due to the consequential increase in the number of branches between nodes and connections between trees. In the limit case of $n_b = 30000$ where tree pruning achieves solutions of comparable of quality to standard tree

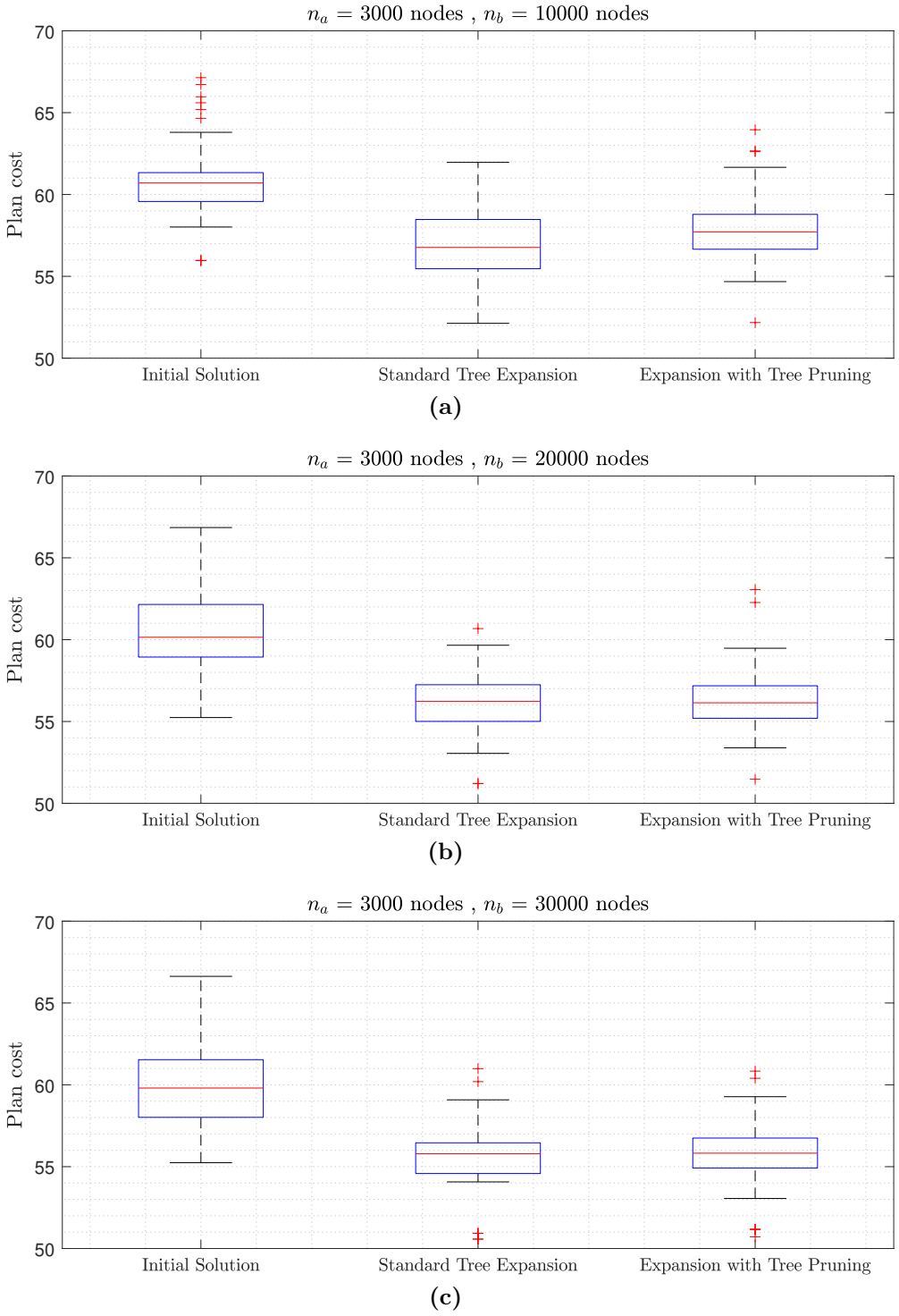


Figure 5.10: Plan cost of solutions obtained over 50 runs for initial solution, solution from standard tree expansion and solution from expansion with tree pruning, (a) $n_a = 3000$ and $n_b = 10,000$, (b) $n_a = 3000$ and $n_b = 20,000$, (c) $n_a = 3000$ and $n_b = 30,000$.

expansion, the total reduction in memory is 97.0%.

Figure 5.11: The Pioneer 3-DX differential drive mobile robot

Based on these observations, we can conclusively say that expansion with tree pruning can achieve solutions identical in quality to standard tree expansion given sufficient planning time, but with the added advantage of requiring substantially smaller search trees. As we have seen in Fig. 5.10c, only $1/10^{th}$ of the nodes were required by expansion with tree pruning to generate solutions of the same quality as standard expansion. Even in cases similar to Fig. 5.10a where expansion with tree pruning had not yet reached the same solution quality as standard tree expansion, tree pruning provides a computationally efficient way of improving the cost of solutions found by Multi-T-RRT* without consuming additional computational resources in terms of memory required to store the number of tree nodes. This significant improvement in the use of resources has important practical implications, particularly for lightweight systems such as on-board computers on MWRs with payload limitations. As the results in Table 5.2 have shown, memory reductions in the range of 97% can be achieved, sparing a substantial amount of computing resources for other processes performed on-board.

5.5.3 Dynamic Re-planning

To evaluate the behaviour of the dynamic component of DA-TPP, a physical experiment was conducted on a Pioneer 3-DX (P3DX) MWR (see Fig. 5.11). The P3DX is a differential drive mobile robot equipped with internal wheel encoders

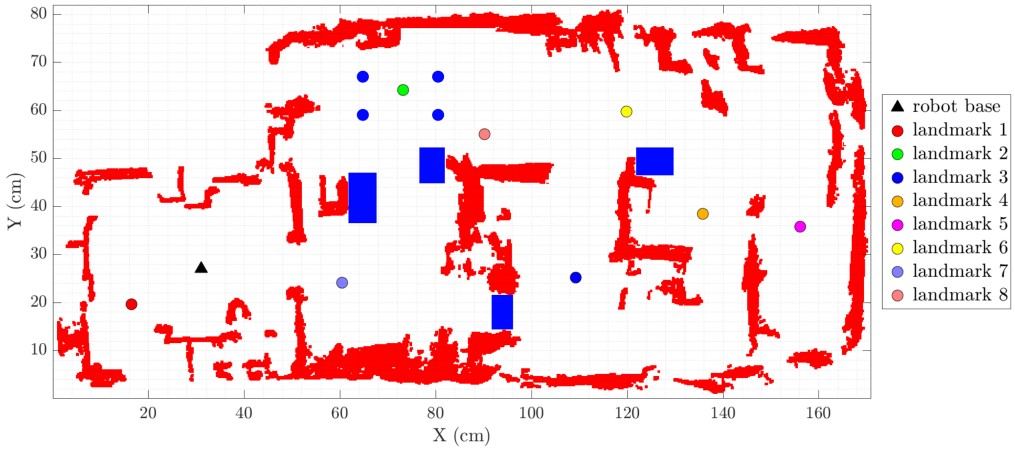


Figure 5.12: ITPP problem for dynamic re-planning experiment consisting of an a priori known indoor office environment mapped with SLAM using the Pioneer 3-DX robot and a Hokuyo URG-04LX-UG01 laser range finder (shown in red). A set of unknown obstacles were placed after mapping as shown in blue.

Table 5.1: Storage memory required for all trees

n_a	n_b	**Init. solution**	**Standard expansion**	**Tree pruning**
3,000	10,000	1,340 KB	8,648 KB	1,495 KB
3,000	20,000	1,340 KB	28,862 KB	1,704 KB
3,000	30,000	1,340 KB	60,529 KB	1,825 KB

to provide odometry-based localisation. A Hokuyo URG-04LX-UG01 laser range finder with a working range of 20 mm to 5600 mm was installed onto the P3DX to provide external perception of the environment. Communications between the host PC running the planner, the P3DX robot and the Hokuyo laser range finder was achieved using the ROS framework [167]. An indoor office environment was set up initially without movable obstacles and a preliminary map of this environment was obtained by mapping the environment using simultaneous localisation and mapping (SLAM). Once the mapping was complete, a set of unknown obstacles was placed in the environment such that the previously generated map was only a partial representation of the true environment. The subsequent ITPP problem is illustrated in Fig. 5.12, where known obstacles are shown in red while unknown obstacles are shown in blue.

The experiment consists of finding an initial plan to visit all landmarks using

the base planner with the Multi-T-RRT* algorithm set to return a solution when the number of nodes n in the tree reached 3000. This initial plan was sent to the robot for execution once obtained. Using the fully-integrated dynamic and anytime planning components of the DA-TPP framework (as shown in Fig. 5.3b), the planner continued to expand the Multi-T-RRT* search trees using the tree pruning method, with $N_{max} = 3000$ and $\eta_a = 0.95$. As the robot achieved each task, a request was made to the anytime planner to retrieve the latest plan for the remaining set of tasks to be completed. During execution, each time the robot encountered an unknown obstacle that interfered with its current path, the anytime expansion procedures were interrupted to run the re-planning sub-routines. An instance of local path correction was first performed to find a new collision-free path to the current goal landmark. Then, with $\eta_d = 1.05$, the minimum cost bound criteria (Eq. 5.2) was used to determine whether global re-planning was required. When called, the global re-planning procedure solved a new instance of the ITPP problem using the robot's current location as a new starting point. Anytime planning procedures resumed as normal on completion of the dynamic re-planning sub-routines. Throughout this experiment the maximum velocity of the mobile robot was set to 0.15 m/s.

The results of the experiment are summarised in Fig. 5.13 and Table 5.2. Fig 5.13a shows the initial plan computed by the base planner component of the DA-TPP framework along with the Multi-T-RRT* search trees for $n = 3000$. Here the initial visiting sequence of landmarks is[2]:

$$p_{init} = \{l_0, l_2, l_8, l_4, l_5, l_6, l_3, l_7, l_0, l_1, l_0\}$$

In Fig. 5.13b, the true executed sequence of paths are shown together with (i) the set of unknown obstacles overlaid in the environment and (ii) the set of search trees at the end of execution, with n maintained at 3000 nodes through tree pruning. The *true* executed visiting sequence of landmarks is:

$$p_{exe} = \{l_0, l_2, l_8, l_6, l_4, l_5, l_4, l_3, l_7, l_0, l_1, l_0\}$$

[2]For convenience, I have represented the sequence of landmarks using the same notation defined for an action sequence. However, the *true* action sequence should instead comprise of the movement actions that connect each pair of successive landmarks. Let us assume that the appearance of $\{...l_i, l_j...\}$ implies a movement action $a_{i,j}$ that involves moving from landmark l_i to landmark l_j.

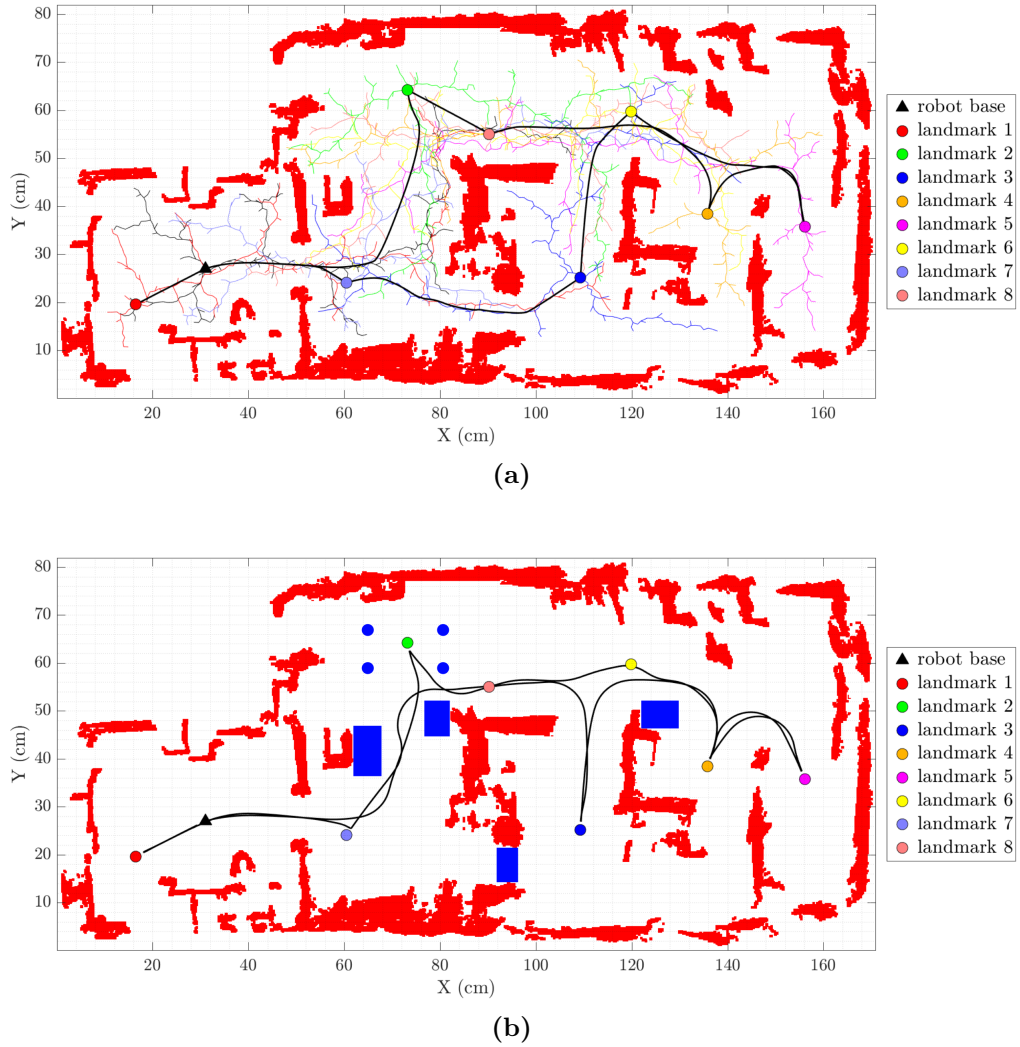


Figure 5.13: Example results of dynamic re-planning application in an office environment, (a) initial plan with only partial awareness of obstacles in the environment, (b) true executed paths using the DA-TPP framework (originally unknown obstacles are overlaid in blue).

Observe that the final executed sequence of motions have been adapted both in the low-level paths (e.g. when moving from l_2 to l_8) and at the high-level task plan. These changes primarily arise from re-planning in response to the detection of interfering obstacles. However, minor changes can also be observed in sections of the plan where obstacles did not trigger any re-planning sub-routines. For example, it is visually observable that the executed path for navigating from l_4 to l_5 does not match the initially planned path. These observable changes are a result of continuous improvements made at the level of path planning through the anytime planning procedures.

In Table 5.2, a number of key performance indicators for this experiment are reported, which includes:

1. the total length of the initial plan
2. the total length of the executed plan
3. the number of global re-planning calls
4. the number of local path correction calls
5. the planning time required to obtain the initial solution
6. the average CPU time for global re-planning
7. and the average CPU time for local path correction.

As we may expect, the total distance travelled in the true executed plan is longer than the total path length of the initial plan. This extra distance travelled was incurred through the additional path segments required to navigate around obstructions (most noticeable when navigating from l_3 to l_7). Furthermore, in this scenario the number of global re-planning calls is identical to the number of calls to the local path correction sub-routine. In other words, global re-planning was triggered every time a new interfering obstacle was detected.

With regards to planning efficiency, notice that the average CPU time required for global re-planning is 57% less than the CPU time required to generate the initial solution. When we take into consideration the number of global re-planning calls made over the entire duration of the task execution, global re-planning reduced the planning time by 17.7 seconds compared to re-planning from scratch (i.e. performing a fresh instance of planning each time an obstacle was detected). The sub-routine was able to achieve this by exploiting the previous exploration of the partial environment. This meant that existing information was preserved and re-used where possible to minimise the effort required to re-explore regions of the environment that had remained unchanged. While the improvements observed in this experiment conducted within the office environment can be considered moderate, the time-saving achieved through global re-planning can be very significant for more complex problems where generating an initial solution can require much longer planning times. Furthermore, when combined with the anytime planning component, global re-planning enabled the planner to exploit the additional

Table 5.2: Dynamic Re-planning Results

Name	Value
Initial plan length	448.96 cm
Executed plan length	456.97 cm
Number of global re-plans	5
Number of local path corrections	5
t_{CPU} for initial solution	6.19 s
$\bar{t}_{CPU}$ for global re-planning	2.65 s
$\bar{t}_{CPU}$ for local path correction	0.14 s

sampling obtained through continued tree expansion to obtain higher quality solutions. This could not be achieved if re-planning from scratch was performed instead.

As for local path correction, the average CPU time consumed by the subroutine was 0.14 seconds. This can be considered real-time according to the definition given in Chapter 3. In addition to this, local path correction enables the de-coupling of low-level path re-planning from global re-planning to enable an MWR to evade small (and possibly dynamically-moving) obstacles without re-evaluating its influence on the optimality of the global action sequence.

5.6 Summary

The DA-TPP framework is a probabilistically complete and asymptotically optimal task and path planner developed to address ITPP problems for MWRs. At its very core is a base planner that combines symbolic task planning with numeric cost functions derived from motion planning in the continuous geometric domain. Two additional components extend the capability of this planner according to the requirements of the application.

The anytime planning component enables to robot to retrieve an initial feasible solution to an ITPP problem within seconds (varying according to the complexity of the problem). Once retrieved, the anytime planner continues to refine the quality of the solution by improving the plan at the level of motion planning and in the higher level action sequence. The dynamic re-planning component enables online adaptive planning in dynamic scenarios (where environments may

either be partially-known or contain dynamically-moving obstacles etc.) and can be further divided into two sub-routines: local path correction and global re-planning. Local path correction handles re-planning at the level of motion planning for individual paths to ensure collision is avoided while traversing towards the current goal landmark. Global re-planning on the other hand seeks to adapt the entire action-motion sequence with the aim of maintaining an optimal plan for the remaining set of goal objectives that have not yet been met when changes in the environment are detected.

The capabilities of the framework has been validated through a number of experiments conducted to evaluate the individual aspects of the DA-TPP. Most notably, I have shown that the DA-TPP framework can enable **real-time** dynamic obstacle avoidance at the level of individually executed paths, while updates to a high-level task plan can be performed in response to changes in the environment through the use of global re-planning at a fraction of the planning time required for re-planning from scratch (on average, a 57% reduction was achieved in the experiment considered in this chapter).

From a practical implementation perspective, the anytime component of the DA-TPP algorithm can provide an initial plan for a robot to begin execution within seconds. Taking into account the generally lengthy duration of tasks performed by MWRs (as well as the individual traversals between task locations), by the time the robot is ready to execute the second task we can expect the anytime planner to have converged to a near-optimal action-motion sequence for the application's underlying ITPP problem.

Finally, this chapter has also introduced a tree pruning method for multi-tree based sampling algorithms that enable the search space to be further explored without consuming additional memory. This has substantial implications on the practicalities of the DA-TPP for deploying on-board mobile robots, which are often limited in computational resources. Since sampling algorithms are asymptotically optimal at best, without tree pruning a large number of samples may be required to converge towards an optimal solution. Through the use of tree pruning, this challenge can be overcome by placing a limit on the burden of the algorithm on computational resources (which can be adjusted by changing the maximum permitted number of tree nodes).

Chapter 6

Spatially-Constrained Robotic Task Sequencing

In the remaining two chapters, I tackle a specific sub-problem of task planning for manipulators called the Robotic Task Sequencing Problem (RTSP). Similar to the MTP, the RTSP involves finding an efficient tour to visit a sequence of task points defined in the task space, taking into consideration the kinematic redundancy of the manipulator. A fundamental difference in the RTSP compared to the previously studied task planning problem for mobile robots is the substantial increase in the number of goals in the planning problem. Common application domains for the RTSP typically consist of several hundred to thousands of task points that must be visited by a robot's end effector. Seeking to address this problem using the strategy described in Chapters 4 and 5, where the solutions to the motion planning problem between all possible combinations of task point pairs were first computed and subsequently used to obtain the task sequence, becomes intractable for RTSPs due to the massive search space arising from both large task point sets and kinematic redundancy. This renders the use of the Multi-T-RRT* (see Chapter 4) algorithm developed to simultaneously solve all path planning problems between multiple goals inapplicable in this problem domain[1].

While there exist numerous algorithms that solve a relatively relaxed version of the RTSP, where the number of task points involved are few in number and the environment is mostly clutter-free (see Section 2.2.3), they are generally inadequate for solving more complex problems that involve many task points and

[1]In the previous MWR domain we had only considered up to 9 landmarks in the environment.

where the robot is subject to significant spatial constraints.

In this chapter, I introduce the challenges of RTSP and present a novel clustering-based algorithm for efficiently solving *spatially-constrained* RTSPs. The algorithm, called Cluster-RTSP, is capable of handling large sets of task points and provides *near-optimal* solutions. The performance of the algorithm is evaluated through a number of benchmarks against an existing state-of-the-art approach both in simulation and through a case study conducted on a physical surface inspection-like task. I show that the algorithm is able to reduce computation time by up to 90% and the task execution time of the resulting solution by up to 60% when compared to the state-of-the-art.

6.1 Introduction

Robotic task sequencing is a fundamental problem that appears in one form or another within modern industrial robotics. Consider applications such as free-form surface inspection, thermal mapping, laser-welding, spray-painting and drilling [168,169]. A common aspect in all of these applications is the requirement for a manipulator to perform a large number of repetitive tasks at points defined across a workpiece. From a practitioner's perspective, developing an optimised motion sequence to complete these tasks efficiently is important for maximising process throughput, but obtaining this plan is no trivial matter. It is particularly challenging and time-consuming to determine an optimal task point visiting order and corresponding motion plans for tasks that involve hundreds or thousands of points, yet in practice this has still been predominantly dealt with manually by skilled programmers. This places limitations on the practicality of roboticising applications that involve: (i) rapid deployment in single-use instances where a developed plan is executed only once, making long offline planning particularly costly, (ii) highly variable tasks and processes that change frequently, rendering existing offline plans irrelevant, and (iii) unstructured environments beyond carefully designed industrial shop floors, where surrounding obstructions substantially limit the reachable workspace of the robot.

In the face of these challenges, numerous efforts have been made within the robotics research community to develop planning approaches that tackle the generalised version of these kinds of sequencing problems, which is what we now refer to as the RTSP. Conceptually, the RTSP describes the problem of computing a

sequence of robotic motions (optimised according to some performance criteria) that enable a robot to visit a set of task points by starting from (and returning to) a given home configuration. Certainly, the problem is closely related to the Travelling Salesman Problem (TSP) [101], which fundamentally also solves for a shortest visiting route for a set of 'cities'. However, recall from Section 2.2.1.4 that when considering TSP-like problems in the robotics domain, additional considerations must be taken into account in relation to a robot's *kinematic redundancy*. Put simply, the complexity of RTSP is increased by the existence of multiple robot configurations that can reach a single task point described in the task space. One may choose to formulate this type of problem as a GTSP, such that configurations are partitioned into groups according to the task point that is reached from each of these configurations and only one configuration from each group should be visited.

The quality of a tour is determined by the cost of the motions required to advance through each successive configuration within the sequence. In standard TSP problems, the common distance metrics considered for evaluating the quality of a tour are generally simple to compute. When applied to the RTSP, these metrics are only able to provide an approximation of the motion cost. In existing RTSP literature, some works have adopted these approximations with relative success [107, 132, 135], but those studies have only considered mostly uncluttered environments. When extending the application of algorithms to cases where the robot is subject to more substantial spatial constraints, these metrics no longer provide an effective approximation as they do not reflect the cost of more elaborate motions required to avoid collision. To obtain the true cost of motion for a single path in the TSP graph, it would be necessary to perform a path planning query between the two configurations involved. Problematically, solving the RTSP by exhaustively computing the true motion costs in this way would be too time-consuming in practice for tasks that involve even a moderate number of points, since the cost of motion between every single combination of configurations would need to be computed.

Some of the earliest work in the RTSP simplified the problem by arbitrarily assigning a random configuration to each task point, allowing the problem to be formulated as a standard TSP. This methodology would in general lead to sub-optimal solutions since C-space information is not considered. In fact, careful selection of configurations from among the inverse kinematic (IK) solutions for

each task point could enable better quality solutions to be found. Indeed more recent work have proposed approaches that objectively select configurations in an attempt to find better task sequences. Some authors have studied the use of global optimisation methods to achieve this, but these required considerably long computation times (in the order of thousands of seconds [126]) that strictly limit their use to offline programming. Nowadays however, modern applications demand methods that possess the flexibility to adapt to changing task parameters. In these cases long planning times would unfortunately lead to bottlenecks in the operational process.

In contrast to the above, the authors of RoboTSP [135] applied a decoupled strategy to solve the RTSP as a two-step problem involving firstly a task point ordering problem (solved in task space), and a configuration assignment problem (solved in C-space). By tackling the problem using both task space and C-space information, the authors were able to solve RTSPs involving several hundred points in the order of minutes. Nevertheless, their method is unable to account for the costs accumulated through collision avoidance (as I will show later in this chapter).

In this chapter I build upon these prior works and present a new, competitive method for solving RTSPs that is able to obtain **near-optimal solutions** in the presence of spatial constraints imposed on a robot while possessing **high planning efficiency**[2]. Similar to the approach presented in [135], I decompose the RTSP into two stages: the configuration assignment stage and the task sequencing stage. This method, henceforth referred to as the **Cluster-RTSP** algorithm, applies a recursive configuration reduction strategy based on the concept of *population similarity* to reduce the number of candidate configurations for each task point from the set of IK solutions to a single *best-fit* configuration. This individual configuration is subsequently assigned to the corresponding task point. The X-means algorithm [170] is used to partition the set of assigned configurations into clusters, allowing the RTSP to be formulated as a Clustered TSP (CTSP). Effectively, this clustering procedure divides the task sequencing stage into multiple TSP sub-problems, which are individually solved and subsequently combined to form the complete solution. In this way a significant reduction in computation time can be achieved for problems involving many task points. Using the

[2]The method has been evaluated on tasks of up to 1500 points and in all cases a solution was obtained in less than 2 minutes

RoboTSP as a benchmark, I show that the Cluster-RTSP is capable of finding solutions of a higher quality for spatially-constrained applications, while in uncluttered environments its performance is comparable to existing solutions. In both cases the Cluster-RTSP outperforms existing methods in terms of planning speed. Note that in this work I evaluate the quality of a solution according to the *task execution time*, which is defined as the time required for the robot to complete the task of visiting all task points from the start of execution based on the planned task sequence. This is different from the *computation time*, which is the time required by an algorithm to compute a solution (i.e. the planning time) and is used to evaluated an algorithm's planning efficiency.

Throughout this chapter I will analyse the algorithm with respect to optimality and complexity. Additionally, I present the results of a case study involving the deployment of the Cluster-RTSP algorithm for a mock surface inspection task consisting of a number of pipes arranged within a cell. Using the KUKA AGILUS KR6 R900 sixx robot, I discuss a number of important considerations for real-world implementation of the algorithm.

6.2 Cluster-RTSP Algorithm

Recall from the literature review presented in Section 2.2.3 that while existing approaches have, by and large, been able to compute high quality solutions to RTSPs in mostly uncluttered environments, challenges still remain in balancing the effort of optimising a task sequence that takes into account the spatial constraints imposed by obstacles and solving the problem within practical time. It is crucial that these challenges are addressed in order to enable the robotic capabilities required to meet the demands of more challenging applications and the increasing need for greater flexibility in modern robotics. From a low-level perspective, a key limitation exist in tackling the sequencing component of RTSPs using distance metrics defined in the task space: these metrics do not take into consideration the kinematic properties of the robot nor the spatial constraints imposed by obstacles into account. Consequently, features such as the reachability of target points and the possibility of encountering singularities have no influence on the solution provided by task space metrics. This can eventually manifest into unsmooth trajectories or low quality solutions, especially in cluttered environments.

In this section I present the Cluster-RTSP, a fast algorithm developed to

overcome the limitations of existing methods for solving spatially-constrained robotic task sequencing problems. The algorithm achieves *near*-optimal solutions to RTSPs by first assigning a good candidate configuration (according to a similarity metric relative to the global *population* of IK solutions) to each task point. This approach to configuration assignment avoids the explicit computation of all motion costs (thus reducing the computational resources consumed by the algorithm), while providing an effective way to estimate the best configuration for each task point. The sequencing sub-problem is then formulated as a CTSP to further reduce the computation time required to compute a low-cost task sequence for large problems. Importantly, the CTSP is formulated and solved in C-space to overcome the limitations of solving RTSP problems in task space.

6.2.1 Problem Formulation

Let us formally define the RTSP in the following way. Given a task space T and a set of n task points $P_n = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n)$, $\mathbf{p}_i \in T$, specified in the task space, a feasible *task sequence* is defined as $S_T = (\mathbf{p}_0, \mathbf{p}_{\{1\}}, \mathbf{p}_{\{2\}}, \ldots, \mathbf{p}_{\{n\}}, \mathbf{p}_0)$, such that the robot visits every task point in P_n exactly once, starting from (and returning to) the home position $\mathbf{p}_0$. Given a robot C-space C, the task sequence may alternatively be represented by a *configuration sequence* $S_C = (\mathbf{q}_0, \mathbf{q}_{\{1\}}, \mathbf{q}_{\{2\}}, \ldots, \mathbf{q}_{\{n\}}, \mathbf{q}_0)$, where each configuration $\mathbf{q}_{\{i\}} \in C$ uniquely reaches one task point in P_n, and $\mathbf{q}_0$ represents the home configuration.

Now, recalling the definition of an optimal path in Chapter 4, let us define $\sigma^*_{i,j}$ as the optimal collision-free trajectory to move from configuration $\mathbf{q}_i$ to $\mathbf{q}_j$, and $c_p(\sigma^*_{i,j})$ as the cost of this motion. Letting $\Omega = (\sigma^*_{0,\{1\}}, \sigma^*_{\{1\},\{2\}}, \ldots, \sigma^*_{\{n\},0})$ denote the set of optimal trajectories that enable the sequence of configurations S_C to be visited contiguously, the objective of the RTSP is to determine the optimal task sequence S^*_C such that the accumulated cost of Ω is minimised.

Generally speaking, the primary objective of an industrial robotic sequencing task is to maximise throughput, which corresponds to minimising the duration of a task. Thus in this work I use the task execution time, computed as the sum of the individual duration of each trajectory, to evaluate the quality of solutions. This is also convenient for benchmarking purposes as the majority of existing methods in literature also report the task execution time as the primary performance indicator.

For each task point $\mathbf{p}_i \in P_n$, there exists a set of IK solutions, $Q_i \subset C$ that can reach $\mathbf{p}_i$, which is obtained by solving the inverse kinematics problem (as described in Chapter 3). When spatial constraints are imposed on the robot, some of these configurations may become invalid due to collision. Task points can be identified as unreachable when no collision-free IK solutions exist (this can occur either due to obstruction or when tasks points simply lie beyond the robot's work envelope). Let us denote Q' as the set of assigned configurations for P_n such that only one valid IK solution exists in Q' for each task point in P_n. Solving an RTSP therefore comprises of finding an optimal configuration assignment $\mathbf{q}_i^* \in Q_i$ for each task point $\mathbf{p}_i$.

By the definitions given above, the problem of task sequencing and motion planning is highly coupled: to compute the optimal trajectories between task points, we require the selection of configurations for each task point and their visiting order, but on the other hand, to solve for a configuration sequence we require the motion costs between configurations, whose true values can only be obtained through motion planning. As shown in previous literature, attempting to solve the RTSP by brute force (including the computation of all motion costs) becomes highly impractical even for a relatively small set of n task points.

In the method presented herein, I decompose the RTSP into two smaller sub-problems, comprising of: **configuration assignment** and **configuration sequencing**.

6.2.2 Algorithm Description

With reference to the pseudo-code provided in Algorithm 4, the Cluster-RTSP applies the following steps to solve an RTSP:

1. For all task points $\mathbf{p}_i \in P_n$, the set of collision-free **IK solutions** Q_i is computed. In my implementation, the *IKFast* kinematics solver, a module in the *OpenRAVE* environment [171], was used to achieve this. (See Lines 2-4)

2. A **configuration assignment** procedure estimates the optimal configuration $\mathbf{q}_i^*$ for each $\mathbf{p}_i$ using a similarity heuristic to recursively evaluate the similarity of every configuration in Q_i to the population of all IK solutions in Q, and reducing the number of candidate configurations in Q_i until it

Algorithm 4 Cluster-RTSP

Input: Set of n target points P_n and home configuration $\mathbf{q}_0$
Output: Ordered configuration sequence S and corresponding set of trajectories Ω

1: $S \leftarrow \mathbf{q}_0, Q \leftarrow \mathbf{q}_0$
2: **for all** $\mathbf{p}_i \in P_n$ **do**
3: $\quad Q.append(getAllIKSolutions(\mathbf{p}_i))$
4: **end for**
5: $Q' \leftarrow configurationAssignment(Q)$
6: $clusters \leftarrow configurationClustering(Q')$
7: $D \leftarrow computeDistanceMatrix(clusters, \mathbf{q}_0)$
8: $gtour \leftarrow globalTSP(clusters, D)$
9: **for all** $idx \in gtour$ **do**
10: $\quad cluster \leftarrow clusters[idx]$
11: $\quad entry, exit \leftarrow getEntryExitPoints(cluster)$
12: $\quad tour \leftarrow localTSP(cluster, entry, exit)$
13: $\quad S.append(cluster[tour])$
14: **end for**
15: $S.append(\mathbf{q}_0)$
16: **for all** $\mathbf{q}_i \in S$ **do**
17: $\quad trajectory \leftarrow planTrajectory(\mathbf{q}_i, \mathbf{q}_{i-1})$
18: $\quad \Omega.append(trajectory)$
19: **end for**

converges to one configuration. This generates the set of optimised[3] configurations Q', which lie in close proximity in the C-space. (See Line 5)

3. The set of assigned configurations Q' is partitioned into **configuration clusters** using the clustering algorithm *X-means* [170], which groups configurations according to their distribution in C-space. (See Line 6)

4. A **configuration sequence** is computed by applying the CTSP formulation to the configuration clusters. Using the 2-Opt algorithm [133], an *inter-cluster* visiting order is first obtained by solving a high-level TSP comprising of the set of clusters as the cities to be visited. Each cluster is then used to form a low-level TSP, which is individually solved to obtain the local configuration sequence within the cluster (i.e. the *intra-cluster* configuration sequence). Finally, the solutions are aggregated to obtain the

[3]Note the subtle difference between the terms *optimized* and *optimal*. Here the heuristics enable the assignment of good configurations, but there is no guarantee that this assignment provides the global optimal solution to the RTSP. See Section 6.2.4 for more details.

complete configuration sequence S_C^*.[4] (See Lines 7-14)

5. **Motion planning** queries are called for each pose-to-pose motion along the solution S_C^*. These planning queries can be fulfilled by any motion planner applicable to robotic manipulators. In my implementation, the bi-directional RRT [50] was used in conjunction with time parameterization, both available within OpenRAVE, to obtain collision-free trajectories. These trajectories satisfied both the joint limits and the maximum velocity and acceleration limits of the robot. (See Lines 16-18)

Fig. 6.1 summarises the software architecture of the Cluster-RTSP algorithm. I will empirically show later in Section 6.3 that by performing both configuration assignment and configuration sequencing in C-space, the Cluster-RTSP is capable of providing better solutions to sequencing problems involving **spatial constraints**. Furthermore, by adopting clustering techniques to transform the problem into a CTSP, the computation time required for sequencing **large sets of task points** can be drastically reduced in comparison to existing methods. Importantly, to the best of my knowledge no previous work in RTSP literature have considered these types of sequencing problems collectively.

6.2.2.1 Configuration Assignment

To achieve a near-optimal assignment of configurations to the set of task points, the algorithm computes a *similarity measure* for each configuration in the set of all valid configurations Q. All configurations that are IK solutions to the same task point are compared according to these similarity values, and the m configurations with the worst similarity to the global set Q are removed from Q. That is, they are no longer considered as candidate configurations for the corresponding task point. This is performed recursively until the algorithm converges to a single configuration $\mathbf{q}_i^*$ for each task point.

The similarity heuristic is computed by first defining a dissimilarity function δ between two configurations $\mathbf{q}$ and $\mathbf{q}'$, which is given by the weighted squared Euclidean distance in C-space:

[4]With a slight abuse of notation, I use the $*$ notation here to indicate that the configuration sequence obtained by the Cluster-RTSP is an *estimate* of the optimal solution, rather than the *true optimal*.

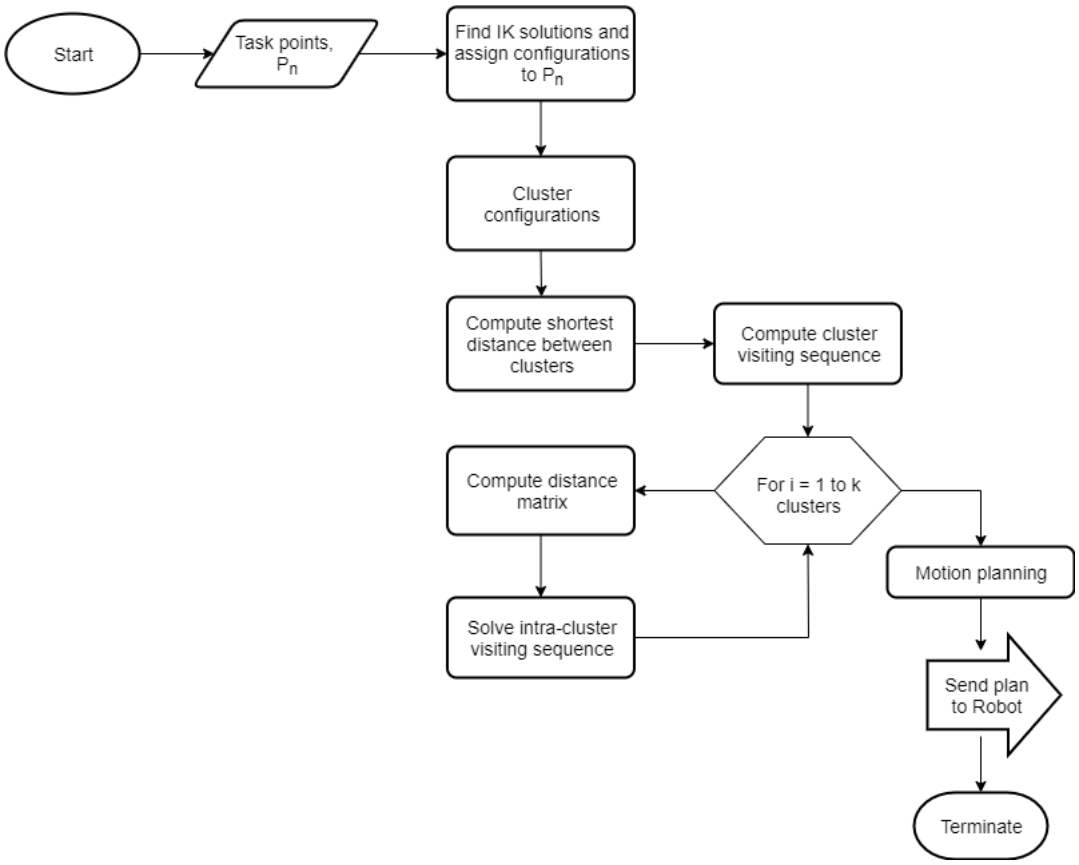


Figure 6.1: Software architecture for the standard Cluster-RTSP algorithm.

$$\delta(\mathbf{q}, \mathbf{q}') = \sum_{j=1}^{DoF} w_j (q'_{[j]} - q_{[j]})^2 \tag{6.1}$$

Here w_j is a positive weight for j^{th} joint, and $q_{[j]}$ refers to the axial angle of joint j for configuration $\mathbf{q}$. The values for the weight of each joint is fixed for a given robot and can be obtained by evaluating the maximum displacement of any point on the robot when actuated about the corresponding joint [135]. Note that it is entirely possible to use alternative metrics, though this would lead to somewhat different solutions to the configuration assignment problem. The study conducted in [172] investigated the effects of applying different metrics to the behaviour of a 7-DoF robot. It found that some metrics performed better for expansion tasks, while other metrics such as the Euclidean distance were more suited for contraction tasks. These findings indicated that no metric is universally more effective across all types of tasks. In this work, the Cluster-RTSP is benchmarked in several different environments. In order to main consistency

across all of these trials, I chose to apply the weighted squared Euclidean metric in all instances. In practice, it could be beneficial to evaluate the effectiveness of different metrics for the given application.

Using Eq. 6.1, the similarity heuristic ϕ for any configuration $\mathbf{q} \in Q$ can be defined as:

$$\phi(\mathbf{q}) = b_\delta \cdot \bar{\delta}(\mathbf{q}) + b_0 \cdot \delta(\mathbf{q}, \mathbf{q}_0) \tag{6.2}$$

where b_δ and b_0 are bias terms such that $b_\delta + b_0 = 1$, and $\bar{\delta}(\mathbf{q})$ is the mean dissimilarity between $\mathbf{q}$ and every other configuration in the set Q and is given by Eq. 6.3.

$$\bar{\delta}(\mathbf{q}) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \delta(\mathbf{q}, \mathbf{q}_i) \tag{6.3}$$

In words, Eq. 6.2 describes the similarity of configuration $\mathbf{q}$ to every other candidate configuration, with an additional bias term that penalises those configurations that do not lie in close proximity to the home configuration. Since ϕ is derived from the dissimilarity function, a configuration with a larger ϕ value is considered more dissimilar to the rest of the configurations considered in the configuration assignment problem. Thus, to converge towards $\mathbf{q}^*$, we seek the configuration with the lowest ϕ value.

As mentioned earlier, the algorithm uses a recursive procedure to converge towards a single configuration per task point. Indeed it is possible to evaluate the similarity heuristic only once for all configurations and simply assign the best configuration from among each set of IK solutions $Q_i \subset Q$ for task point $\mathbf{p}_i$. However, doing so means that the poor candidate configurations currently in Q would contribute to the values of ϕ for all other configurations, resulting in somewhat deceptive heuristic values. This makes it difficult to discern the best configuration from among multiple good configurations. To filter out these effects, a recursive process was applied to eliminate m_i configurations with the largest ϕ values from each subset Q_i according to Eq. 6.4:

$$m_i = \max\left(1, \lfloor \ln |Q_i| \rceil\right) \tag{6.4}$$

Here the value of m_i is chosen as the maximum between 1 and the natural logarithm of the cardinality of Q_i (rounded to the nearest whole number). Once

these configurations have been removed from Q, the values of ϕ for all remaining configurations are updated. By iterating this process, the algorithm converges logarithmically towards $\mathbf{q}_i^*$ for each task point $\mathbf{p}_i$ (that is, $|Q_i| = 1 \mid_{1 \leq i \leq n}$).

Finally, one modification was made to the described procedures above to improve the computational performance of the algorithm. In Eq. 6.2, computing the mean dissimilarity of a configuration against all other configurations in q can be expensive for problems involving anywhere from several thousand configurations onwards (for an example, Table 6.2 reports the number of configurations considered in the trials conducted for benchmarking). This can significantly limit the efficiency of the algorithm for large sets of task points. To resolve this, a preliminary instance of clustering is performed on the set Q at the beginning of each iteration to partition configurations into k clusters. This is achieved using the X-means algorithm[5], an extension of the well-known *k-means* algorithm [173].[6] The outputs of the algorithm are the set of cluster centroids X, the number of configurations in each cluster, stored within the set R, and the membership information of each configuration describing which cluster they belong to.

Letting $\mathbf{x}_c \in X$ denote the centroid of cluster c (computed as the mean of each cluster) and $r_c \in R$ denote the number of configurations in cluster c, I modify Eq. 6.3 to compute the **weighted** mean dissimilarity, $\bar{\delta_w}(\mathbf{q})$, given as:

$$\bar{\delta_w}(\mathbf{q}) = \sum_{c=1}^{k} \frac{\delta(\mathbf{q}, \mathbf{x}_c) \cdot r_c}{r_n} \tag{6.5}$$

Where r_n denotes the total number of configurations across all clusters. Now, instead of computing the mean dissimilarity between $\mathbf{q}$ and every other configuration in Q, only the centroids of each cluster is considered for the calculation. This drastically reduces the number of δ computations from possibly several thousand and upwards to just k times for each ϕ computation.

Algorithm 5 gives a summary of the procedures used to perform configuration

[5]We will explore how this algorithm works in Section 6.2.2.2, where a detailed description of the algorithm is provided in the context of configuration clustering for Step 3 of the Cluster-RTSP algorithm.

[6]The k-means algorithm clusters a dataset into a predefined number of k clusters. While k-means is a simple and effective algorithm for partitioning data points into clusters, it is generally difficult to determine an appropriate value for k. The extension provided by X-means addresses this by enabling the algorithm to objectively determine the optimal number of clusters according to a model fitness evaluation criterion. This results in a clustering that partitions the dataset into identifiable groups without overfitting (e.g. creating a cluster for each data point in the extreme case).

Algorithm 5 configurationAssignment

Input: Population of all valid IK solutions Q
Output: Selected configurations Q'

while $|Q_i| \neq 1, \forall Q_i \subset Q$ **do**
  $X, R \leftarrow Xmeans(Q)$
  for all $Q_i \subset Q$ **do**
    $\Phi \leftarrow similarityMeasure(Q_i, X, R)$
    $Q_i \leftarrow sort(Q_i, \Phi)$
    $m_i \leftarrow computeReductionSize(Q_i)$
    $Q \leftarrow trim(Q_i, m_i)$
  end for
end while

assignment as pseudo-code.

6.2.2.2 Configuration Clustering

Given the set of assigned configurations Q', a configuration sequence can be obtained by formulating the problem as a standard TSP, solved using C-space distance metrics. The subsequent TSP sub-problem could then be solved using the 2-Opt algorithm for its notable efficiency. However, as noted in various works [135, 174, 175], even for this algorithm the complexity of solving a TSP is exponential in practice. Thus it does not scale well for particularly large sets of task points. To overcome this, a clustering algorithm is applied to Q' to partition configurations into clusters. By doing so, the original sequencing problem can be divided into several smaller sequencing sub-problems, which is simpler to solve than the original problem.

A fundamental consideration for data clustering in general is the choice of an appropriate value for the number of clusters k. In RTSPs, it can be difficult to determine an optimal k for partitioning Q' as this is generally not known a priori and varies according to the nature of the particular task. To deal with this, the X-means algorithm is applied to first compute k and then perform the actual configuration clustering procedure. X-means is an extension of the k-means algorithm, where both adopt a spherical Gaussian model assumption (the algorithm partitions data points into spherical clusters). In k-means, data points are recursively assigned to the nearest cluster among k centroids (initialised randomly) according to a squared Euclidean distance metric. After the membership of data points are determined, the location of centroids are updated as the mean of all

points belonging to that cluster. These steps are repeated until the locations of the centroids stabilise. The obvious drawback to the k-means algorithm is the requirement for k to be defined explicitly as an input.

X-means eliminates this shortcoming by objectively computing an optimal k value that lies between the given limits $[K_{min}, K_{max}]$. The algorithm first performs an initial clustering using the same procedures as k-means with $k = K_{min}$. A model selection criterion is used to evaluate the quality of fit for this value of k. The algorithm then loops over each cluster (referred to as the *parent* cluster) and bisects the members into two *children* clusters. These children clusters are collectively evaluated for fitness using the same model selection criterion and their score is compared against the parent cluster. Under the condition that the children clusters score better than the parent cluster (i.e. a better fit of these local points is achieved), k is incremented by one. These steps are repeated until no further bisections are accepted (due to poorer fit) or when k reaches K_{max}. A fresh instance of the k-means algorithm is applied to the original data set using the resulting k value to obtain the final set of configuration clusters.

The model selection criterion plays an important role as it must adequately discern whether the addition of a cluster would lead to overfitting the input set of points. In the Cluster-RTSP, the model selection criterion is based on the Bayesian Information Criterion (BIC) [176] as used by the original authors of the X-means algorithm [170]. The BIC is based on the likelihood function and incorporates additional terms to penalize the number of parameters in a model to limit the case of overfitting. When applied to configuration clustering, parameters refer to the number of clusters k and the dimensionality of the configurations, d (i.e. the number of DoFs). By exploiting the spherical Gaussian assumption in the context of clustering, the BIC is given by:

$$BIC(M_a) = \hat{l_a}(Q') - \frac{h_a}{2} \cdot \log r_n \tag{6.6}$$

Here M_a represents the a^{th} model, h_a denotes the number of parameters in model a, r_n is the cardinality of Q' and $\hat{l_a}(Q')$ is the log-likelihood function of the set Q'. Let us define $\hat{\sigma}^2$ as the maximum likelihood estimate for variance (under the spherical Gaussian assumption):

$$\hat{\sigma}^2 = \frac{1}{r_n - k} \sum_{i=1}^{r_n} (\mathbf{q}_i - \mathbf{x}_i) \tag{6.7}$$

Where x_i is the centroid associated to the i^{th} configuration. Letting Q_c denote the set of points in cluster c and $r_c = |Q_c|$, the log-likelihood of a cluster is obtained by:

$$\hat{l}(Q_c) = -\frac{r_c}{2}\log(2\pi) - \frac{r_c \cdot d}{2}\log(\hat{\sigma}^2) - \frac{r_c - k}{2} + r_c \log r_c - r_c \log r_n \tag{6.8}$$

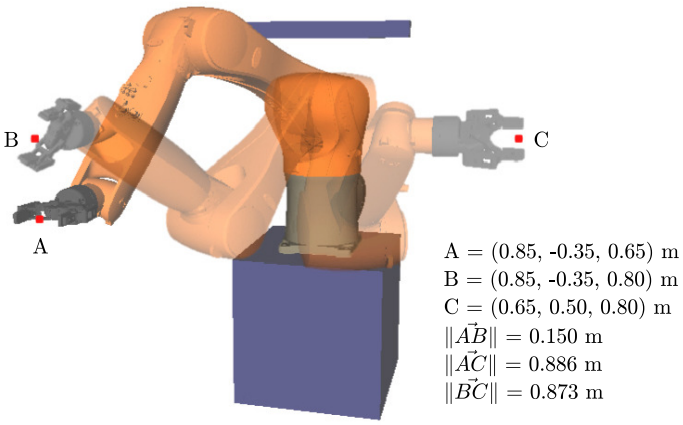
Suppose the parent cluster of the configurations Q_c is represented by model A and the corresponding two children clusters that spawned from the bisection of the Q_c is represented by model B. Model B is accepted as a better fit to Q_c if the following inequality holds:

$$BIC(M_B) > BIC(M_A) \tag{6.9}$$

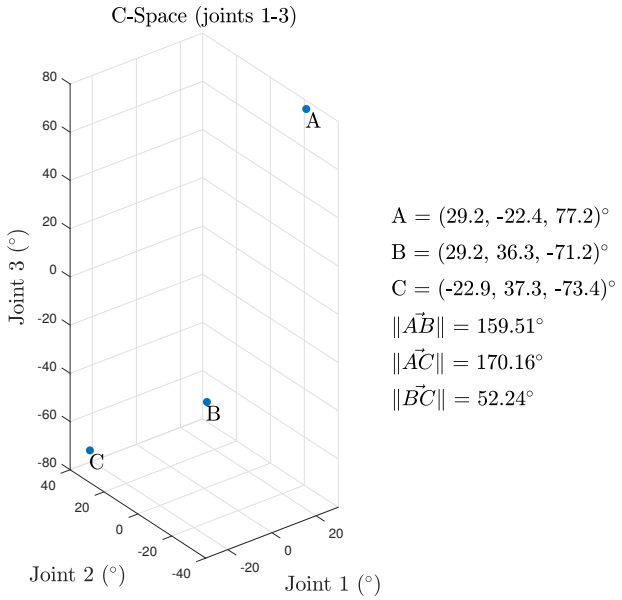
While I have not covered the derivations of the BIC formulation, interested readers are directed to [177], which provides a concise derivation of Eqs. 6.6-6.8.

Finally, I wish to emphasise that the squared Euclidean distance metric used for evaluating the distances between configurations and cluster centroids is applied in the C-space, as opposed to its more common use in task space. When applied in the task space, the X-means algorithm would cluster configurations according to how close the corresponding task points lie in the 3-dimensional world. Conversely, by applying the metric in C-space, the resulting partitions take into consideration the actual distances between configurations, which better reflects the actual motions required to reach each task point.

To help illustrate this, Fig. 6.2 gives an example of the distance values between three task points (and the corresponding IK configurations) obtained in task space (Fig. 6.2a) and in C-space (Fig. 6.2b). In this example, task points A and B lie close together in the task space, while task point C lies comparably further away. However, to actually reach these task points, the robot requires a drastically different configuration for A and B, but C can be reached using a configuration that more closely resembles configuration B. Thus when applying the Euclidean distance metric, very different inferences would be obtained depending on which space is considered. Since this work is concerned with finding the shortest duration sequence for an RTSP, it is intuitive to apply clustering in the C-space, which describes the robot state transitions that directly correlate to the robot's trajectories.



(a)



(b)

Figure 6.2: Example of differences in task space and C-space metrics for three given task points A, B and C, (a) distances between A, B and C given by Euclidean metric in task space (b) distances between configurations for A, B and C given by Euclidean metric in C-space (only first three joints are considered for visualisation).

6.2.2.3 Clustered Travelling Salesman Problem

Using the clustering procedures described in Section 6.2.2.2, the configuration sequencing problem is reformulated as a CTSP, first introduced by Chisman in 1975 [178]. CTSPs can be expressed in various forms [179], but we will restrict

our attention to the free CTSP. The free CTSP describes problems in which no restrictions are placed on the visiting order of the clusters (i.e. they may be visited in any order). The only requirement is that all points within a cluster should be visited contiguously before advancing to the next cluster, otherwise the problem would revert to a standard TSP. Free CTSPs also assume that any point in each cluster can be used as the entry or exit point when moving between clusters, though, like the standard TSP, any point should only be visited once.

In this work, a solution to the CTSP is obtained by solving multiple smaller TSP sub-problems. Following the benchmarking results on TSP solvers presented in [135], the 2-Opt algorithm [133] was selected to solve these instances of TSPs thanks to its planning efficiency and near-optimal performance. The first TSP deals with the ordering of the numerous clusters, each considered as a single entity, to obtain an inter-cluster visiting sequencing. In order to apply the 2-Opt algorithm, a pairwise distance matrix D containing the distance between each element considered in the TSP is required. This is generated by finding the closest points between each pair of clusters according to the C-space Euclidean distance metric. Additionally, two dummy clusters containing $\mathbf{q}_0$ are added to the set of clusters for sequencing. These are fixed as the start and end clusters within the 2-Opt algorithm such that the solution takes into consideration the cost of moving from the home position to the first cluster and returning to the home configuration after the last cluster.

Once the inter-cluster visiting sequence is obtained, the required entry and exit points can be determined simply from the closest points for the cluster pairs along the sequence that were found when generating D. Each cluster (excluding the dummy clusters added earlier) are used to formulate an individual TSP sub-problem at the intra-cluster level. By defining the corresponding entry and exit points in the cluster as fixed start and end points, a local configuration sequence can be determined by again applying the 2-Opt algorithm. Finally, by aggregating the inter-cluster visiting order with the local configuration sequences found at the intra-cluster level, a complete configuration sequence S can be obtained. This now provides sufficient information to compute the set of collision-free trajectories by performing a motion planning query for each successive configuration in S.

6.2.3 Complexity Analysis

An analysis of the complexity of the Cluster-RTSP is provided below, where I consider each component of the algorithm in turn.

In Step 1 of the algorithm where IK solutions for each task point is obtained, suppose λ is the upper-bound on the number of valid IK solutions for any task point $\mathbf{p}$. Then, for n task points, the complexity of computing IK solutions for the problem set is $\mathcal{O}(\lambda n)$.

In configuration clustering (used in steps 2 and 3), it is well known that using k-means to obtain an optimal clustering for a set of input points is an NP-hard problem [180].[7] In practical implementations of the algorithm, however, k-means is generally run for a defined number of iterations, t, where each run is initialised with randomly generated cluster centroids. Consequently, the true complexity of the k-means algorithm in practice is given by $\mathcal{O}(tkdn)$, where k denotes the number of clusters and d is the dimensionality of the problem.

For the configuration assignment procedure in Step 2 of the algorithm, let us define K as the maximum number of clusters obtained by the X-means algorithm. Using the weighted mean dissimilarity $\hat{\delta}_w$ to compute ϕ gives a complexity of $\mathcal{O}(Kn)$ for a single iteration. Recall that the procedure uses a recursive process to converge logarithmically to Q'. Thus the overall complexity for configuration assignment is $\mathcal{O}(Kn \log n)$.

Compared to the other components of the algorithm, the theoretical proof of the time complexity of the 2-opt algorithm is far more complex. Numerous authors have studied the time complexity of the algorithm for problem instances involving various types of point distributions [174, 175, 181]. I note in particular that in the analysis conducted by the authors in [174], an upper bound of $\mathcal{O}(n^{4+\frac{1}{3}})$ was derived for Euclidean instances involving uniform distributions with dimension $d \geq 2$. Nevertheless, several authors have reported a notable gap between theoretical analysis and experimental observations. Broadly speaking, finding a solution[8] to the TSP using the 2-Opt algorithm has an exponential complexity in practice. In the CTSP formulation involving multiple calls to the 2-Opt al-

[7] The quality of clustering produced by k-means varies according to the location of the initial cluster centroids. Since the implementation of the k-means generally involve randomly generating these initial centroids, a single instance of k-means is unable to guarantee an optimal solution.

[8] The solutions obtained by 2-Opt does not guarantee convergence as it is subject to local minima issues.

gorithm, a theoretical upper bound on complexity based on the analysis in [174] can be approximated as $\mathcal{O}(km^{4+\frac{1}{3}})$, where $m \ll n$. To put into perspective this difference in complexity, let us consider the case where the n points are equally distributed across k clusters. In this scenario, $m = \frac{n}{k}$, and the theoretical upper bound on the complexity of the CTSP becomes $\mathcal{O}(\frac{1}{k^3}n^{4+\frac{1}{3}})$. This means that even with only two clusters (i.e. $k = 2$), the upper bound on complexity falls by a factor of 8, while for three clusters the complexity becomes just $1/27^{th}$ of the original problem. While in practice points tend not to be equally distributed across clusters as in the given example, the actual reduction in complexity is usually not far off from this analysis.

The final step of the algorithm consisting of the computation of the complete set of trajectories has a linear complexity $\mathcal{O}(n)$, where a motion planning query is performed for each task point to be visited. However, the actual planning time required for a motion planning query itself is difficult to quantify as it varies drastically according to the dimensionality of the problem, the nature of the planning domain and the spatial constraints involved. In Section 6.4 I show empirically that for problems where the robot is subject to substantial spatial constraints, the computational resources consumed by the Cluster-RTSP algorithm is dominated by motion planning queries.

6.2.4 Optimality

Throughout this chapter I describe the Cluster-RTSP as a *near*-optimal algorithm. Thus in this section I make a key distinction between the terms *near*-optimal versus the *true* optimal. I refer to a *true*-optimal algorithm as one that possesses a guarantee to find the global optimal solution. To my knowledge, no existing methods in literature so far has been able to provide this guarantee for general RTSPs. For clarity, let us briefly review the properties of some of the most common existing methods to solving RTSPs.

Consider the iterative GLKH solver, which formulates and solves the RTSP as a GTSP. The algorithm seeks increasingly higher quality solutions by performing a search over a large number of iterations, retaining the best solution found, but it does not provide any guarantee that the search would converge to the true optimal solution. The RoboTSP, an algorithm used to benchmark the Cluster-RTSP in Sections 6.3 and 6.4, is able to assign configurations for a **given** sequence

such that the cost of the complete set of trajectories is minimised, but there is no guarantee that the **actual sequencing** of task points leads to the global optimal solution. The GA, which has been applied to solve RTSPs in a number of works, is well-known for its effectiveness in finding high quality solutions to many general optimisation problems. However, the algorithm only heuristically improves the likelihood of finding the true optimal and does not provide any means to evaluate the closeness of a solution to the global optimum (it is not uncommon for a GA to be stuck in local minima).

Ultimately, the absence of a means to determine the true optimal solution to an RTSP makes it difficult to quantify the quality of solutions obtained by a planner. While theoretically speaking an exhaustive search could provide the true optimal to an RTSP, it involves computing the costs of true trajectories to move between all possible pairs of configurations among the global set of IK solutions. Coupling this with the already complex problem of finding global solutions to TSPs [182] make a brute force approach intractable for large problems in practice.

A common feature in the majority of RTSP planners is the use of approximated motion costs, obtained through simple **task space metrics**, to direct the search for a solution to the sequencing sub-problem. The term near-optimal was chosen to describe the Cluster-RTSP algorithm as it better approximates the true cost of motion through the use of **C-space metrics** to inform the configuration sequencing procedure. Previous work found this difficult as the complexity of the sequencing problem increased significantly as a result of kinematic redundancy, which introduced multiple configurations to consider per task point. GLKH-based methods were an exception to this as they model the sequencing problem as a GTSP, but, as previously reported, these methods are particularly computationally expensive. The Cluster-RTSP overcomes these challenges through the introduction of a similarity heuristic (also informed by C-space metrics) to evaluate the fitness of configurations for assignment to task points **prior** to solving the sequencing sub-problem. While some previous works have also considered the assignment of configurations prior to sequencing, these were generally made arbitrarily without deterministically evaluating the suitability of configurations for the task at hand.

By following this approach, the Cluster-RTSP is able to avoid the explicit computation of true motion costs that are irrelevant to the final solution. However, for this very reason, there is no guarantee that the estimated motion costs used

to inform the algorithm accurately represents the true motion costs for moving between pairs of configurations, which is a limitation of this algorithm.

Little attention has been given to the optimality of 2-Opt for solving the TSP sub-problems thus far. As briefly mentioned previously, 2-Opt is a comparatively efficient algorithm for solving TSPs compared to methods such as those based on neural networks [182–187]. However, the 2-Opt algorithm suffers from local minima, resulting in solutions that are inferior in quality to the aforementioned alternative methods. This is another limitation of the Cluster-RTSP that prevent the algorithm from finding the true optimal solution to an RTSP.[9]

While these limitations of the Cluster-RTSP has important implications when considered in relation to the true optimal, in Section 6.3 and 6.4 I show empirically that the algorithm outperforms existing methods both in terms of solution quality and planning efficiency for (i) problems involving large sets of task points, and (ii) problems involving substantial spatial constraints. I also show that in the case of uncluttered environments, the algorithm's performance is comparable to existing methods. Finally, I also provide a benchmark of the 2-Opt algorithm against the aforementioned alternative TSP solvers to quantify the trade-off between efficiency and solution quality at the level of the TSP sub-problem.

6.3 Benchmarking in Simulation

I now present a number of simulation-based experiments conducted to benchmark the various key aspects of the Cluster-RTSP algorithm. I first provide an evaluation of the 2-Opt algorithm used in conjunction with the CTSP formulation adopted in this work for solving standard TSPs (which are widely accepted as the standard for TSP benchmarks) and compare its performance against other effective methods reported in literature. I then investigate how the number of clusters k affects the performance of the proposed CTSP approach when applied to RTSPs. The Cluster-RTSP is then carefully benchmarked against RoboTSP to quantify the quality of solutions obtained and the planning efficiency of the algorithm. The RoboTSP algorithm was chosen to benchmark the Cluster-RTSP as it is a competitive algorithm shown to outperform other existing methods in terms

[9] Although in this work I solely consider the use of 2-Opt to solve individual instances of TSPs, practitioners may opt to use a different TSP solver according to application needs. The Cluster-RTSP has been developed with modularity in mind to allow for such modifications to be made.

of computation time while being able to provide solutions of comparable quality for the problems considered in [135]. To ensure a fair and consistent comparison, the first set of trials was conducted on the *Airbus Shopfloor Challenge* problem originally used to benchmark the RoboTSP. I then consider a series of test environments involving arrangements of obstacles to compare the two algorithms' performances when subject to increasingly complex spatial constraints.

In all of the reported experiments, planning was conducted on a system running on an Intel® Xeon® CPU E3-1270 v3 (3.50 GHz) with 32 GB RAM and an NVIDIA Quadro K2000 graphics card.

6.3.1 Benchmarking TSP Solvers

The approach used in this work to solve a sequencing problem using the 2-Opt algorithm in conjunction with a CTSP formulation can be generalised for standard symmetrical TSPs. Indeed it is necessary to quantify the behaviour of solving the configuration sequencing sub-problem in this way as it directly contributes to the overall performance of the Cluster-RTSP algorithm. A comparison was conducted to benchmark the CTSP-based 2-Opt method (**CTSP-2-Opt** for short) used in this work on a set of standardised problems available in TSPLIB [188]. The TSPLIB is an open-access library resource that provides a collection of problems along with the best solutions found to date to enable researchers to benchmark TSP solvers on standardised data sets.

For each of these problems, I compare the results obtained by CTSP-2-Opt with the solutions obtained by a number of alternative proven methods for TSPs. Building upon the results originally collated in [182], Table 6.1 reports the performance of the CTSP-2-Opt along with the solution quality achieved by the following TSP solvers:

- Kohonen Network Incorporating Explicit Statistics Global (KniesG) [183]
- Kohonen Network Incorporating Explicit Statistics Local (KniesL) [183]
- Simulated Annealing with 2-Opt improvement (SA) [184]
- Budinich's Self-Organizing Map (Budinich) [184]
- Expanded Self-Organizing Map (ESom) [185]

- Efficient Self-Organising Map technique (Setsp) [186]
- Kohonen's Cooperative Adaptive Network (CAN) [187]
- Recurrent Neural Network with "Winner Takes All" (RNN-WTA) [182]

In Table 6.1, the quality of solutions obtained are given as a percentage error that describes the deviation of the solution from the best known solution for the given problem. According to this table, the quality of the solutions achieved with CTSP-2-Opt is marginally worse than those obtained using other methods in all problems except for the *rat195* dataset. Quantitatively, CTSP-2-Opt provides solutions that lie within an average of 0-6% error **difference** compared to other methods, and is always within 3% error difference from the worst-case results among the alternative methods (the worst solution among the benchmark results for each dataset is emphasised in italics). Certainly, there is a much bigger gap between the solutions found by CTSP-2-Opt and the best solution among the compared methods. However, I wish to note that this always coincides with the CAN method, which consistently outperforms all other methods by a large margin. Arguably, this showcases the exceptional performance of the CAN method rather than discredit the performance of the CTSP-2-Opt method.

While from the outset it may appear that the CTSP-2-Opt approach is a strictly inferior method for solving TSPs, the results provided in Table 6.1 does not provide a full picture of each method's computational performance. This is because the quality of solutions does not reflect the *computational cost* required to compute the solution. While the CAN method excels in findings solutions very close to the optimal, the authors of the algorithm reported a computation time of 159.3 seconds for the *pcb442* dataset [187]. Granted, this result was obtained in 2003 using a Silicon Graphics O2 workstation. Scaling this to the processing power of the Intel® Xeon® CPU E3-1270 v3 (3.50 GHz) used in this work according to the Whetstone benchmark [189] gives an equivalent computation time of 23.5 seconds.[10] In contrast to this, the CTSP-2-Opt required approximately 8 seconds to solve the *pcb442* problem, while solutions to all other prob-

[10]The Whetstone benchmark is a general purpose benchmark for computer system performance, where a machine's speed is measured by the number of Millions of Whetstone Instructions Per Second (MWIPS). Using these figures, the estimated computation time of an algorithm for a given machine can be obtained by scaling its true computation time on another machine according to the ratio between the two machines' MWIPS ratings. According to [190] the performance of the Silicon Graphics O2 workstation is rated at 424 MWIPS.

Table 6.1: Comparison of methods for solving TSPLIB symmetric TSP problems

TSP problem		**Average error (%)**								**CTSP-2-Opt**	
dataset	*n*	*WRNN*	*KniesG*	*KniesL*	*SA*	*Budinich*	*Esom*	*Setsp*	*CAN*	*error (%)*	*CPU time (s)*
eil51	50	1.16	2.86	2.86	2.33	*3.1*	2.1	2.22	**0.94**	4.20	0.06
st70	70	*3.38*	2.33	1.51	2.14	1.7	2.09	1.6	**1.33**	4.72	0.08
pr107	107	*3.14*	0.42	0.73	1.54	1.32	1.48	0.41	**0.17**	3.17	0.13
pr152	152	*3.25*	1.29	0.97	2.64	2.04	0.89	1.17	**0.74**	3.57	0.66
rat195	195	7.19	11.92	12.24	*13.29*	11.48	7.13	11.19	**5.27**	10.71	0.39
kroa200	200	*10.6*	6.57	5.72	5.61	6.13	2.91	3.12	**0.92**	10.92	0.17
pcb442	442	*11.17*	10.45	11.07	9.15	8.43	7.43	10.16	**5.89**	14.80	8.32

Notes: The errors refer to the percentage deviation of solutions from the best known solution for each TSP problem. The best-performing result for each problem is highlighted in **bold**, while the worst performing solution among the compared methods (excluding CTSP-2-Opt) is emphasized in *italics*. The results of the 8 compared methods were obtained from [182]

lems were obtained in under 1 second. The authors of the RoboTSP also made similar observations in terms of the planning efficiency when they benchmarked their method for RTSP against a Recurrent Neural network (RNN)-based approach [135]. Going back to the requirements of modern applications that cannot be met by substantial offline planning (refer to Section 6.1), when considering the complete process cycle time it is generally necessary to consider not only the task execution time (quality of solution) but also the total computation time required to generate a plan. In these scenarios, being able to reduce the computation time by several fold significantly outweighs the marginal improvements achieved in task execution time. Thus when taking into account both solution quality and planning efficiency, the CTSP-2-Opt can be considered a superior method for applications were online planning capabilities is necessary.

6.3.2 Benchmarking CTSP

In Section 6.2.3, the theoretical upper bound on the complexity of the CTSP procedure had been derived as $\mathcal{O}(\frac{1}{k^3}n^{4+\frac{1}{3}})$ for the case where points are equally distributed across clusters. However, as previous analysis of the standard 2-Opt algorithm has found, practical observations showed an exponential growth in complexity for the number of points in the problem. To evaluate empirically the reduction in computational cost by applying a CTSP formulation to the configuration sequencing sub-problem, an experiment was conducted whereby the number of clusters k were manually assigned during the clustering phase of the algorithm (i.e. the algorithm reverts back to a regular k-means clustering).

The experiment was conducted on a virtual KUKA KR6 R900 sixx 6-DoF industrial robot and involved solving the planning tasks shown in Fig. 6.3, where Fig. 6.3a shows the distribution of task points for each of the four environments (though the actual number of points was varied across trials). Notice that each environment represents a different arrangement of obstacles. Environment A corresponds to a clutter-free environment where no spatial constraints are imposed on the robot. Environment B contains three planar obstacles that are difficult to detect without performing motion planning as they generally do not invalidate the majority of IK solutions for task points that lie close to the obstacles, but nevertheless greater motion costs would be incurred from moving between task points in the vicinity of these obstacles. Only one obstacle is present in

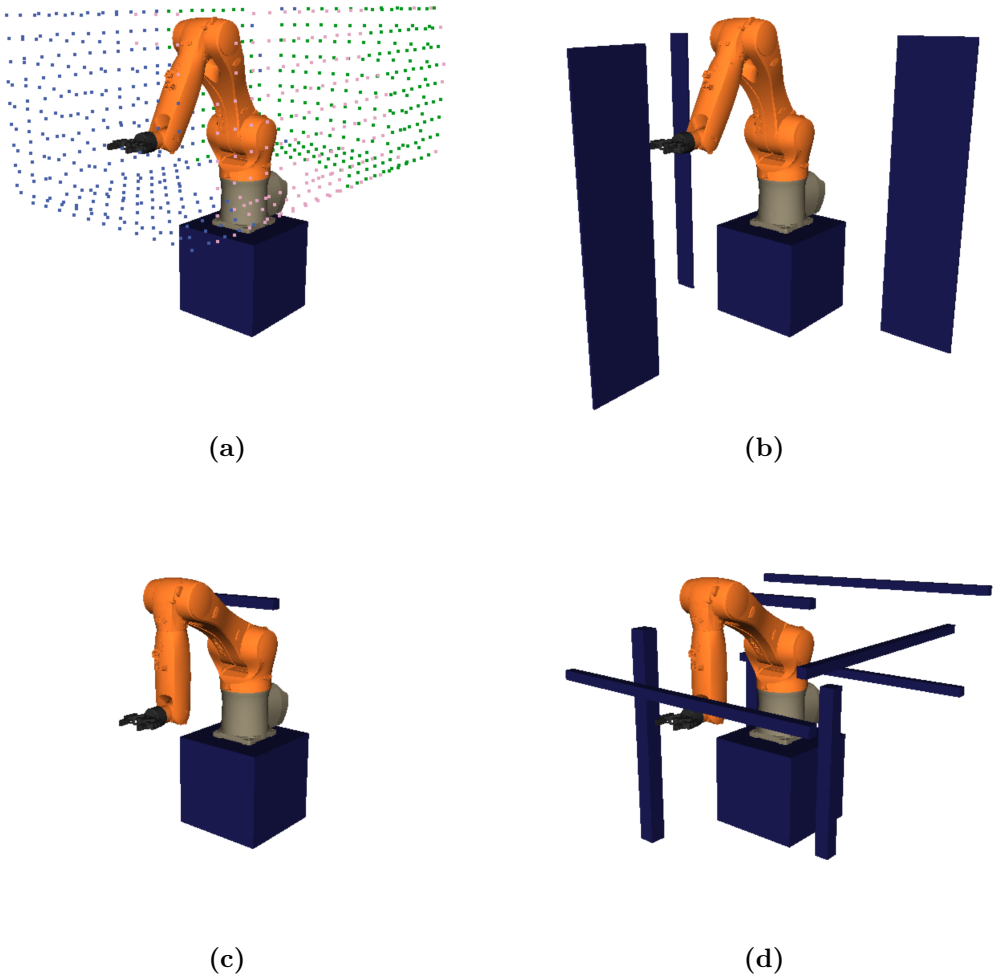


Figure 6.3: Environments generated for evaluating the performance of the Cluster-RTSP. (a) Environment A - clutter-free and shown with sample input target points, (b) Environment B - contains three planar obstacles, (c) Environment C - contains a single bar obstacle limiting motion of joint two, (d) Environment D - cluttered environment imposing significant spatial constraints on robot.

Environment C, though the location of this obstacle introduces significant spatial constraints on the robot as it lies very close to the robot's second joint. This prevents the robot from reaching the majority of task points in the upper half of the distribution using an 'elbow-up' configuration. Finally, Environment D involves significant clutteredness around the robot that further restricts the set of feasible motions available to the robot.

In numerous industrial applications such as inspection, thermal mapping and drilling, the yaw angle of the end effector (i.e. the rotation about the approach

vector) at each task point can be arbitrarily assigned. The authors of RoboTSP accounted for this by treating the DoF that coincides with the yaw rotation of the end effector (i.e. joint 6 for the KUKA KR6 R900 robot) as a *free DoF*. As a result, there could be infinitely many IK solutions for any given task point. To maintain consistency in the comparison against RoboTSP in later evaluations, I follow the authors' treatment of this free DoF in [135], which consists of discretising the free DoF to generate a finite set of IK solutions. In all of the experiments reported in this chapter, a discretisation level of $\frac{\pi}{3}$ was used unless otherwise stated. For the KUKA KR6 considered in this experiment (as well as others reported later) up to 96 IK solutions could be obtained for any single task point. Table 6.2 reports the number of task points considered across multiple trials, along with the corresponding total number of IK solutions Q found in each environment. Notice that as the free C-space reduces from more spatial constraints being introduced into the problem, the number of IK solutions decrease as a result of task points becoming unreachable due to collision.

Fig. 6.4 reports the computation time required to solve the planning problems in Environment A and D with k set to 1, 3, 6, 12 and 40 clusters. Here the trials for $k = 1$ is equivalent to solving the RTSP without formulating the configuration sequencing sub-problem as a CTSP. Since all points are considered in one single instance of TSP, the algorithm simply reverts to applying 2-Opt to the set of task points as in a standard TSP.

From these results it can be seen that for the trials that were solved without applying clustering, the growth in computation time is approximately exponential, as already observed in previous studies. From approximately 600 task points and upwards, there is a noticeably larger difference in computation time between the trials solve with 1 cluster compared to all other trials. In Environment A, as the number of clusters increases, the increase in computation time can be more closely approximately by a **linear** function. This is unsurprising, as the theoretical analysis has indicated that the complexity can be reduced by a factor of $\frac{1}{k^3}$, which significantly dampens the rate of exponential growth for large k values. In Environment D, the behaviour of the growth in complexity is not as well defined from visual inspection. Nevertheless, while the trials for 1 cluster still approximately indicate exponential growth, it is clear that even by setting k to the smallest value of 3 considered in these trials, a significant decrease in computation time can be achieved. Interestingly, we can observe that the computation

Table 6.2: Total number of valid IK solutions

Task points	Env. A	Env. B	Env. C	Env. D
224	11,206	10,830	8,324	4,954
328	16,452	15,832	12,206	8,118
561	28,462	27,230	21,056	13,632
763	38,648	37,154	28,840	18,087
919	46,650	44,944	34,246	21,624
1,402	70,993	68,141	52,360	32,668
1,844	94,128	90,588	69,619	44,145

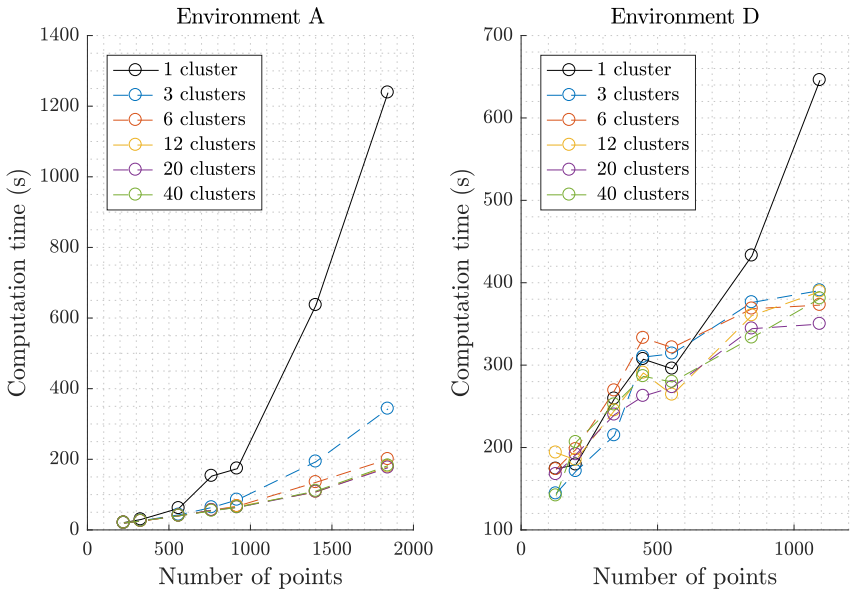


Figure 6.4: Computation time for Cluster-RTSP with fixed k values in configuration clustering.

time for 1 cluster is less at times for problems involving less than 600 task points. As discussed in Section 6.2.3, performing a motion planning query can be costly, particularly for cluttered environments. Since the sequencing solution obtained by applying different values of k can differ, the actual motion planning queries performed in each trial may be different. When problems involve smaller sets of task points, it is possible that the computation time required to perform motion planning outweighs any reduction in computation time achieved with clustering. In section 6.4 a breakdown of the computation time required by individual components of the algorithm is given (see Fig. 6.13).

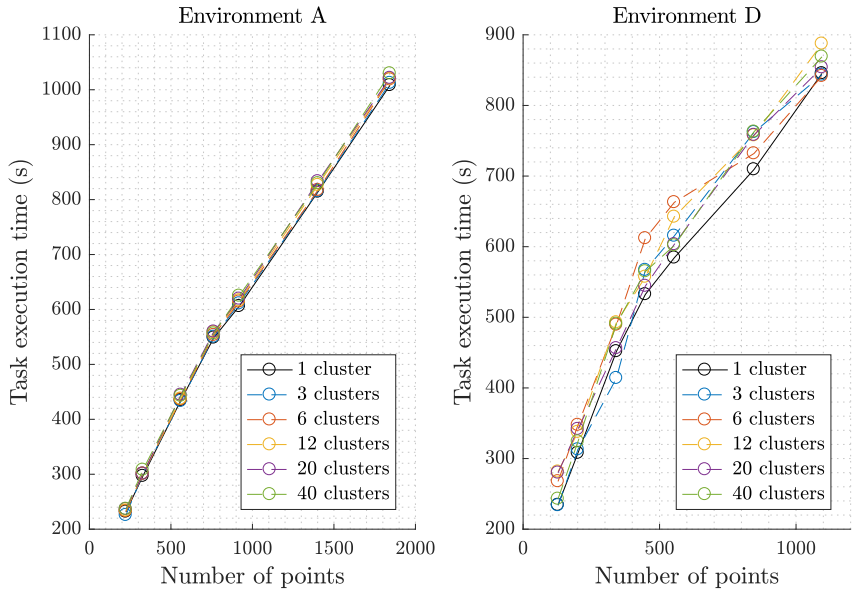


Figure 6.5: Task execution time for solutions obtained using Cluster-RTSP with fixed k values in configuration clustering.

Fig. 6.5 shows the task execution times of the solutions obtained in these trials. For the trials conducted in Environment A, solutions of comparable quality was obtained regardless of the number of clusters used to divide the sequencing problem. In contrast, the results for Environment D show a greater degree of variability among the solutions obtained by different k values. For all but one trials, the best solution was obtained using 1 cluster, suggesting that by applying clustering in highly complex environments generally lead to some deterioration in solution quality. Even among the trials that involved clustering, there is no clear indication of an ideal k value that provided a lower cost solution in all instances for Environment D. This suggests that the value assigned to k is an important consideration and indeed requires the evaluation of model fitness objectively select k for each individual problem. Ultimately I conclude that by adopting a CTSP formulation for RTSPs, the complexity of solving the sequencing sub-problem using the 2-Opt algorithm can be reduced to an approximately linear growth in practice. However, in likeness to the conclusions drawn from the benchmark on TSP solvers, doing so to improve planning efficiency comes at the cost of marginal decrease in solution quality.

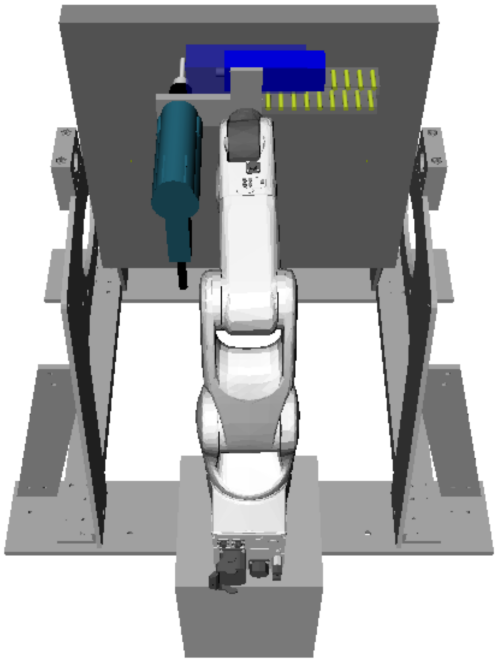


Figure 6.6: The Airbus Shopfloor Challenge environment originally used to benchmark RoboTSP [135]

6.3.3 Benchmarking on Airbus Shopfloor Challenge

To evaluate how the Cluster-RTSP performs for known RTSP problems, the algorithm was evaluated on the original planning problem used to benchmark RoboTSP. The development of the RoboTSP algorithm was motivated by the Airbus Shopfloor Challenge [191], a competition that involved the task of drilling a set of holes across a panel mounted on a jig. Fig. 6.6 shows a virtual representation of the environment developed by the authors of RoboTSP to participate in the challenge.

In this drilling task, the jig acts as an obstacle that somewhat limits the set of motions that can be performed by the robot. However, in this RTSP the task points are distributed across a single planar surface optimally located in front of the robot. In other words, the setup was carefully designed to minimise the risk of collision between the robot and the jig for the anticipated drilling task. For this reason, the spatial constraints present in this case is considered to be relatively relaxed.

The Cluster-RTSP was benchmarked against the RoboTSP by using the open-source implementation of the RoboTSP algorithm made available by the original

authors. As such, all simulations were conducted in OpenRAVE version 0.9.0 using a virtual model of the Denso VS060 6-DoF industrial robot. Care was taken to preserve the values of common parameters between the two alogirthms to maintain a fair comparison. These parameters include the stand-off distance (which describes the distance between a given task point and the actual point in task space to be reached by the robot), the motion planner used to fulfill motion planning queries, and the discretisation level. Table 6.3 lists all the parameters used by each algorithm to solve the Airbus Shopfloor Challenge problem.

For both algorithms, sets of nine trials comprising of 93, 130, 180, 245, 354, 432, 542, 627 and 843 task points were conducted for the following four free DoF discretisation levels: $\frac{\pi}{2}$, $\frac{\pi}{3}$, $\frac{\pi}{4}$ and $\frac{\pi}{6}$. Fig. 6.7 reports the task execution times for the solutions obtained in these trials. Generally speaking, the task execution times for the solutions obtained by Cluster-RTSP and by RoboTSP are by and large the same. These results indicate that the Cluster-RTSP provides solutions of comparable quality to existing methods for simple problems involving relatively few spatial constraints.[11]

I note however, that for the discretisation level of $\frac{\pi}{6}$, RoboTSP was able to find marginally better solutions than Cluster-RTSP, with an average of 4% shorter solutions. As discussed in Section 6.2.4, the Cluster-RTSP is a *near*-optimal algorithm, but possesses some limitations. Simply put, the configuration assignment procedure uses heuristics to *approximate* the best configuration for each task point. However, it does not guarantee that the assigned configuration is indeed truly optimal. With a larger discretisation level, it becomes more difficult to discern the best configuration from among several 'nearly as good' configurations. RoboTSP on the other hand first solves for a task sequence and then deterministically finds the optimal configuration assignments to each task point according to this sequence by applying Dijkstra's search. This means that for any **fixed** task sequence, RoboTSP can guarantee that the optimal configurations are used. Its limitation, of course, is that there is no guarantee that the sequence itself leads to the true optimal solution for the RTSP. In the Airbus Shopfloor Challenge scenario, where the spatial constraints were relatively relaxed and task points were

[11]The RoboTSP had originally been benchmarked against a GLKH solver (yet another competitive method) for RTSPs, where it had been shown that the RoboTSP was able to find solutions of equal quality to this alternative method. Based on these findings, we can indirectly infer that Cluster-RTSP provides solutions of comparable quality to other methods such as GLKH.

Table 6.3: Parameter settings for algorithm implementation

RoboTSP		**Cluster-RTSP**	
Parameter	Value	Parameter	Value
Stand-off	0.001 mm	Stand-off	0.001 mm
Motion planner	Bi-RRT	Motion planner	Bi-RRT
TSP metric	Euclidean (task space)	TSP metric	Euclidean (C-space)
C-space metric	Max joint difference	Config. select metric	Weighted Sq. Euclidean
-	-	Clustering metric	Sq. Euclidean
-	-	b_δ, b_0	{0.9, 0.1}
-	-	K_{min}, K_{max}	{3, 40}

evenly distributed across a single planar surface, the task space metrics used by RoboTSP generally sufficed for estimating the motion costs between task points. However, as we consider more complex environments we will see that this is not always the case.

To benchmark the planning efficiency of the Cluster-RTSP, the CPU times required by both algorithms were also compared, as shown in Fig. 6.8. For trials involving small values of n (less than 300 points), both Cluster-RTSP and RoboTSP required approximately the same amount of computation time. This is unsurprising as motion planning generally consumes the most computational resources for problems involving relatively small sets of task points. In Section 6.3.2 I showed that the time-saving effects for a small number of task points is minimal. Thus in these circumstances the computational differences between the two algorithms are insignificant as both algorithms apply motion planning in the same way. However, once n exceeded 300 points, the superior planning efficiency of the Cluster-RTSP becomes clear. Taking the best case scenario as an example, in the trial corresponding to 843 task points for a discretisation level of $\frac{\pi}{2}$, the Cluster-RTSP was able to obtain a solution in 19.6% of the computation time required by RoboTSP, yet for this trial both algorithms provided a solution of the same quality. In fact, Fig. 6.7 shows that the computational complexity of the RoboTSP is always exponential in practice due to its use of the standard TSP formulation. These findings are in agreement with the results of the benchmark

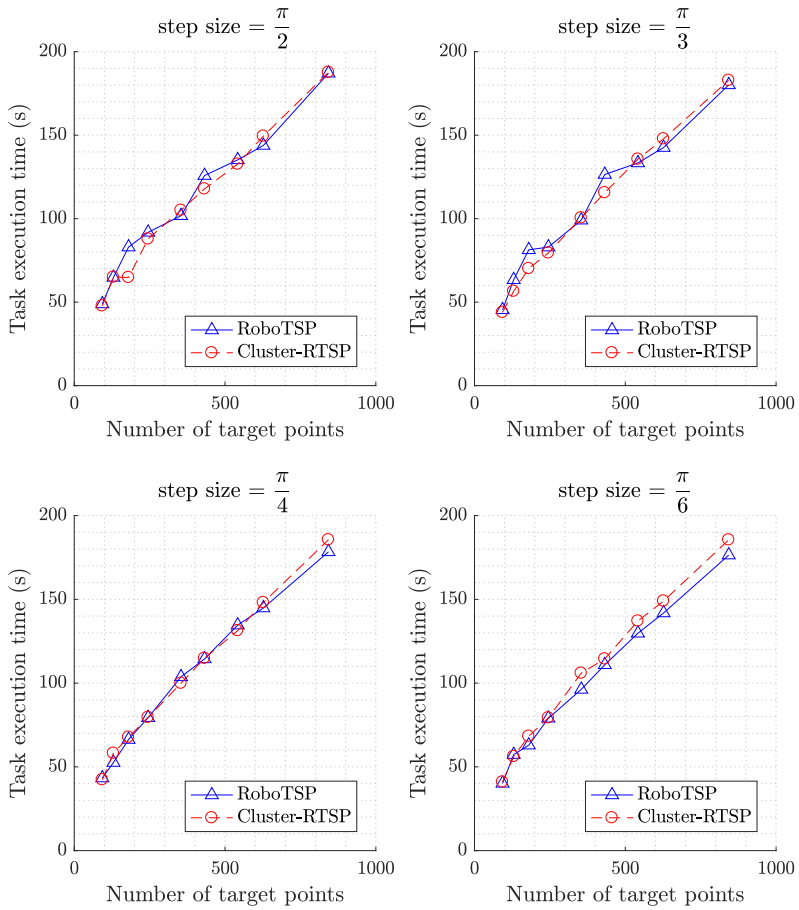


Figure 6.7: Task execution time for RoboTSP and Cluster-RTSP algorithms applied to the Airbus Shopfloor Challenge. The discretisation step size for the 6^{th} free DoF was set to $\frac{\pi}{2}$, $\frac{\pi}{3}$, $\frac{\pi}{4}$ and $\frac{\pi}{6}$ respectively.

on CTSP (see Section 6.3.2). On the other hand, the Cluster-RTSP exhibits a dampened exponential growth, which, for the trials corresponding to step sizes of $\frac{\pi}{2}$ and $\frac{\pi}{3}$, can be closely approximated as a linear growth. This similarly agrees with the theoretical analysis of complexity presented in Section 6.2.3. Since it had already been shown that the RoboTSP algorithm is able to solve the Airbus Shopfloor Challenge scenario several orders of magnitude faster than existing methods, then according to these findings I conclude that the Cluster-RTSP is a competitive algorithm that can solve RTSPs at least as fast as other methods reported in literature.

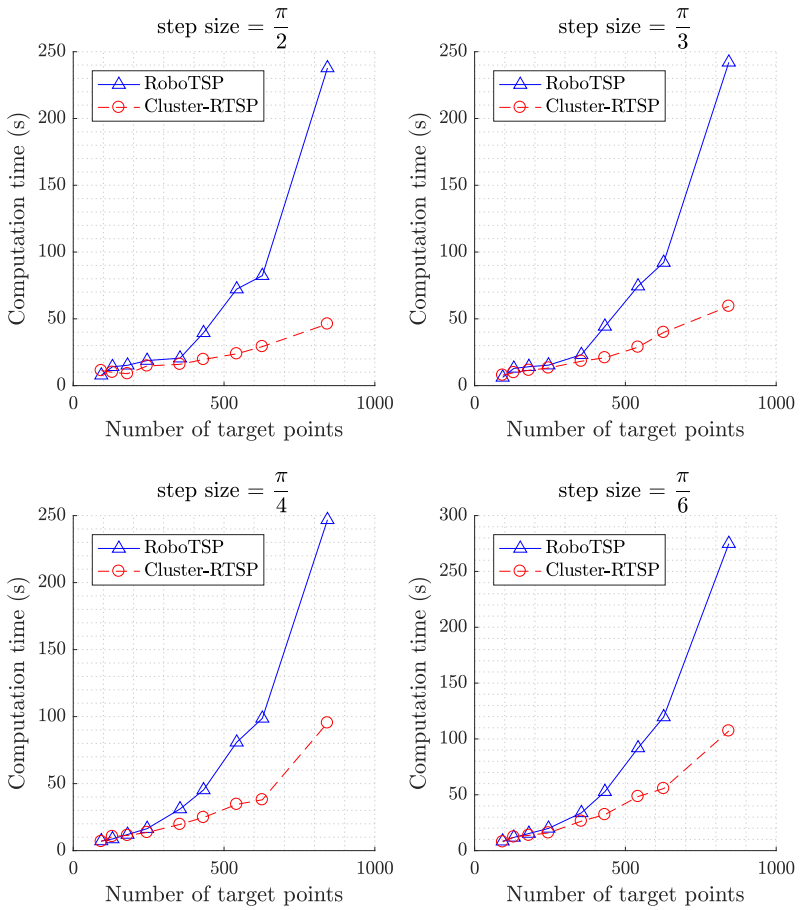


Figure 6.8: Computation time for RoboTSP and Cluster-RTSP algorithms applied to the Airbus Shopfloor Challenge. The discretisation step size for the 6^{th} free DoF was set to $\frac{\pi}{2}$, $\frac{\pi}{3}$, $\frac{\pi}{4}$ and $\frac{\pi}{6}$ respectively.

6.3.4 Benchmarking on Environments A-D

Let us now consider the performance of the Cluster-RTSP for the environments previously shown in Fig. 6.3. By applying the algorithm to solve the sets of planning tasks listed in Table 6.2 for each environment, the behaviour of the algorithm can be tested on RTSPs involving one or more of the following:

- Task points distributed across multiple planes (applies to all trials)
- Planar obstacles that do not invalidate most IK solutions but obstruct any linear motion between task points that lie in close vicinity to the obstacles (applies to Environment B)
- Obstacles located closely to the joints of the robot, which severely limits the robot's range of allowable motions (applies to Environments C and D)

- high cluttered environments that introduce hard spatial constraints to the RTSP (applies to Environment D)

Following the same procedures used in Section 6.3.3, the Cluster-RTSP was benchmarked against RoboTSP for trials involving 224, 328, 561, 763, 919, 1402 and 1844 input task points in each of the four environments. Note that the same sets of task points were used in each environment. However, since each environment introduced different spatial constraints on the robot, only a subset of the input task points were reachable and this varied according to the environment. In the results that follow shortly, the computation time and task execution time are reported for the true number of visited task points and not the size of the input task set. Values for the number of task points visited in each trial according to the environment are provided in Table 6.4, which also gives the number of clusters computed by the X-means algorithm for partitioning the set of assigned configurations. All parameter values reported in Table 6.3 were preserved for this comparison.

Fig. 6.9 shows the task execution times, obtained from solving each set of trials in Environments A-D, plotted against the number of task points. The results corresponding to Environment A show that both algorithms were able to achieve solutions of approximately the same cost, which is in agreement with the comparison made for the Airbus Shopfloor Challenge problem (in both problems no hard spatial constraints are imposed on the robot). The results for Environment B likewise show that both algorithms perform equally in the presence of planar obstacles. This observation could have been anticipated since both algorithms do not evaluate true motion costs to direct the search for an optimal configuration sequence. The obstacles in Environment B has minimal impact on the reachability of task points (i.e. they do not invalidate many configurations since they do not occupy a large volume in space). Rather, they largely act as obstructions for pose-to-pose motions that cannot be detected without performing motion planning queries. Thus both algorithms can be considered 'blind' to these types of obstacles.

As we move onto Environment C, we can begin to see differences in the quality of solutions obtained. In all trials conducted in Environment C, the Cluster-RTSP outperformed the RoboTSP algorithm with reduced task execution times of up 14.5%. These observations extend into Environment D, where the Cluster-RTSP consistently found better solutions than RoboTSP. In the best case, a reduction

Table 6.4: Number of target points reached and corresponding number of clusters obtained for Cluster-RTSP

	Env. A		**Env. B**		**Env. C**		**Env. D**	
Task points	*Points reached*	k	*Points reached*	k	*Points reached*	k	*Points reached*	k
224	224	3	214	3	180	3	128	3
328	328	3	314	3	261	3	201	3
561	561	3	531	3	444	4	342	5
763	763	3	721	3	605	4	448	3
919	919	3	885	3	723	4	554	4
1,402	1,402	5	1,331	8	1,117	4	846	3
1,844	1,844	8	1,772	8	1,459	5	1,095	4

Note: The number of target points visited depends on the constraints imposed by the obstacles in each environment. For environments B-D, certain points cannot be reached due to collision with all IK solutions for the given target point.

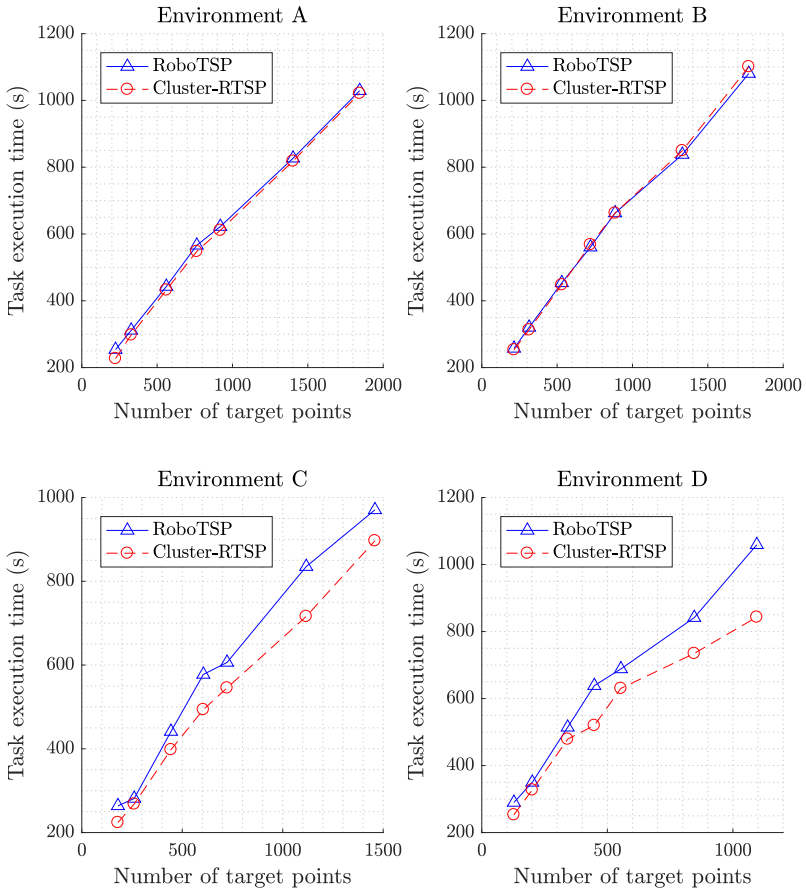


Figure 6.9: Task execution time for problem instances shown in Fig. 6.3

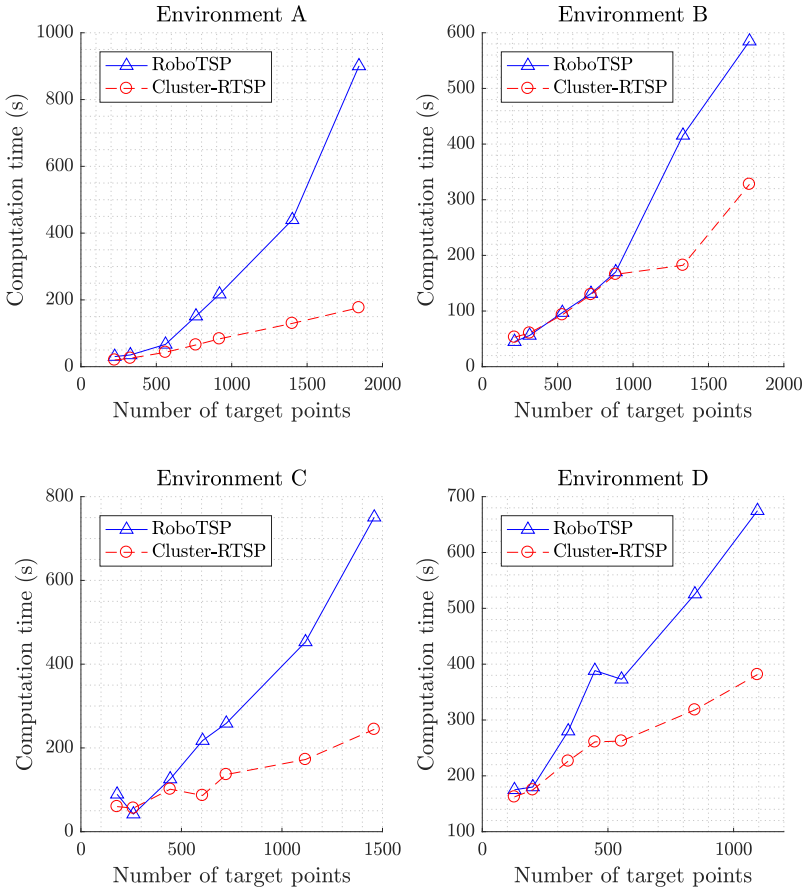


Figure 6.10: Computation time for problem instances shown in Fig. 6.3

of 22.2% in task execution time was achieved with Cluster-RTSP (corresponding to the trial involving 1095 visted task points). These results empirically show that the use of task space metrics to estimate motion costs, a key feature of the RoboTSP and indeed the majority of existing methods, can lead to sub-optimal solutions for problems that involve hard spatial constraints. Conversely, the Cluster-RTSP is able to find better quality solutions as it exploits the availability of C-space information to identify those task points that likely require more costly motions to reach and generating a suitable sequence to account for this accordingly.

Now let us direct our attention to the planning efficiency of the algorithms for these trials. Fig. 6.10 shows the amount of computation time spent to compute the solutions reported above. In addition to the superior solution quality achieved by the algorithm, the Cluster-RTSP outperforms the RoboTSP in planning speed, as was the case in the Airbus Shopfloor Challenge (a time-saving of

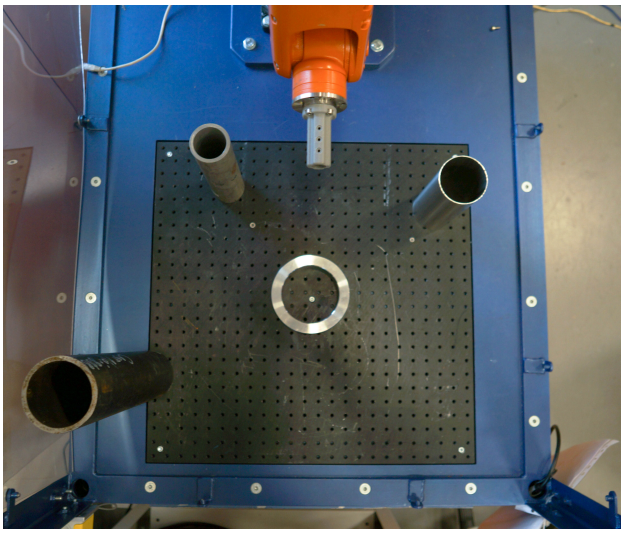
up to 70.3% was found for 1844 task points in Environment A). Interestingly, the size of the planning task for which a difference in planning time starts to become apparent differs across the environments. In Environment B for example, both algorithms required approximately the same amount of computation time for $n \leq 900$ task points. In contrast to this, the Cluster-RTSP was able to find solutions significantly faster for $n \geq 200$. I will show in the next section that these variations are related to the quality of the solution, as a poor sequence can additionally manifest into more time-consuming motion planning queries for environments with hard spatial constraints.

6.4 Experimental Evaluation

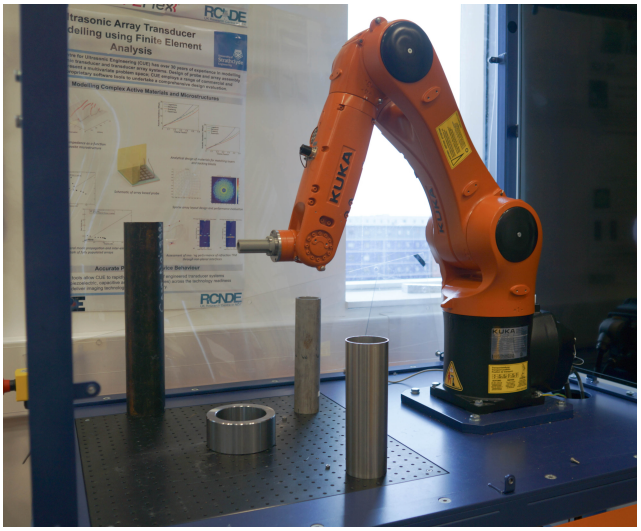
Building upon the evaluations conducted in simulation, I benchmark the Cluster-RTSP on an application example that involves the surface scanning of an arrangement of pipes. In addition to assessing the task performance and computational efficiency of the algorithm, I discuss the advantages and key considerations for the implementation of the algorithm in practical applications.

The setup of the task consists of a KUKA KR6 R900 sixx robot mounted within a cell surrounded by walls on all four sides. A 3D-printed tool is attached to the end effector to represent a typical sensing probe for inspection applications. Four hollow pipes were distributed across the front region of the robot's workspace as shown in Fig. 6.11. This arrangement of objects and surrounding infrastructure impose hard spatial constraints on the robot as it must always remain completely inside the cell while being required to manoeuvre dexterously between the pipes throughout the task. The inspection task itself requires the robot to visit a set of inspection points equally distributed across the outer surfaces of the four pipes. Planning problems involving four sets of inspection points that contained 425, 645, 948 and 1499 points, respectively, were used to compare the performance of Cluster-RTSP against RoboTSP in this application. Again, all parameters reported in Table 6.3 apply in these trials.

To operate the robot, solutions obtained by the algorithms were sent to the robot's control system - a KUKA KR C4 controller - via the ITRA toolbox developed for interfacing with this series of KUKA controllers (see Section 3.5 for details). From among the multiple control methods available in ITRA, I use the computer approach for external control to send the solution as a complete set of



(a)



(b)

Figure 6.11: Physical setup for surface inspection of pipes task with the KUKA KR6 R900 sixx robot in the default 'elbow-up' home configuration. (a) Top-down view, (b) side view.

trajectories, defined in C-space coordinates and interpolated at steps of 12 ms, as described in [159, 192]. When computing the individual time-parameterized trajectories during each motion planning query, the robot joint velocity and acceleration limits were set to 50% of the rated maximum values, which were obtained from the manufacturer's manual [193].

The results for task execution time are provided in Fig. 6.12. Unlike previous trials where task points were distributed across planes, in this application where task points are more freely distributed across the entire task space. Furthermore,

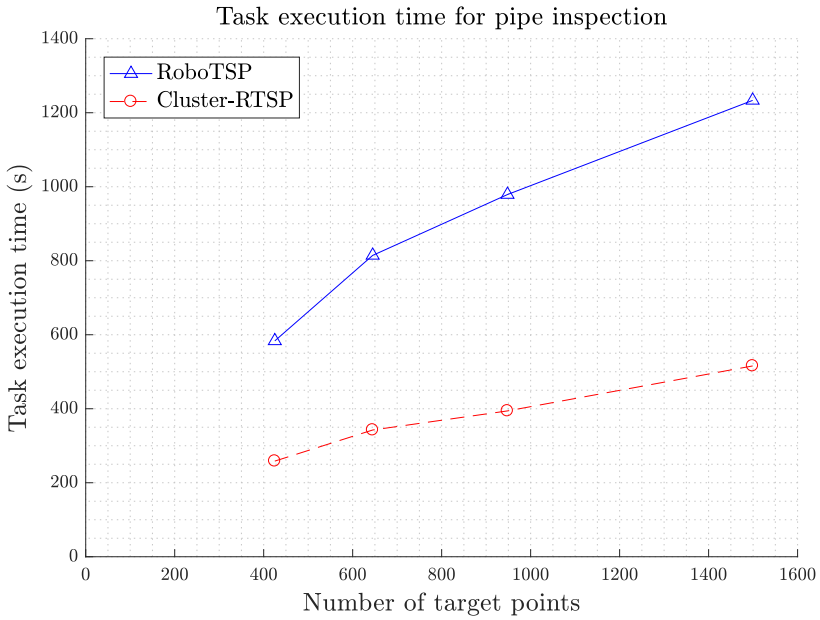


Figure 6.12: Task execution times for the pipe surface inspection task.

the layout of the robot's environment introduces substantial spatial constraints on the robot that were not present in some of the earlier trials. When solving an RTSP of this nature, Fig. 6.12 shows that the Cluster-RTSP can achieve significant performance improvements compared to the RoboTSP. Across the four trials, the Cluster-RTSP was consistently able to reduce the task execution time by 55-60%. This level of performance could be achieved as the Cluster-RTSP algorithm was able to identify the small subset of task points whose collision-free IK solutions were drastically different from the default 'elbow-up' configuration seen in Fig. 6.11b. The Cluster-RTSP clusters these configurations together when partitioning the set of assigned configurations Q', thus ensuring that these configurations are visited contiguously. The RoboTSP on the other hand is unable to recognise these features and thus finds itself alternating between drastically different configurations where the corresponding task points lie close together in the task space. From a practical standpoint, these contrasting behaviours translate into substantial differences in productivity.

Fig. 6.13 reports a breakdown of the computation time in terms of the high-level components for both algorithms. This compromises of the TSP solver and motion planning procedures common to both algorithms, while all other sub-functions unique to each algorithm is collectively grouped as miscellaneous operations. Notice that in all cases the computation time for solving the TSP is

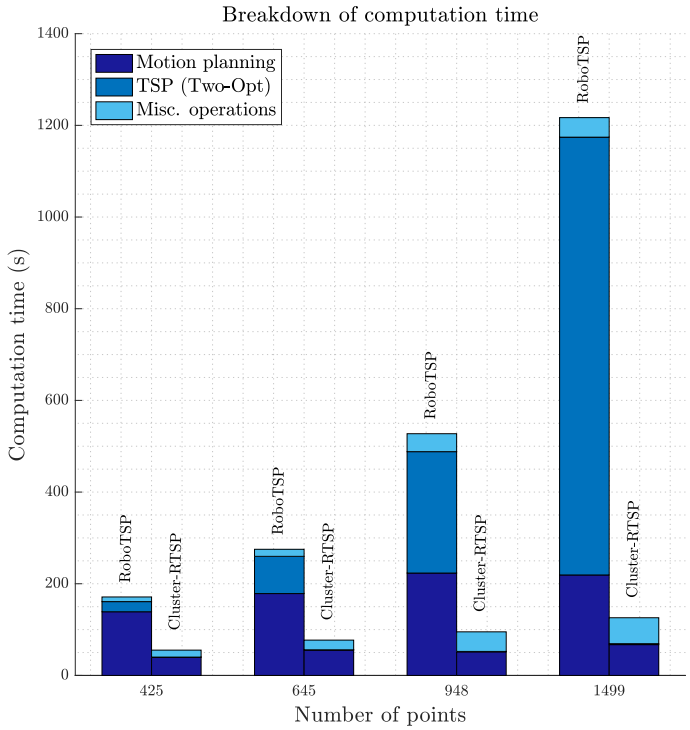


Figure 6.13: Breakdown of computation time for the pipe surface inspection task. *Misc. operation* includes all functions in each respective algorithm excluding motion planning and solving the TSP with 2-Opt.

negligible for the Cluster-RTSP, while for the RoboTSP the computation time is dominated by this step for $n = 948$ and $n = 1499$. Interestingly, while both algorithms use exactly the same motion planner (Bi-RRT implemented on OpenRAVE), motion planning incurs a significantly greater computational cost in the RoboTSP. As briefly mentioned earlier in Section 6.3.4, a poor quality sequence can negatively impact the planning time for fulfilling motion planning queries. This is because finding a collision-free trajectory between two very different configurations is a more difficult problem than finding a collision-free trajectory between two similar configurations in the C-space (particularly in the presence of obstacles). As discussed above, the quality of the sequence obtained by the RoboTSP is substantially poorer for this application, which leads to more costly motion planning queries. Overall the Cluster-RTSP is able to achieve up to 89.7% faster planning as a result of both the CTSP formulation and the reduced complexity in motion planning queries. When we consider the complete process cycle that encapsulates both the time required for planning and for execution, the Cluster-RTSP has the potential to reduce the total time to complete a given task

by several folds, depending upon the difficulty of the RTSP (for the inspection task comprising 1499 points, the Cluster-RTSP was able to save 1808.7 seconds altogether from planning through to execution). This has clear advantages for one-off applications where any single plan is executed only once.

I now give particular attention to some of the key considerations that apply when implementing the algorithm for practical applications. To contextualise this, a set of trajectory tracking experiments were conducted on the surface inspection task described earlier in this section. Using a task point set of $n = 425$, I vary the maximum joint velocity and acceleration limits across the robot's full working range (from 10% through to 100%) and apply the Cluster-RTSP algorithm to perform the inspection task. The solution was sent to the robot for execution using the ITRA toolbox. Conveniently, the ITRA toolbox enables the feedback of true robot coordinates to be returned to the host PC to monitor the state of the robot at each interpolation cycle. This enables the evaluation of the trajectory tracking error across the entire duration of the task by comparing the sent coordinates against the robot's true coordinates at any given time step. Let us consider the following three specific types of error:

- Maximum joint error - the error in joint angles given by the maximum value among the individual joint angle difference between the sent coordinates and feedback coordinates (in degrees)
- Total joint error - the error in joint angles given by the Euclidean distance in C-space between the sent coordinates and the feedback coordinates (in degrees)
- Total position error - the end effector position error given by the Euclidean distance in task space between the forward kinematic solution of the sent joint coordinates (equivalent to a sent position) and the feedback position (in millimetres)

Fig. 6.14 shows the maximum trajectory tracking errors between the executed and sent trajectories across the range of velocity and acceleration limits. It is apparent that the largest trajectory tracking errors were observed when the limits were assigned values below 50%, where the peak total position error was recorded at 15.2 millimetres, while the maximum joint error and total joint error reached 2.8° and 3.7°, respectively. For all other velocity and acceleration limits,

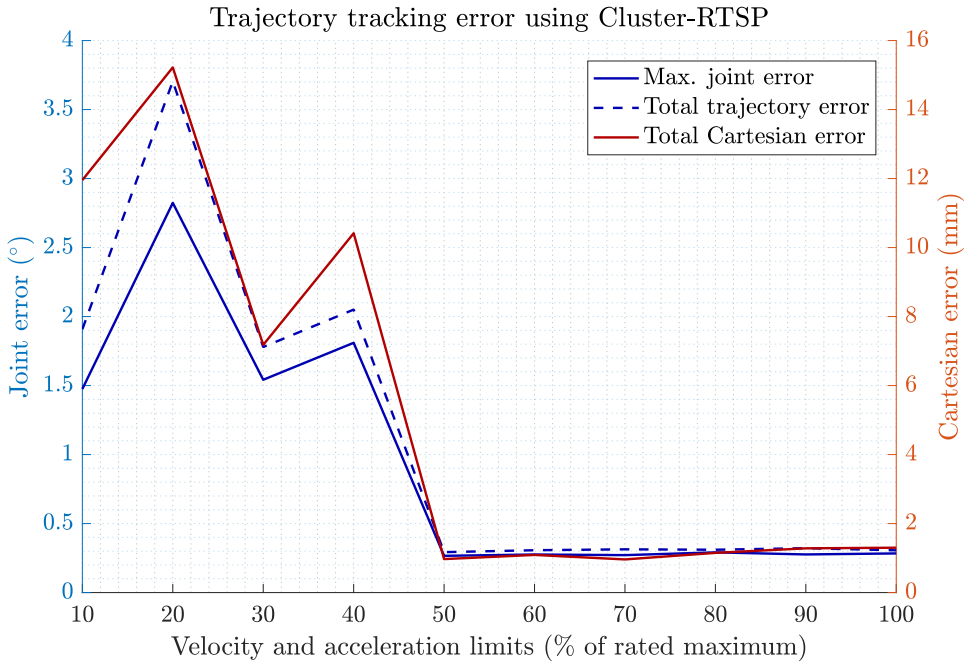


Figure 6.14: Trajectory tracking error for pipe surface inspection experiments.

the position and joint errors consistently remained below 2 millimetres and 0.5°, respectively.

I first wish to clarify that these errors did not originate from the planned motions generated by the Cluster-RTSP. Rather, these errors arise from the limitations of the hardware and are specific to the test system. This is not universally true for all RTSP methods, as algorithms that do not take singularities into consideration may lead to unintended stops during execution due to failure of the robot to transition through a singularity. That is why some authors of previous work have mentioned the necessity for explicit consideration of robot *manipulability* to identify regions in task space near singularities [19, 107]. However, these problems are avoided in the Cluster-RTSP as it solves the RTSP entirely in C-space. Nevertheless, systems in the real world often do not perform exactly as it should in ideal circumstances. Trajectory tracking error is one such common observation in practice and should therefore be considered when implementing methods to solve RTSPs for critical applications (e.g. the oil & gas and nuclear sectors commonly involve sensitive or high-risk applications where precision is paramount to the safety of the facility). In the particular system used in this experiment, a practitioner would need to consider the benefits of executing a robot will low velocity and acceleration limits against the reduced trajectory

tracking performance achievable at those limits. Of course, these considerations are specific to the robotic system deployed.

To deal with more general limitations of physical systems in the real-world, a number of RTSP implementation considerations can be taken into account to overcome tracking errors that cannot be tolerated by default. In applications that require the robot to maintain a minimum amount of clearance from obstacles, a common practice used within the robotics research community involves the enlargement of obstacles, either by inflating obstacles while preserving their 3-dimensional form, or by applying various types of bounding boxes for a simpler representation. Generally, attention should be given to the effects of obstacle enlargement, as it can introduce substantially greater spatial constraints not previously present. The evaluations in this chapter have shown that the performance of the Cluster-RTSP does not deteriorate when faced with problems that involve hard spatial constraints. This is not the case for those existing methods that do not account for spatial constraints when developing task sequences, while other methods that do have not yet been proven to do so efficiently.

For some applications there may exist contact-based interactions between the robot end effector and a workpiece. In these scenarios, virtually enlarging all objects in the environment to avoid minor collisions may prevent the RTSP solver from finding a solution, as the robot must move close to the surface of the workpiece to perform its tasks. Instead, it is possible to introduce *standoff distances* to offset the actual positions reached by the end effector when visiting task points. This standoff distance should be determined according to the maximum position error of the system to ensure that the robot does not initially make contact with the workpiece as a result of tracking error. Where contact with the workpiece must then be made in order to accomplish a given task (e.g. for drilling tasks and contact-based inspection), force-torque sensing could be used for fine precision in order to achieve contact with the workpiece.

6.5 Summary

In this chapter I have presented a computationally efficient, near-optimal algorithm for solving general RTSPs. The algorithm has been applied to a number of different planning problems to demonstrate its effectiveness in addressing problems of varying complexity. I have carefully benchmarked the algorithm to evalu-

ate the performance of individual components and the overall performance of the algorithm. Through an in-depth comparison with the RoboTSP, a state-of-the-art approach in literature, I have shown that the Cluster-RTSP is able to find higher quality solutions to RTSPs involving hard spatial constraints while being computationally more efficient than existing methods for problems involving large sets of task points (as Section 6.4 showed, the Cluster-RTSP was able to reduce the task execution time by up to 60% and the computation time by up to 90% compared to the RoboTSP). When we consider the complete process cycle time encapsulating both planning and execution, the Cluster-RTSP has the potential to reduce the total time for task completion by manifold. Even in the worst case, the Cluster-RTSP is able to provide solutions of comparable quality to existing methods while always being at least as fast at finding this solution (this generally occurs for simple problems where existing methods can adequately approximate the true optimal solution). Finally, a discussion on the considerations for practical implementation was provided for readers interested in the practicalities of the algorithm.

In the next chapter I provide a further investigation into the Cluster-RTSP algorithm to show how it can be extended to enable **partial planning** and **re-planning**, concepts which (to the best of my knowledge) have not been considered so far in the RTSP literature.

Chapter 7

Towards Dynamic Robotic Task Sequencing

7.1 Introduction

In the previous chapter I introduced the Cluster-RTSP algorithm, a new method for solving manipulator-based RTSPs. It is characterised by fast planning times and near-optimal solutions even in the presence of hard spatial constraints. However, until now the RTSP has solely been addressed as a static planning problem. With the exception of RoboTSP [135] and Cluster-RTSP, all existing methods of solving RTSPs in literature require extensively long planning times even for problems with a relatively small set of task points. One of the motivations for the development of the Cluster-RTSP was the limitation of these existing methods being strictly for offline planning only. Certainly, these methods provided a number of benefits in removing the necessity for a skilled programmer to develop plans for RTSPs. Developing plans for RTSPs manually can be taxing on a human programmer, especially for complex problems. These plans are prone to human errors, and it is generally difficult to guarantee a high quality solution when the set of task points is large. Manually solving RTSPs is very time-consuming to begin with, but for safe implementation further preparation time is required to validate plans to ensure robot motions are collision-free and tasks are correctly executed as required by the application. Through the use of existing methods developed for RTSPs, all of these problems can be overcome. Yet, as discussed in Chapter 6, the majority of these methods are inadequate for applications that

demand short planning times.

Taking these practical implementation considerations into account, the Cluster-RTSP algorithm extends the range of applications that can be addressed through RTSP planning. Though the RoboTSP is considered a fast method in its own right, the complexity analysis presented in Section 6.2.3 and the experimental evaluations presented in Sections 6.3-6.4 show that the Cluster-RTSP supersedes the computational performance of RoboTSP (by up to 90% in trials considered) for large sets of task points. To the best of my knowledge, no other method in the RTSP literature has reported computational efficiency within the same order of magnitude as Cluster-RTSP for similar problem sizes.

In this chapter I present work that further pushes the state-of-the-art in RTSP by exploring two new concepts within the context of this problem. Motivated by the ambition to enable adaptive planning capabilities in RTSP, I introduce the concept of *Dynamic* RTSP (**DRTSP**), which considers applications in which the problem parameters (e.g. tasks or environment) change during execution. Like in the static case, the problem can be considered a close relative of the *dynamic* TSP (DTSP), but with greater complexity due to the additional considerations for kinematic redundancy, collision avoidance and motion planning. As an intermediate step towards realising this capability, I also introduce the concept of partial planning to RTSPs, which adopts the idea of planning-during-execution familiar to the anytime planning component of the DA-TPP presented in Chapter 5.

Using the Cluster-RTSP as the core building blocks, I describe two additions that exploit the clustering feature of the algorithm to enable **partial planning** and **dynamic re-planning**, respectively. Partial planning can be implemented stand-alone from dynamic re-planning and allows the robot to begin executing a task plan before a complete task sequence has been determined. From a practical perspective, this further reduces the idle time of the robot during planning. Dynamic re-planning extends the elements of partial planning to provide online adaptation of a task sequence for applications that are prone to dynamic changes in the environment. These changes can affect the validity of the configurations assigned to task points, or obstruct the planned trajectories for moving between task points. Dynamic re-planning handles these events online through the re-assignment of configurations to task points, re-planning of partial plans, and re-planning of trajectories between configurations, avoiding the costly process of

re-planning from scratch.

With these in mind, the contributions of this chapter is as follows:

1. I introduce and define the concept of DRTSP, a new variant of the RTSP that extends the original static optimisation problem (SOP) into a dynamic optimisation problem (DOP), which encapsulates dynamic change in problem parameters relating to the task or environment.

2. I describe the concept of partial planning in the context of RTSP and present a modification to the Cluster-RTSP that makes use of configuration clustering to conveniently sub-divide a task into smaller **sub-tasks** for execution. By solving for an **executable partial plan** of each sub-task sequentially, I show that execution can begin before planning of all sub-tasks is complete. Simulation-based evaluations are used to quantify the reduction in pre-execution planning time for the pipe scanning task considered in Section 6.4. I refer to this modified version of Cluster-RTSP as *partial*-Cluster-RTSP (***p*-Cluster-RTSP**).

3. I present an approach to solving the DRTSP that is based on extending the partial planning variant of the Cluster-RTSP algorithm. This extension, henceforth referred to as *dynamic*-Cluster-RTSP (***d*-Cluster-RTSP**), provides the following capabilities: (i) reassignment of valid IK solutions to task points whose previously assigned configurations become infeasible during execution due to collision, (ii) efficient updates to sub-tasks and re-planning of partial plans to maintain an optimal plan corresponding to the changes in (i), and (iii) re-planning of trajectories to avoid collision with newly detected obstacles during execution. A preliminary investigation into the performance of the *d*-Cluster-RTSP is reported, demonstrating the potential of addressing DRTSPs through integrated task planning and motion planning, and providing a set of quantitative evaluations for comparison in further developments.

The rest of this chapter is organised as follows. In Section 7.2, I present the fundamentals of DRTSP and provide a definition to this new variant of RTSPs. Section 7.3 is dedicated to the modifications made to Cluster-RTSP to provide the partial planning and dynamic re-planning variants of the algorithm. Then, in Section 7.4, I describe the simulation-based evaluations of the two planners,

respectively. A discussion on the DRTSP and the two new variants of the Cluster-RTSP is provided in Section 7.5, and finally Section 7.6 summarises the chapter.

7.2 Fundamentals of Dynamic RTSP

The original RTSP for which all existing literature was aimed at solving is classed as an SOP, where the objective is to find the best solution from among the set of feasible solutions given fixed problem variables that do not change with respect to time. This implies that offline planning methods can be adequately used to solve a standard RTSP since any solution obtained would remain valid when sent for execution. Interestingly, so far no existing work has considered the *dynamic* variant of the RTSP, which can be considered a DOP extension of the problem.

In optimisation theory, the DOP is a widely considered class of problems that seek to *track* the optimal solution for a problem involving dynamic variables that change over time. Indeed the ITPP problem described for MWRs in Chapter 5 belongs to this category, where one of the objectives of the DA-TPP framework was to update the action-motion sequence to maintain a high quality solution as changes in the environment were observed.

Setting the RTSP within a similar context, let us consider sequencing problems for which the dynamic variables comprise of changes to the environment and the task itself. An obvious scenario that involves these dynamic variables is in human-robot collaborative tasks, where a dynamically moving subject (the human) shares the same workspace with the robot. For both safety and productivity reasons, we would ideally like the robot to re-plan around the human when interference is detected, rather than sit idly until the human no longer invades the original set of planned motions. Other examples where dynamics come into play can be found in applications involving the inspection of corroded and damaged parts (such as radioactive waste containers), where the geometries of these objects no longer match their original models. In these instances, it may be necessary to update the distribution of task points to account for geometric deformations. As a final example, consider applications that involve the cutting or reshaping of a workpiece. As these operations are performed, unforeseen loss of structural integrity can result in deformations that act as an obstruction for subsequent trajectories or require the addition of further task points to recover from unsuccessful cutting/reshaping. Generally, applications that carry a certain

degree of uncertainty in the task can be considered a dynamic process.

It is important to note that in terms of complexity, the dynamic variant of RTSPs is unique even among general DOPs. While general DOPs are concerned only with satisfying an objective function under the influence of dynamics, in RTSPs collision avoidance must be accounted for explicitly. This necessity for online collision-checking during execution arises from both types of dynamic variables (changes in the task and the environment). Addressing these dynamic variables for RTSP must therefore take into consideration not only the quality of sequencing but also the feasibility of individual configurations and trajectories. Taking all of these into consideration and building upon the notation introduced in Section 6.2.1, I define the **Dynamic RTSP (DRTSP)** as follows.

Definition (Dynamic RTSP)**.** *Given the dynamic set of non-static obstacles $O(t)$ and the dynamic set of task points $P_n(t)$, where t is the real-world time, find and track the optimal configuration sequence $S_C^*(t)$ that enables every reachable point $\boldsymbol{p}_i \in P_n(t)$ to be visited once in a contiguous sequence while guaranteeing that the set of executed trajectories Ω_{exe} used to visit each configuration $\boldsymbol{q}_i \in S_C^*(t)$ is collision-free.*

As noted in Chapter 2, there is a lack of literature that has specifically investigated the DRTSP. Perhaps the closest resemblance to the dynamic nature of the problem is found in *Dynamic* TSP (DTSP), which describes a class of TSPs where the number of cities and corresponding distances between cities vary with time. I refer the reader to Section 2.2.4.2, where a literature review of related work has been provided.

Based on the findings of the literature review, I wish to highlight a number of remarks with regards to the differences between DRTSP and DTSP. Although a number of commonalities exist between the two types of problems, the DRTSP possesses additional unique features that make the problem substantially more complex than the DTSP. First of all, the existence of kinematic redundancy specific to the manipulator planning domain significantly magnifies the search space. As changes in the environment affect the feasibility set of IK solutions, the problem of assigning configurations must be dynamically handled in some way to optimise the configuration sequence. Secondly, aside from updating the visiting sequence to adapt to the addition and removal of task points, the DRTSP involves explicit considerations for dynamic motion planning to ensure that the

robot does not collide with the dynamic environment when moving between task points. As a result, despite the success of EAs for DTSPs, they do not translate well for solving DRTSPs (as we saw in Chapter 6, GA is generally inferior to RTSP-specific methods due to the lack of an efficient means to handle kinematic redundancy).

In addition to the above, the methods developed to address DTSPs have generally sacrificed solution quality to some extent to enable more efficient planning. For EAs, these relate to the parameter settings unique to each algorithm (e.g. the number of iterations performed for iterative methods). However, in the general DTSP literature, the best quality solution achieved by an algorithm appears to be the primary means of evaluating performance and has usually been compared through offline planning trials. Less attention has been given to the planning efficiency of algorithms as these studies have, for the most part, not been concerned with achieving real-time performance (in fact several of the work discussed above had no mention of the required computation time). However, in adaptive robotics, there is a much greater emphasis on achieving short planning times in dynamic scenarios to enable online adaptiveness for avoiding collisions and minimising idle time.

Indeed one of the long-term goals of adaptive robotics is to provide *real-time* planning behaviour. Putting this in the context of DRTSPs, we must acknowledge a number of limitations. Even in the standard RTSP we have found that it is difficult to obtain a close approximation of the true optimal solution due to its notably greater problem complexity compared to TSPs. Taking into consideration the current challenges in balancing solution quality and planning efficiency in DTSPs, some form of trade-off between achieving high quality solutions and achieving near real-time performance is necessary in the DRTSP. Furthermore, even with the current state-of-the-art in DTSPs, the fastest re-planning performance achieved well exceeds the threshold to be considered truly *real-time* (I refer back to the definition provided in Chapter 3). Since the DRTSP additionally handles motion planning, collision avoidance, inverse kinematics and configuration assignments, we cannot expect to achieve truly real-time adaptiveness in dynamic scenarios. For these reasons, when evaluating the method for addressing DRTSPs presented in the remainder of this chapter, I place particular emphasis on quantifying the **near** real-time performance of the algorithm with a view of setting a benchmark for future developments on DRTSPs.

7.3 New Variants of the Cluster-RTSP

This section presents a number of modifications to the Cluster-RTSP algorithm aimed at providing the algorithm with the capability to handle DRTSPs. As an intermediate step towards developing the adaptive capability required to re-plan dynamically during execution, I first present the p-Cluster-RTSP, a modified version of the original algorithm that supports the idea of partial planning. I then introduce the d-Cluster-RTSP, which further builds upon the original algorithm to directly address DRTSPs.

In this section, I will refer back to the standard Cluster-RTSP architecture to aid the description of these algorithms and highlight the key modifications. This standard architecture corresponds to the flow chart shown in Fig. 6.1.

7.3.1 p-Cluster-RTSP

In dynamic scenarios, an extensive plan devised in advance suffers from the risk of becoming irrelevant by the time it comes to execution due to potential changes in the environment. This renders a notable amount of pre-execution time wasted as a new plan must be computed once the changes are perceived. One approach to minimising this risk is to delay planning until the very last moment. This opens up the way for the concept of partial planning, where the set of task points are divided into subsets that form smaller sub-tasks. The process of computing a **partial plan** for each sub-task can then be performed as separate planning instances solved in a decoupled manner. By making this smaller plan self-contained (meaning that the partial plan supplies sufficient information to enable the completion of the entire sub-task taking into account the robot's current state when sent to the robot for execution), it becomes possible to compute a preliminary partial plan for the first sub-task that is sent for execution, while all subsequent sub-tasks are solved online during the execution of the previous sub-task.

For now let us consider this concept for a purely static RTSP (we will return to the dynamic considerations in Section 7.3.2). The Cluster-RTSP provides a convenient yet deliberate mechanism for dividing the set of task points P_n into multiple sub-tasks through the partitioning of assigned configurations into clusters. Additionally, the sequencing of configurations within each cluster is solved as a local TSP just as they would be when formulated as a sub-task sequencing problem. In other words, the Cluster-RTSP easily extends to partial

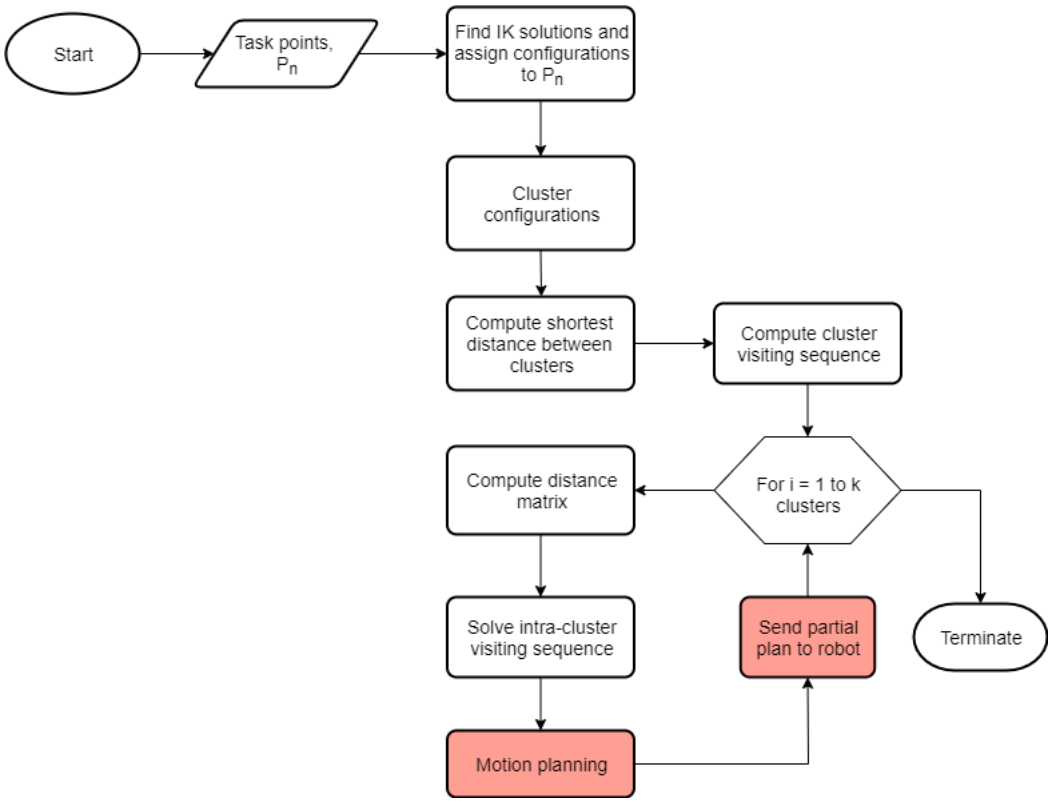


Figure 7.1: Software architecture for the p-Cluster-RTSP algorithm.

planning with very few modifications. I introduce here the p-Cluster-RTSP, which provides a minor modification to the Cluster-RTSP to enable partial planning behaviour as shown in Fig. 7.1 (modifications have been highlighted in color).

In the standard Cluster-RTSP algorithm previously shown in Fig. 6.1, the computation of trajectories for moving between successive configurations in the task sequence was performed once all TSP sub-problems had been solved. Only when all trajectories have been obtained is the complete plan sent to the robot. In contrast, the p-Cluster-RTSP performs a set of motion planning queries for each local intra-cluster sequence as they are obtained. This provides a partial plan, Γ that enables the sub-task consisting of the task points contained in the local cluster of configurations to be executed by the robot. Now, rather than awaiting for the entire plan before beginning execution, each partial plan is sent to the robot prior to solving the next TSP sub-problem. From a practical point of view, this means that the execution of a task can begin much sooner than before, while the planner continues to devise a plan for visiting task points later in the global sequence.

Notice that in the p-Cluster-RTSP, all steps up to the computation of the inter-cluster visiting sequence is kept unchanged. In other words, although the algorithm decouples the computation of TSP sub-problems, it initially considers all task points when identifying the grouping of task points (represented by assigned configurations) and the high-level sequence for visiting the groups of task points. This guarantees that the overall solution is still optimised to reduce the total execution time of the entire task as in the original version of the algorithm.

Finally, it is necessary to highlight that although the concept of partial planning was motivated by the desire to delay planning until directly before a sub-task is executed to minimise the risk of dynamic changes rendering a plan irrelevant, the p-Cluster-RTSP does not actually behave in this way. Rather than wait for the robot to complete the execution of the current partial plan, the algorithm continues to plan for all subsequent sub-tasks in isolation from the execution process. If another partial plan to a later sub-task is obtained before the previous one has finished executing, the new partial plan is placed in a queue to be sent to the robot once it is freed from the earlier sub-task. This means that it is possible for the algorithm to finish devising plans for all sub-tasks before the robot completes the execution of the first sub-task. I therefore wish to emphasise that the p-Cluster-RTSP does not adequately address DRTSPs in any way.

Nevertheless, the p-Cluster-RTSP was deliberately developed as its own individual variant of Cluster-RTSP as it offers a notable advantage over the standard Cluster-RTSP in static applications. This variant significantly reduces the pre-execution planning time required to obtain an initial set of executable actions that contribute towards achieving the objectives of the RTSP. Furthermore, it accomplishes this without sacrificing solution quality. That is, for a static RTSP the p-Cluster-RTSP and the standard Cluster-RTSP results in the same plan being executed at the end of all planning procedures. In other words, p-Cluster-RTSP produces solutions of the same quality but additionally minimises the idle time of the robot while a plan is being devised at the start of a task. For applications where substantial amount of time is available for offline planning, this does not necessarily offer particular benefit. However, as we shift towards the direction of online planning for RTSPs, where pre-execution planning times directly affects a robot's idle time, p-Cluster-RTSP proves superior.

It is important to note that there exists some situations where the behaviour of p-Cluster-RTSP is less desirable than Cluster-RTSP. One drawback of partial

planning is the inability to visualise and assess the intended actions of the robot before execution. The standard Cluster-RTSP allows an operator to assess the complete solution and make any manual adjustments desired prior to deployment (this could be achieved through means such as simulations and numerical analyses). p-Cluster-RTSP does not allow for this as the plan for visiting task points later in the sequence is not known until during execution. This is a particularly important consideration for applications where strict safety regulations must be met (e.g. in the oil and gas, nuclear and petrochemical sectors).

7.3.2 *d*-Cluster-RTSP

The p-Cluster-RTSP was introduced above as a partial planning variant of the Cluster-RTSP for static RTSPs. Building upon the principle of partial planning, let us now consider the DRTSP that forms the core of this study.

To adequately address the effects of change in a DRTSP, a planner is required to support some or all of the following:

1. Reassignment of a feasible IK solution to a given task point p_i when the previously assigned configuration to p_i becomes invalid due to collision with $O(t)$ at time t.

2. Update the task sequence with new task points that are added to the set $P_n(t)$ over time t

3. Treatment of task points that become unreachable due to $O(t)$, or equivalently the removal of task points from $P_n(t)$, at time t

4. Re-planning of trajectories to avoid collision with $O(t)$ at time t when moving between task points

5. Tracking of the optimal configuration sequence $S_C^*(t)$, which varies with t as a result of 1-3.

The d-Cluster-RTSP, presented herein, is proposed to address each of the above features unique to the DRTSP. It was developed under the key underlying assumption that the new best solution to a sequencing problem after changes to $O(t)$ and $P_n(t)$ is close to the previous best solution as discussed in the analysis of DTSPs provided by Tinós [150]. This generally remains true for problems

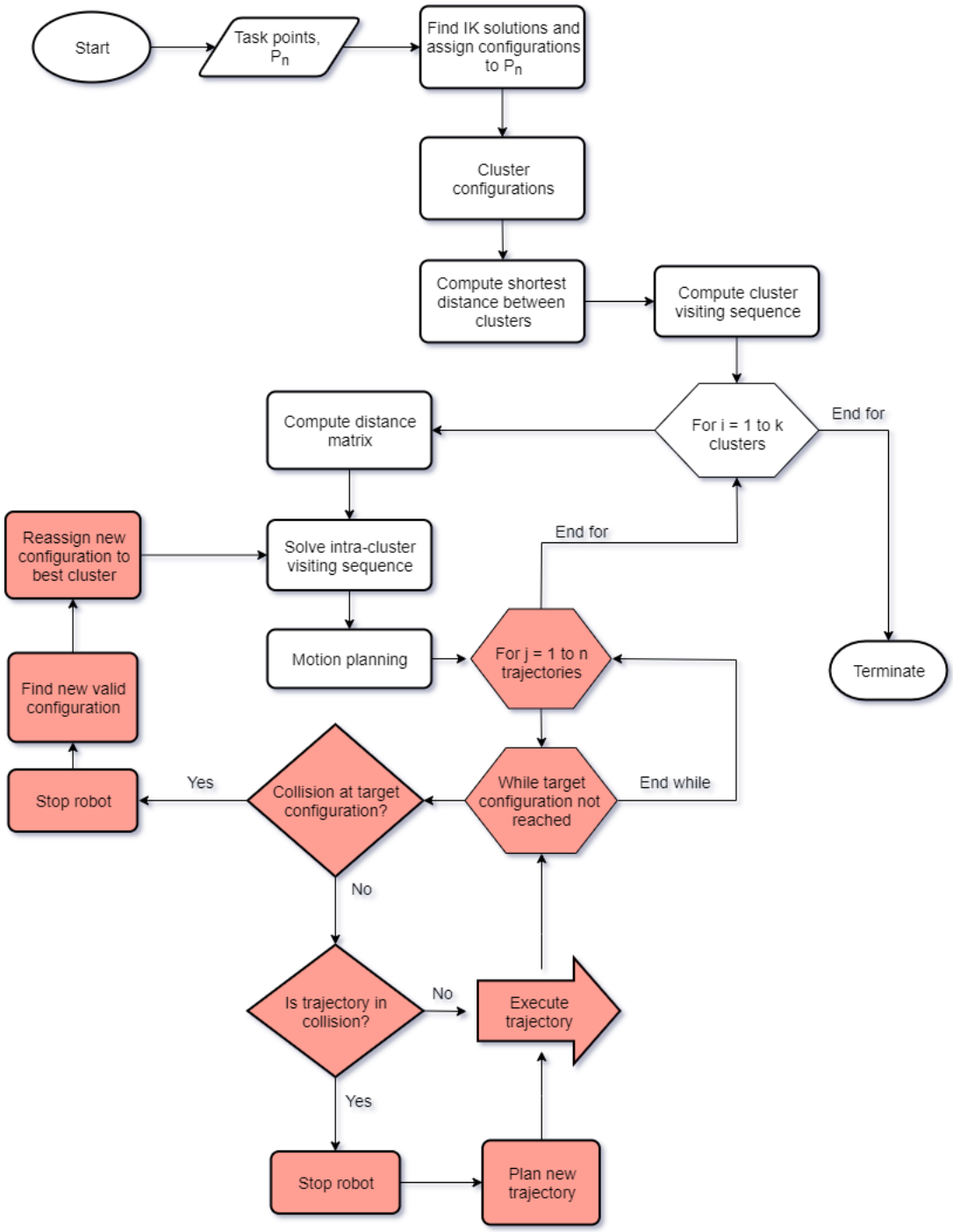


Figure 7.2: Software architecture for the d-Cluster-RTSP algorithm.

involving a large set of $P_n(t)$ and where the frequency/magnitude of changes at any given time step Δt are considered small.

The d-Cluster-RTSP builds upon the architecture of the Cluster-RTSP al-

gorithm and adds three important online re-planning routines to the original algorithm. These are: (i) configuration reassignment, (ii) sub-task re-planning and (ii) trajectory re-planning. The overall high-level description of the algorithm is shown in Fig. 7.2 and comprises of the following. The standard procedural steps required to obtain a high-level cluster visiting sequence is performed as per usual and forms the pre-execution planning procedures in the same way as p-Cluster-RTSP. Afterwards, the mechanism for solving sub-tasks individually to obtain partial plans is adopted from p-Cluster-RTSP. However, once the set of trajectories is obtained for one sub-task, rather than continuing to plan for the next sub-task, the algorithm recursively re-evaluates the validity of the current plan using the latest task set and the currently known state of the world as the robot executes the sub-task. This continues until the sub-task is completed.

The re-evaluation procedures comprise of the three re-planning routines introduced above. For each trajectory planned within a sub-task, the algorithm first checks the validity of the target configuration (i.e. the goal of the trajectory) corresponding to a task point p_i in the sub-task. if a collision is detected at this configuration, a **configuration reassignment** routine is used to find a new configuration for p_i. It achieves this by finding the new set of feasible IK solutions for p_i and computing the similarity heuristic value ϕ (Eq. 6.2) for each resulting configuration. The configuration q_i' corresponding to the lowest value of ϕ (which describes the configuration that is most similar to all the other configurations assigned to the set of task points $P_n(t)$) is assigned to p_i. Once the new configuration q_i' is obtained, the **sub-task re-planning** routine re-allocates p_i to the sub-task (from among the remaining set of incomplete sub-tasks) that minimises the cost incurred to reach the new configuration. This is achieved by comparing the C-space distance between q_i' and the centroids of all clusters $x_c \in X$. The configuration is then assigned to the cluster associated to the nearest centroid. If this corresponds to the current sub-task, a new partial plan is generated by re-solving for the intra-cluster visiting sequence and computing new trajectories where the order of configurations is different to the previous plan. Otherwise, the current partial plan is repaired by computing a trajectory from the current robot configuration to the next configuration in the sub-task. If q_i' is added to another cluster, the corresponding x_c is updated to account for this new member.

Note that in the above procedures, two special cases exist that require additional considerations. In the first case where no feasible configurations exist for

Algorithm 6 configurationReassignment

Input: Task point p_i, obstacles $O(t)$ and set of assigned configurations Q'
Output: New assigned configuration q_i'
1: $Q_i \leftarrow getIKSolutions(p_i)$
2: **if** Q_i is $\varnothing$ **then**
3: **return** $q_i' = \varnothing$
4: **else**
5: **for all** $q_i \in Q_i$ **do**
6: $\phi(q_i) \leftarrow computeSimilarityHeuristic(q_i, Q')$
7: **end for**
8: **return** $q_i' \leftarrow \arg\min_{q_i \in Q_i}(\phi(q_i))$
9: **end if**

p_i (i.e. the task point becomes unreachable), the point is stored in an **unreachable list** (initially containing those task points that could not be reached when computing IK solutions in step 1 of the standard Cluster-RTSP algorithm). This list can be used to initiate a second instance of RTSP once all sub-tasks have been completed to re-attempt to visit those task points that became unreachable during execution, or it may simply be returned to the operator to inform them of the incomplete tasks. In either instance, the algorithm resumes by repairing the partial plan with a new trajectory from the robot's current configuration to the next configuration in the partial sequence. For the second case where the infeasible target configuration coincides with the entry point of the cluster (i.e. the first task point to be reached in the sub-task), the algorithm must additionally assign a new entry point to enable the robot to advance through the partial plan. This new entry point is chosen as the closest configuration within the cluster to the current robot configuration. The algorithm then resumes with the generation of a new partial plan as normal.

These two routines are summarised in Algorithms 6 and 7, respectively.

Let us now turn our attention to the trajectory re-planning component of the re-evaluation procedures. While the target configuration is not in collision with $O(t)$, the algorithm checks for collision along the planned trajectory. If no collision is detected, the trajectory is sent to the robot for execution. Where collision at any point along the trajectory is found, a new trajectory is obtained by performing a motion planning query[1] with the updated set of obstacles $O(t)$.

[1]For the implementation of d-Cluster-RTSP I use the Bi-RRT algorithm from OpenRAVE. However, in practice any fast motion planner that supports C-space planning for manipulators

Algorithm 7 subtaskReplanning

Input: Task point p_i, Assigned configuration q'_i, set of cluster centroids X, current partial plan Γ, current entry configuration q_{entry} and unreachable list L

Output: New partial plan Γ' and unreachable list L

if p_i was entry point of sub-task **then**
  $q'_{entry} \leftarrow getNewEntryConfiguration(\Gamma)$
else
  $q'_{entry} \leftarrow q_{entry}$
end if
if q'_i is $\varnothing$ **then**
  $L \leftarrow append(L, q'_i)$
  return $\Gamma' \leftarrow repairPartialPlan(\Gamma, q'_{entry})$ and L
else
  for all $x_c \in X$ **do**
    $f(x_c) = EuclideanDist(q'_i, x_c)$
  end for
  $x'_c \leftarrow \arg\min_{x_c \in X}(f(x_c))$
  Assign q'_i to cluster corresponding to x'_c
  $X \leftarrow updateClusterCentroids(X, q'_i)$
  if x'_c corresponds to current sub-task **then**
    return $\Gamma' \leftarrow recomputePartialPlan(\Gamma, q'_{entry})$ and L
  else
    return $\Gamma' \leftarrow repairPartialPlan(\Gamma, q'_{entry})$ and L
  end if
end if

These evaluations are recursively performed as shown in Fig. 7.2 until the current target configuration has been reached or updates have been made to the partial plan. Note that in instances where the robot has begun executing a trajectory, any detected collision along the trajectory or at the target configuration triggers an immediate halt command that stops the robot's motion safely. Any subsequent motion planning queries then use the robot's current configuration as the start configuration (i.e. when repairing the partial plan or re-planning the current trajectory).

With reference to the five requirements described at the beginning of this section, let us review the capabilities of the d-Cluster-RTSP for solving DRT-

can be used according to the requirements of the application. For example, if real-time dynamic motion planning is required, one could implement the dynamic roadmaps method studied in Chapter 3, though this carries its own limitations as discussed in the chapter.

SPs. Firstly, the algorithm explicitly addresses the reassignment of feasible IK solutions for task points that become unreachable from the originally assigned configuration. Task points that have become entirely unreachable are removed from their original sub-tasks and stored in an unreachable list, which allows for the task point to be revisited in a later instance. During the execution of planned motions, the algorithm continuously evaluates the feasibility of the current trajectory and halts the robot when collision is detected. Re-planning is performed with an updated obstacle set to obtain a new collision-free trajectory. In terms of tracking the optimal configuration sequence $S_C^*(t)$, the d-Cluster-RTSP provides the following behaviour. Since we assume that the new best solution always lies close to the previous best solution, it is implicitly assumed that the high-level inter-cluster visiting sequence always corresponds to the near-optimal solution (a property of the original algorithm as discussed in the previous chapter). Thus by adaptively re-allocating a task point (according to its assigned configuration) to the best cluster from among the original set of clusters, the overall solution executed by the robot is presumably always near the true optimal solution of the DRTSP at any given time t.

Finally, I wish to note the following with regards to the planning efficiency of the algorithm. The d-Cluster-RTSP has been carefully constructed to delay the evaluation of the validity of target configurations and trajectories until immediately before they are executed (as well as during execution) to avoid consistently re-planning all future actions due to the generally high computational cost of solving an RTSP (this applies even within the level of sub-tasks). As a result of this, any single sub-problem that is considered by the algorithm is generally much smaller than the complete problem considered as a whole. This allows much shorter planning times for generating a short-term partial plan and enables near real-time adaptiveness to dynamic changes during execution.

In the next section, I report quantitative results that characterize the computational performance of both the p-Cluster-RTSP and the d-Cluster-RTSP for trials conducted in simulation.

7.4 Simulation-Based Evaluation

Two experiments were conducted to evaluate the performance of the new variants of Cluster-RTSP introduced in the previous section. The first experiment is

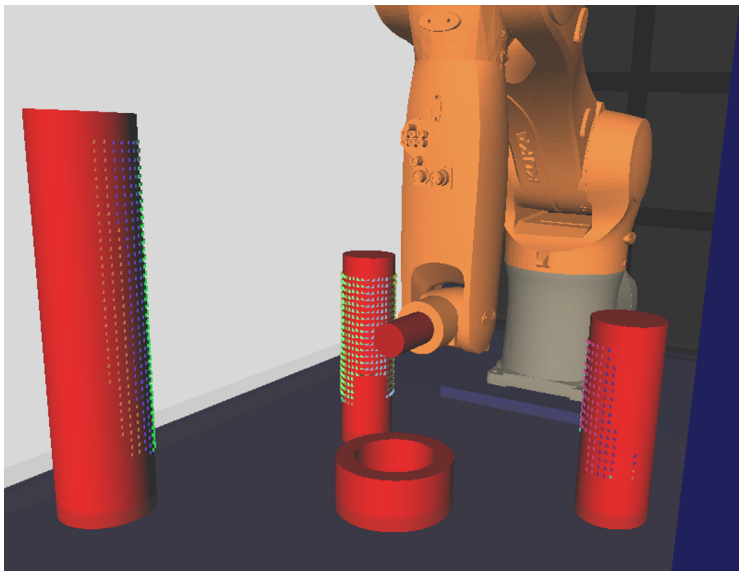


Figure 7.3: Simulation environment for the pipe scanning task.

based on the pipe scanning task described in Section 6.4 and is used to quantify the benefits of p-Cluster-RTSP for static RTSPs. The second experiment was developed to quantify the dynamic re-planning performance of the d-Cluster-RTSP in terms of planning speed. It involves a dynamic setup consisting of a manually controlled quadrotor that shares the same environment as the robotic manipulator. The manipulator is required to complete an RTSP-based activity while avoiding collision with the dynamically-moving quadrotor.

Both sets of simulations were conducted on a Linux-based system running an Intel® Xeon® CPU E3-1270 v3 (3.50 GHz) with 32 GB RAM and an NVIDIA Quadro K2000 graphics card. In all trials conducted, the minimum and maximum number of clusters was set to $K_{min} = 5$ and $K_{max} = 20$, respectively. All other algorithm parameters listed previously in Table 6.3 were preserved in these trials.

7.4.1 Partial Planning

The performance of the p-Cluster-RTSP was evaluated on a simulation of a pipe scanning task consisting of four pipes located across the frontal workspace of the robot. Reachable task points were equally distributed across the outer surfaces of these pipes as illustrated in Fig. 7.3. Here four sets of trials were conducted with the number of task points in the set P_n varied between 425, 645, 948 and 1499.

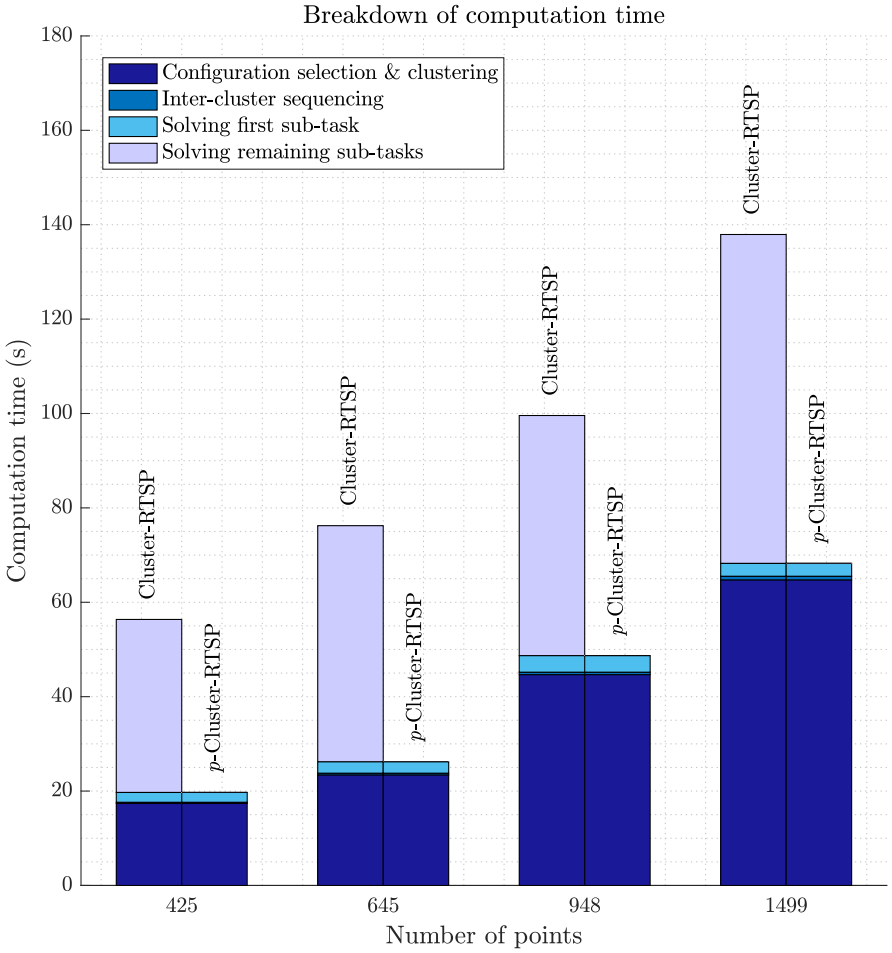


Figure 7.4: Breakdown of CPU time for standard vs partial planning.

To quantify the achievable reduction in pre-execution planning time (equivalent to the amount of time the robot spends in idle mode for *online* planning), the computation time required to obtain an executable partial plan for the first sub-task was compared against the time required by the standard Cluster-RTSP algorithm to fully solve the complete RTSP (recall that the Cluster-RTSP only returns an executable plan when the final solution has been obtained). Fig. 7.4 reports the computation times recorded for each of the four trials solved using the Cluster-RTSP and the p-Cluster-RTSP. These are broken down into the computation times required for: (i) configuration selection and clustering, which encapsulates steps 1-3 of the Cluster-RTSP (see Section 6.2.2), (ii) inter-clustering sequencing, (iii) solving the first sub-task (i.e. the computation time for solving the intra-cluster sequencing problem and computing subsequent trajectories) and (iv) solving all other sub-tasks (applicable only for the standard Cluster-RTSP).

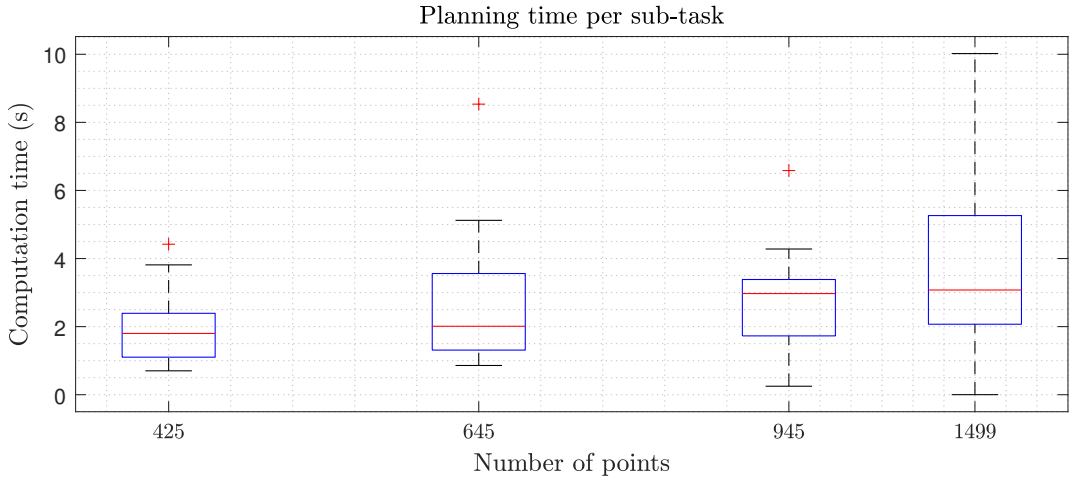


Figure 7.5: Planning time for solving individual sub-tasks.

Observe that in all trials, partial planning was able to reduce the pre-execution planning time by more than 50%. For smaller problems where less time was required for configuration selection and clustering, this reduction reached approximately 65%. This is a significant speed-up compared to the standard algorithm. Taking into account that the original Cluster-RTSP already achieves much shorter planning times than other existing methods in literature, the p-Cluster-RTSP sets a new bar for the achievable planning speed in solving static RTSPs.

Let us now consider the planning-during-execution behaviour of the p-Cluster-RTSP, which is used to obtain partial plans for subsequent sub-tasks during the execution of preceding sub-tasks. It is desirable for the computation time required to solve a single sub-task to be less than the execution time required to complete a sub-task to enable continuous execution without intermediate idle time. However, since clustering does not, in general, provide equally partitioned points across clusters, a degree of variability exists in the planning and execution times across sub-tasks. Fig. 7.5 reports the computation time required to obtain a complete partial plan for each sub-task when solving the pipe scanning problem, where the distributions across sub-tasks are shown as box plots. Similarly, Fig. 7.6 reports the distributions of the resulting execution times for each sub-task.

Notice that, on average, the planning time required to compute a partial plan is significantly lower than the average time required for completing a sub-task, which generally satisfies the requirements for planning to be less than the execution time. However, it is important to note that the maximum planning times observed (corresponding to sub-tasks with more task points) exceed the minimum

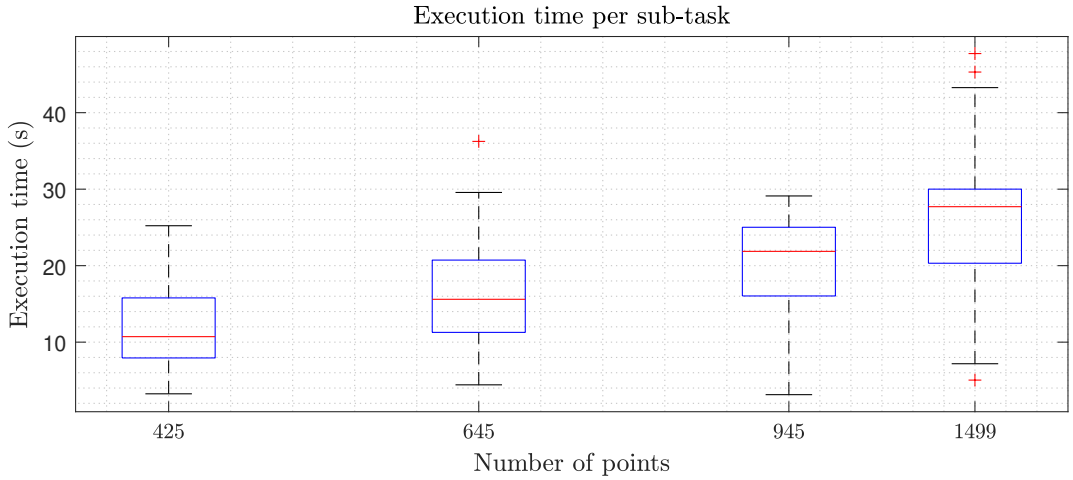


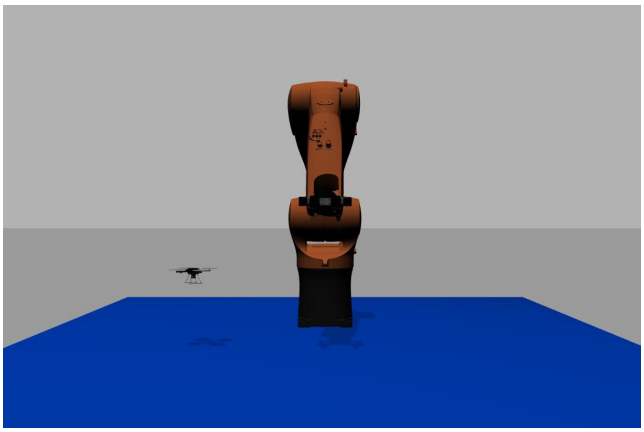
Figure 7.6: Execution time of individual sub-tasks.

execution times (associated to sub-tasks with fewer task points) in all four trials. In other words, it is possible that the robot completes the execution of a preceding sub-task while the algorithm is still planning for the next sub-task. In these cases the robot must remain in idle mode while waiting for the next set of instructions. Indeed this duration does not exceed more than a few seconds. If we consider the complete process cycle time consisting of both planning and execution, the p-Cluster-RTSP algorithm still provides substantially higher efficiency compared to the Cluster-RTSP due to the large reduction in pre-execution planning time. Nevertheless, the frequency and likelihood of the robot entering idle mode during execution is low as the algorithm continues to solve each sub-task in sequence without waiting for the execution of a previous sub-task to complete. Since the average planning time is well below the average execution time, the net difference between planning and execution generally offsets any individual instances of extended planning time.

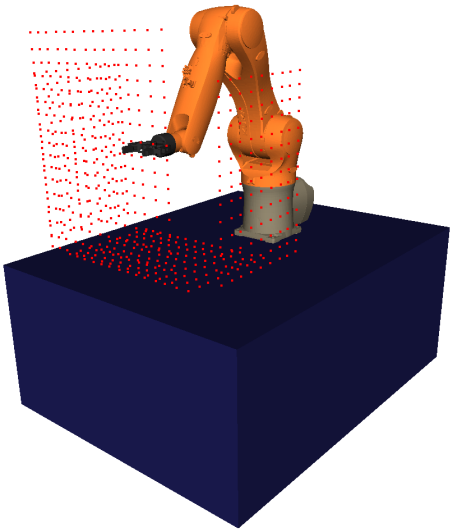
7.4.2 Dynamic Re-planning

7.4.2.1 Evaluating Planning Efficiency

To evaluate the planning efficiency of the d-Cluster-RTSP, a DRTSP simulation environment developed on the open-source Gazebo simulator [194] was set up as shown in Fig. 7.7a. It consists of the 6-DoF KUKA KR6 R900 sixx industrial manipulator fixed on top of a surface in an open space. Commands were sent to the robot through ROS [167]. A quadrotor was free to fly within the same space



(a)



(b)

Figure 7.7: Simulation environment for the evaluation of d-Cluster-RTSP, (a) Gazebo environment consisting of the KUKA manipulator and a quadrotor in an open space, (b) the distribution of task points that form the sequencing problem.

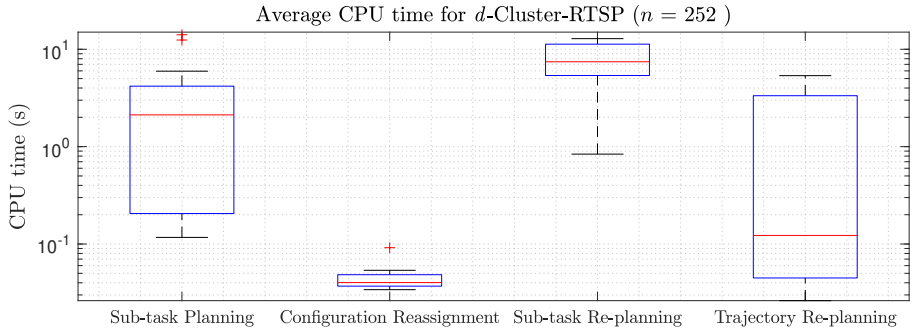
as the manipulator and was controlled manually through the ROS package *teleop twist keyboard* [195]. The manipulator was required to visit a set of task points distributed across the planes shown in Fig. 7.7b. During execution, the quadrotor was dynamically flown to interfere with the manipulator's task, triggering the three re-planning routines: configuration reassignment, sub-task re-planning and trajectory re-planning. With the maximum velocity and acceleration limits set to 50% of the rate maximum of the manipulator, four trials were conducted with $n = \{252, 363, 616, 972\}$. Note that to emulate the unpredictable nature of dynamic environments, the quadrotor was flown randomly along different flight

paths in each of the four trials.

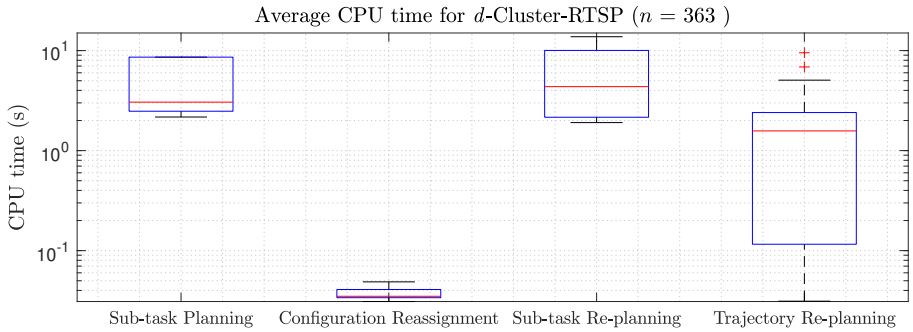
Fig. 7.8 reports the computation times (plotted in log scale) for the main planning procedures in the d-Cluster-RTSP for each of the four trials. As multiple instances of each procedure were called in a single trial, the results are shown as a box plot that captures the spread of the computation times recorded for each procedure.

In these trials, all dynamic re-planning routines were completed within 10 seconds of computation time. The configuration reassignment routine in particular consistently consumed less than 0.1 seconds, which falls well within the threshold for the re-planning procedure to be considered real-time. Importantly, the computation times for each procedure were consistent across all four trials, meaning that the size of the task set P_n had no influence on the computation times of the re-planning procedures. This is unsurprising, since these procedures apply only to a subset of task points that are contained within a single sub-task. In other words, the efficiencies of these procedures are influenced by the size of a **sub-task**, which is correlated with the number of clusters k formed during the configuration clustering procedure common to all variants of the Cluster-RTSP (i.e. the number of sub-tasks). For larger k values the number of task points within each sub-task reduces, leading to shorter computation times for sub-task (re-)planning.

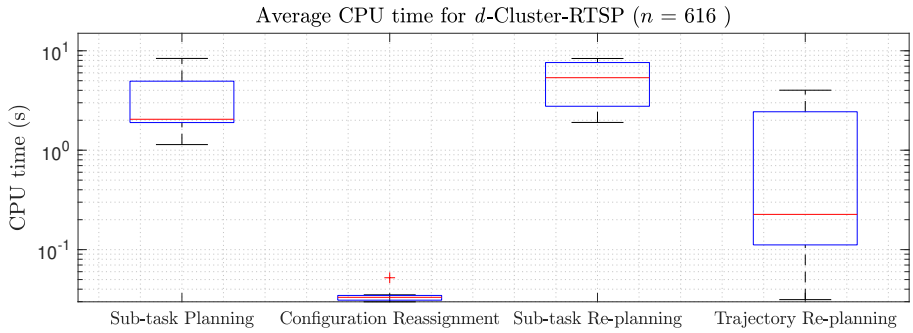
The greatest variability in planning time corresponds to trajectory re-planning. This variability is directly correlated to the complexity of the motion planning query (i.e. the spatial constraints applied to the robot). In this experiment, the complexity was determined by the relative pose of the quadrotor. If the quadrotor lay very close to the target configuration or to the joints of the robot, the motion planner required significantly more time to find a collision-free trajectory. Seeking an efficient approach to solving the motion planning problem subject to difficult spatial constraints has been the primary focus of much research for many years and still remains a challenge for applications that demand fast motion planning. While the dynamic roadmap method investigated in Chapter 3 could satisfy the real-time planning requirements for dynamic applications, it would nevertheless fail when the quadrotor lies close to the target configuration or the robot as the workspace discretization mechanism could make the start or goal configuration appear in collision with the quadrotor from the perspective of the planner (the discretization causes an unintended inflation of free-form obstacles).



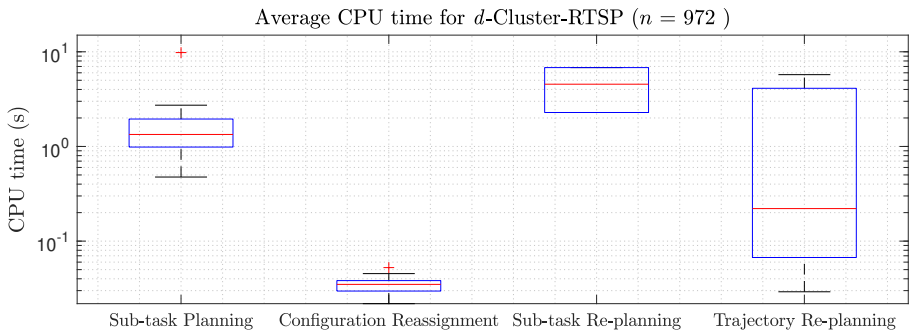
(a)



(b)



(c)



(d)

Figure 7.8: Breakdown of computation time for d-Cluster-RTSP re-evaluation routines.

I do not address this concern further as the d-Cluster-RTSP has been developed with modularity in mind, enabling the use of alternative motion planners as more effective methods are identified and developed.

One particularly interesting observation across all four trials is that sub-task re-planning requires more computation time than the initial instances of sub-task planning. This may at first appear counter-intuitive, as we would expect that as each task point in a sub-task is achieved, the time required to re-plan the remaining points in the sub-task would fall. However, recall that sub-task re-planning involves re-computing the trajectories to update and repair partial plans. As discussed above, extra complexity is introduced into motion planning when the quadrotor invades the space of the manipulator. Since a call to sub-task re-planning implies that an obstacle has interfered with the robot, the complexity of motion planning is generally higher for sub-task re-planning than for the initial calls to sub-task planning.

7.4.2.2 Evaluating Solution Quality

To quantify the quality of solutions obtained using the dynamic variant of the Cluster-RTSP, a comparison was made between the d-Cluster-RTSP and the standard Cluster-RTSP for a number of static planning instances. Planning problems were obtained by taking "snapshots" of the state of the task and the latest partial plan periodically during the dynamic planning trials described in Section 7.4.2.1. These were recorded for use as standalone planning problems solved offline to determine the quality of the solutions obtained using each method. The remaining unvisited task points were used as a fresh input into the standard Cluster-RTSP algorithm to compute a task sequence from scratch, while the d-Cluster-RTSP reused the existing plan to solve the problem. Here re-planning procedures were applied as necessary to repair originally assigned configurations that collided with the quadrotor (which was treated as a static obstacle in each planning instance). Since the Cluster-RTSP algorithm has already been carefully evaluated in Chapter 6 to benchmark its performance in finding high quality solutions, let us consider the quality of its solutions as a reference for the optimal solution for the evaluation of d-Cluster-RTSP. Like in Chapter 6, the task execution time is used to measure the quality of solutions.

Fig. 7.9 reports the task execution time of solutions obtained using Cluster-RTSP and d-Cluster-RTSP for 12 planning instances that were randomly recorded

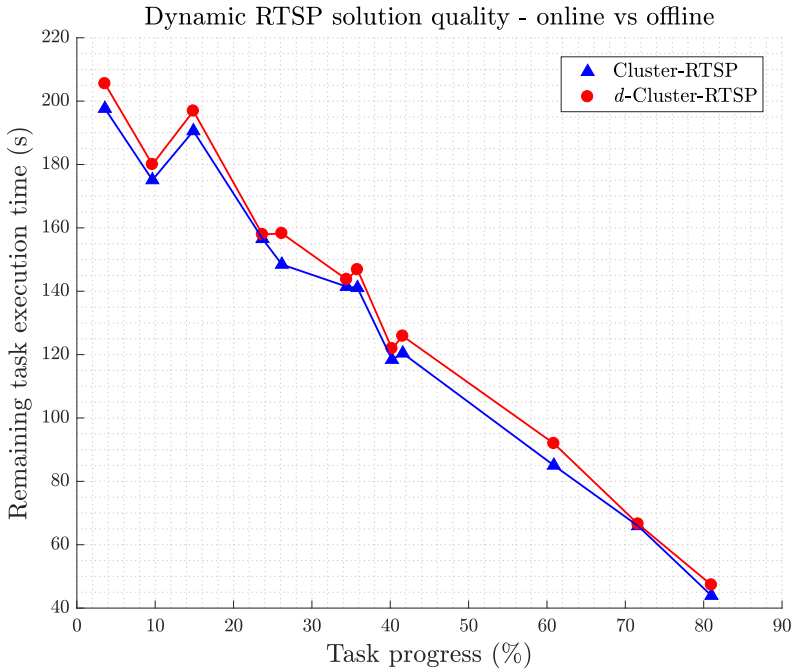


Figure 7.9: Quality of solutions obtained using Cluster-RTSP and d-Cluster-RTSP for static instances across a dynamic task.

during the DRTSP trial with $n = 363$. Since the number of unvisited task points decreased as the original task progressed, the planning instances are represented by a task progress percentage value indicating the time along the original task at which the planning instances were recorded.

It is clear that the solutions of the d-Cluster-RTSP do not diverge from the optimal solution across all planning instances. While the task execution time of these solutions never fall below that of the optimal solutions (that is, the performance of the d-Cluster-RTSP can only be as good as Cluster-RTSP even in the best case), the differences in solution quality never exceeded more than 8.1%, while in some cases the solution of the d-Cluster-RTSP was as close as 0.9% from the optimal solution. This remains true even at 80% task progress, where a large number of changes have taken place, showcasing the capability of the d-Cluster-RTSP algorithm to track the optimal solution under dynamic conditions.

7.5 Discussion

The DRTSP is a class of problems specifically aimed at addresses applications that involve dynamic considerations. Given that many applications in the real world are plagued by uncertainty, methods that can adequately address DRTSPs are, in general, more effective (or at least more safe) for practical implementation. As noted in the related work on DTSPs, previous authors have highlighted the necessary considerations for the trade-off between solution quality and planning efficiency when addressing a DTSP. An interesting behaviour of the d-Cluster-RTSP is that it reverts to the behaviour of the p-Cluster-RTSP when no changes are observed during execution. Since both variants of the algorithm retains the same initial steps to compute the inter-cluster visiting sequence (this includes steps 1-3 of the Cluster-RTSP described in Section 6.2.2), in static environments the quality of the executed plan resulting from both variants is identical to the solution that would be obtained using the standard Cluster-RTSP algorithm, which has already been shown to be near-optimal.

In dynamic scenarios, the d-Cluster-RTSP takes on the assumption that the new best solution is close to the previous best solution after a change in the problem parameters. This implies that for large sets of task points and small changes to the set $P_n(t)$, the optimal inter-cluster visiting order remains unchanged (and therefore the clustering of configurations do not need to be updated). Under this assumption, the algorithm always preserves the original order of clusters and (re-)allocates any new or dynamically-affected task points to the most fitting cluster in order to maintain the optimality of the overall solution (re-planning of sub-tasks ensures that the local configuration sequence is always optimal). Thus when this assumption holds true, the d-Cluster-RTSP continues to return a near-optimal solution. This was observed in the experiment reported in Section 7.4.2.2, where we saw that the algorithm successfully tracked the optimal solution across the full duration of the task (the deviation of the algorithm's solution from the optimal was as low as 0.9% and never exceeded 8.1%). Crucially, the d-Cluster-RTSP was able to achieve this degree of optimality while requiring no more than 10 seconds of idle planning time to repair a task sequence. Until now, no other existing method in literature has provided support for re-planning in RTSPs. Other methods must re-plan from scratch when changes occur, requiring from minutes [135] to hours [136] of computation time depending upon the

method used. With the d-Cluster-RTSP, updates to the executed plan could be achieved consistently within 10 seconds while the quality of the overall solution is maintained throughout the task. Even though a planning time of 10 seconds is still notably far off from the threshold I defined in Chapter 3 for real-time performance ($<$ 180 ms), the algorithm presented in this chapter is a significant step towards enabling online, adaptive planning for robotic task sequencing applications.

One limitation of the above assumption made in this work, however, is that it does not necessarily hold true when the size of the set $P_n(t)$ is small, or when the scale of dynamic changes in the problem parameters is large. In the first case, changes that affect a subset of $P_n(t)$ is more likely to have an impact on the optimality of the cluster visiting order and indeed the set of clusters itself. In the latter case, large changes over time would dramatically alter the landscape of the RTSP, which would result in the true optimal solution being much further away from the original best solution. For example, if task points are frequently added and removed from $P_n(t)$ while spatial constraints are modified concurrently, the latest RTSP may not resemble the original RTSP at all. In these instances the d-Cluster-RTSP would fail to provide a near-optimal solution (though the executed plan would remain collision-free).

Additionally, unlike the methods reviewed for DTSP, the d-Cluster-RTSP algorithm does not return a full solution to the overall RTSP task when a plan is requested. Instead, it waits for the completion of a sub-task on execution before progressing to planning for the next sub-task. This manifests into idle time when transitioning between sub-tasks. For applications where the frequency and scale of changes are very small (e.g. for tasks that are generally static but carries a degree of uncertainty within the problem parameters), one could alleviate this behaviour by solving for a complete plan as per the original Cluster-RTSP, and conduct the re-evaluation routines of the d-Cluster-RTSP as described. When transitioning between sub-tasks, a simple check could be performed to determine if any changes had been made to the sub-task (i.e. had there been any task points added or removed from the sub-task). Appropriate re-planning/repairing of the partial plan would be performed only in instances where changes were found.

Finally, I wish to note that this preliminary study has only considered the DRTSP from the perspective of planning. To successfully implement an adaptive system in the real world for DRTSP applications would require special consid-

erations for the real-time perception of the environment and adaptive control of the robot to safely and reliably respond to these changes. Indeed this is a limitation of this preliminary investigation, and thus evaluations have so far been confined to a simulation environment. A cross-disciplinary approach, like the case study presented in Chapter 3, would be necessary for further investigating the behaviours and performance of the d-Cluster-RTSP (or indeed similar algorithms) implemented in the real world for solving DRTSPs.

7.6 Summary

In this chapter I have introduced the DRTSP, a new form of the RTSP that involves dynamically changing problem parameters and, while adaptive approaches have begun to emerge for general task planning problems (as presented in Section 2.2.4.1), no existing method has specifically addressed this particularly challenging planning problem. This variant of the RTSP cannot be solved using conventional RTSP methods due to the necessity for online adaptive capabilities to maintain an optimal solution and, more importantly, avoid collision. The DRTSP shares some similarity with the DTSP, a class of TSP problems that have received growing attention in recent years. While EAs have been particularly popular for solving these problems, existing methods that have been proven successful for DTSPs cannot be adequately extended to DRTSPs due to the additional considerations required for kinematic redundancy, collision avoidance and motion planning.

To address the specific requirements of the DRTSP, I have presented two new variants of the Cluster-RTSP. The p-Cluster-RTSP was developed as an intermediate step towards realising dynamic re-planning capabilities in the context of RTSPs. Centred around the idea of partial planning, the algorithm enables a system to retrieve partial plans that can be executed standalone to accomplish a set of sub-tasks that collectively achieves the complete goal of an RTSP. By sending instructions to a robot immediately after obtaining the first partial plan and subsequently continuing to plan during execution to obtain partial plans for remaining sub-tasks, I have shown that the p-Cluster-RTSP achieves a substantial reduction (between 50%-65%, as reported in Section 7.4.1) in pre-execution planning time when compared to the original Cluster-RTSP algorithm (and indeed all other existing methods for solving static RTSPs in the literature).

The d-Cluster-RTSP integrates the concept of partial planning with a number of online re-planning routines that enable the re-evaluation of feasibility for configurations assigned to task points and existing planned trajectories. A configuration re-assignment routine enables the re-assignment of a feasible configuration to task points that can no longer be reached through the previously assigned configuration. When this occurs, a sub-task re-planning routine re-allocates the task point to the sub-task that minimises the cost for visiting the task point and updates the partial plan for the current sub-task. Finally, potential collisions along previously planned trajectories are handled through a trajectory re-planning procedure, which evaluates the validity of trajectories during execution and performs motion planning queries when an upcoming collision is detected. Experimental evaluations show that the algorithm is able to maintain a high quality solution within 0.9%-8.1% of the optimal in dynamic environments (Section 7.4.2.2) while requiring no more than 10 seconds of idle planning time at any instance of re-planning (Section 7.4.2.1).

To the best of my knowledge, the work presented in this chapter is the first to consider the DRTSP. The p-Cluster-RTSP pushes the boundaries of achievable planning efficiency for solving static RTSPs, while the d-Cluster-RTSP is the first algorithm to demonstrate the potential for online planning in RTSP applications. Thus in this work, the quantitative evaluations of both algorithms also provide a resource for comparison and benckmarking for future developments that seek to extend the state-of-the-art in RTSPs towards dynamic task sequencing.

Chapter 8

Conclusion

The fast growing adoption of modern robotics has brought a plethora of new challenges to the development of robotic technologies. As robots are assigned increasingly complex tasks in the real-world, there has never been a greater drive for the development of more adaptive robotic systems to adequately cope with the unavoidable presence of uncertainty, dynamics, variability and unpredictability in the real-world beyond the safe enclosure of carefully design cells, fixtures and workstations. Emerging applications for robotics are beginning to place greater emphasis on online planning capabilities to overcome the effects of dynamics for both safety and productivity reasons. While existing research has led to the development of an extensive range of offline planning methods that are effective in addressing the full spectrum of static robotic planning problems, there is still much to be desired in the dynamic counterpart of these classes of problems.

In this concluding chapter, I first discuss the research findings from each chapter collectively in the broader context of adaptive planning and then I summarise the contributions to knowledge presented in this thesis. Following this, I give a discussion on the limitations of proposed methods and close with my views on the future perspectives of this research.

8.1 Key Research Findings

This thesis has presented investigations on two different planning domains, where the problems addressed for each study involve problem-specific considerations that uniquely define one from the other. Hence the methods developed to enable adaptive planning for one domain do not transfer directly to the other domain.

Nevertheless, a number of common research findings have been observed across these two investigations, as well as the literature review and case study reported in Chapters 2 and 3, respectively, that relate to the original aim of this research.

Let us return to the four research questions introduced in Section 1.2.3, which this research had initially set out to address.

What are the necessary considerations for planning in dynamic environments?

In Chapter 2 we observed that in all methods developed for planning in dynamic environments, retaining previous knowledge in some form about the task, the search space or the environment was necessary to achieve fast planning performance. Multi-query motion planning algorithms such as the DRM, for example, achieved this by generating a roadmap across the entire search space to map discretised regions in the task space to the feasibility of individual configurations in the C-space. Dynamic RRT variants on the other hand, dynamically updated search trees by trimming invalid branches and adding new nodes as new obstacle information were perceived. Machine learning algorithms for both task planning and motion planning also retained previous knowledge through learning, enabling the planner to recall past solutions to quickly generate new solutions.

These findings were again observed in the case study on dynamic motion planning, where the DRM algorithm was applied to a dynamic pick and place task. We found that the performance of the algorithm to respond in real-time was directly coupled to the granularity of the workspace discretisation performed in the offline pre-processing phase. This offline planning stage was essential for generating knowledge about the environment to enable the algorithm to successfully compute a path to any new planning problem online. In this research, I stated that this offline planning stage is often undesirable for rapid deployment of path planning algorithms, hence the concept of retaining knowledge had to be transferred to any algorithms developed and used in this work. This consideration appeared in Chapter 5, where dynamic re-planning routines heavily relied upon reusing all valid portions of the search trees obtained by the Multi-T-RRT* algorithm. Unlike the DRM method, where the original roadmap of the C-space is preserved between planning instances, the structure of the search trees in the Multi-T-RRT* change dynamically, retaining any changes made to the tree at a given instance in time. Likewise, in Chapter 7, the top-level ordering of sub-tasks

and assigned configurations for a given task point were retained in subsequent re-planning instances, reducing the number of operations that were required to generate a new solution.

This alone, however, is insufficient for successful planning in dynamic environments. Though previous knowledge can substantially reduce the amount of work needed to compute a solution, often some parts of this knowledge no longer remains valid in the new planning instance. Hence an efficient approach for assessing the validity of plans and individual configurations online is also required. In the simplified C-space for MWRs, where the C-space directly maps to the 2 dimensional task space, it is generally simple to determine the status of collision for any given robot configuration when treating the robot as a point body. The problem becomes much more difficult for general C-spaces due to the non-trivial task of representing obstacles in C-space. The DRM method used in the case study overcame this through the pre-processing phase, which determined the mapping between these two spaces. For methods that avoid the use of pre-processing, iteratively checking each configuration for collision is impractical as this becomes too time-consuming given that it must be performed every time a new update is observed. In Chapter 7, I overcame this by limiting the collision check to the points being visited immediately, and ignoring all subsequent configurations until they were reached. This provided the added bonus of avoiding unnecessary re-planning for tasks that lay far ahead in a plan as the environment would once again change when future actions were finally reached. This resembles the strategy for online re-planning used by the authors in [146], where some computations for future actions were delayed until immediately before their execution.

What are the interactions between the goals of minimising plan cost and maximising planning efficiency when solving a task and motion planning problem?

Looking at the current state-of-the-art, the problem of finding an **optimal** solution to a planning problem by itself is not considered challenging. For example, in the ITPP one could in theory perform individual motion planning queries exhaustively to compute the costs of motion between all landmarks, and apply an optimal task planner such as the Metric-FF to obtain a high quality task plan. As for the RTSP, a number of different techniques such as the genetic algorithm, GLKH and set covering problem have been demonstrated successfully for solving

RTSPs. Clearly, all of these methods are adequate in finding an optimal solution to the planning problem. However, it is when we begin to consider planning **efficiency**, which is a key consideration for adaptive robotics, that we begin to encounter a more challenging problem. In fact, when the demand for planning efficiency approach real-time requirements, the problem becomes what we refer to as a dynamic planning problem, which can be considered the most challenging variant of planning problems to solve.

Throughout the evaluations reported in this thesis, we consistently found that the choice of a planner or solver for handling a sub-component of the planning problem largely affected the balance between solution quality and planning efficiency. Recall in Chapter 4 the comparison between the LPG-*td* and Metric-FF planners for solving the PDDL problem. While LPG-*td* excelled in finding a feasible solution quickly, it generally did not provide the best solution when compared to Metric-FF. On the other hand, Metric-FF required significant planning time even for problems involving relatively few goals. In Chapter 5, we observed that by adjusting the value of the anytime bound constant η_a in the DA-TPP framework, we could either prioritise a faster convergence rate or better solution quality, but each at the cost of the other. In Chapter 6, the benchmarking of TSP solvers for the Cluster-RTSP showed that the CTSP-2-Opt often found solutions that were poorer than those obtained by other compared methods, but at the same time the algorithm provided high planning efficiency. Finally, when benchmarking the behaviour of the Cluster-RTSP for different numbers of clusters k, we found that a single cluster consistently achieved the best quality solution, but it required significantly longer computation time for large task sets. These findings agree with the observations made in the DTSP literature that, generally, somewhat sub-optimal solutions should be accepted as a compromise for higher planning efficiency (refer to Section 7.2). In other words, it is generally not possible to obtain a truly optimal solution when efficiency is prioritised. Thus practitioners must carefully consider the requirements of an application and choose the most appropriate planner according to the planning task when seeking to implement planning algorithms that support adaptive planning behaviours.

Interestingly, however, the evaluations in this thesis suggest that the decrease in solution quality is generally marginal when compared with the significant gain in computationally efficiency. For example, when we compared the LPG-*td* to the Metric-FF planner (Section 4.5.2), we found that although the LPG-*td* produced

solutions that were up to 10% worse than the Metric-FF, it reduced the planning time by 87.3% for a problem involving 7 landmarks and up to 99.9% for a problem involving 9 landmarks. Similarly, when benchmarking the TSP solvers in Section 6.3.1, though CTSP-2-Opt approach found solutions up to 10% worse than the best performing solver, it consistently solved most problems in less than 1 second. This observation was again evident in Chapter 7 when evaluating the planning efficiency and solution quality of the d-Cluster-RTSP. I showed that the algorithm was capable of adapting a solution within 10 seconds of computation time (as opposed to re-planning from scratch, which would require several minutes of computation time), yet was able to maintain a high quality solution that lay within 8.1% from the optimal solution for the given planning instance.

Thus adaptive planning algorithms that seek to obtain optimal solutions to planning problems can generally be described as **near-optimal** relative to their offline counterparts due to the priority given to planning efficiency.

How can problems that have conventionally been solved offline as a static planning problem be addressed more efficiently to enable online planning?

As previously mentioned, multi-query planners generally involve an expensive offline pre-processing phase that limit their suitability for applications requiring rapid deployment. However, achieving optimal solutions efficiently online is difficult without the information provided by offline planning. Existing methods in literature have already demonstrated the use of certain techniques to overcome such problems. For example, Karaman et al. [61] showed how anytime motion planning could be used to quickly generate an initial feasible solution such that execution could begin promptly. The quality of the solution could then be improved during execution by continuing to search for a better plan to replace the later parts of the solution not yet due for execution. However this required a mechanism for deciding which portion of the solution should be refined. Karaman et al. dealt with this through *committed trajectories*, where the robot commits to executing the first segment of a path while the planner seeks to plan a better solution from the end of the segment to the goal. This concept of anytime planning was adopted in Chapter 5 and extended to the level of task planning such that both low-level motions and the high-level task plan were refined over time as the robot executed an initial plan. Applying this at the task level conveniently provided

intermediate goals to consider for refinement - the first action in any plan that was being executed by the robot was considered the only *committed action* that provided a new starting state for subsequent planning. All other actions were not fixed and was subject to further improvements.

A similar idea of planning during execution was also adopted in Chapter 7 for the RTSP domain, where the concept of partial planning was introduced. Rather than planning an entire solution from the outset, partial planning accelerated the time from the start of planning to the beginning of execution by generating only partial plans that satisfied a subset of goals. These partial plans were optimised taking into account the entire task, but many lower-level computations were delayed for actions that were not immediately being performed. Instead, planning was continued during execution to compute the actions required for achieving the remaining sub-tasks. This idea of dividing a task into several sub-tasks that are solved independently had also appeared in other literature, such as in [142]. Section 7.4.1 showed that the method of partitioning tasks into sub-tasks and subsequently solving the problem using a partial planning approach enabled reduction of pre-execution planning time by up to 65%, which is substantial in the quest towards extending competitive offline planning methods to online planning.

What are the practical considerations for implementing adaptive planning algorithms in physical robots?

Throughout the developments presented in this thesis, a number of practical considerations have been investigated to identify the potential limitations of proposed algorithms and the mitigation strategies to overcome them. Firstly, the key findings from Chapters 2 and 3 have shown that the memory consumption of incremental algorithms (i.e. all sampling-based algorithms) and those methods that require offline graph or roadmap generation are generally very high, which limit the use of these algorithms to high-performance machines. However, for many moving-base robots, the computing power available on-board is highly limited as many other system functionalities beyond planning must be supported by a lightweight machine. Hence in this thesis I have shown how the memory requirements of sampling-based algorithms can be bounded without impairing the capability of the algorithm to find high-quality solutions. Through the use of tree pruning in tandem with a heuristic measure for usefulness, I have demonstrated that the memory requirements of iterative sampling can be reduced by up to 97%

while achieving solutions of comparable quality.

Secondly, in Chapter 6 I investigated the problem of trajectory tracking error, showing that physical controllers in the real-world are imperfect. After quantifying the magnitude of joint and position errors for a typical inspection task using a robotic manipulator, I discussed how the Cluster-RTSP algorithm can be implemented to overcome problems caused by imprecise trajectory tracking performance. Furthermore, in Chapter 7 I discussed the performance of the proposed variants of Cluster-RTSP in relation to robot idle time. While research based in the laboratory often neglects this property, it nevertheless has important implications to the productivity of an industrial process in practice. The degree of flexibility offered by planning algorithms to quickly adapt to different scenarios is partly determined by the amount of time a robot remains stationary while the algorithm plans actions to successfully accomplish a task. For this reason, long computation times can be highly undesirable for applications requiring high flexibility. The evaluation of the p-Cluster-RTSP in Section 7.4.1 looked at how the robot idle time could be reduced prior to the execution of a plan for static task sequencing problems. Conversely, in Section 7.4.2 I assessed the d-Cluster-RTSP with respect to the robot idle time as a measure of the reactiveness of the algorithm to adapt online to dynamic changes. In other words, when assessing the performance of an adaptive algorithm for practical purposes, it is important to consider not only the computation time in relation to existing methods, but to also examine the implications of this to the reactiveness of the system for the application it is intended for.

Finally, while this thesis has primarily been concerned with the aspect of planning in adaptive robotic systems, additional considerations for fast, accurate sensing and adaptive control are also necessary when developing physical adaptive systems for the real world. As Chapter 3 has illustrated, task and motion planning requires the availability of reliable perception information. Poor quality sensing data can render the solution to a planning problem irrelevant as the real world would not correspond well with the system's knowledge of the environment. However, providing high-accuracy observations can be difficult for dynamic applications as the data-processing routines must be fast in order to support (near-)real-time robotic reactions (otherwise devised plans will always be outdated in fast-changing environments). This can be challenging for machine vision systems, where image processing is a necessary step in extracting useful

information from recorded images.

From a control perspective, it is not only crucial that the controller is able to track planned motions adequately (particularly in cluttered environments where the robot may often be required to navigate through narrow spaces), but a unique requirement of adaptive robotics is the capability of the controller to safely modify the trajectory of a robot on the fly. This particularly applies to fast moving robots such as manipulators where poor management of the robot's velocity and acceleration profiles can result in damage to joint motors when sudden and sharp changes are made to a trajectory.

8.2 Contributions to Knowledge

This thesis has presented a number of developments that advance the state-of-the-art in optimal and adaptive task and motion planning for robots in dynamic environments. In the following, I summarise the major contributions to knowledge that reflect the wide range of ongoing challenges in the field of adaptive robotics.

Limitations of Adaptive Planning Methods A literature review of existing methods for robotic task and motion planning has been presented, covering fundamental techniques and the state-of-the-art in optimal planning and adaptive planning. Summary tables have been provided to highlight the limitations of the reviewed work, which have been arranged according to the categories of the problems addressed in each work. A thorough discussion on the knowledge gaps based on an analysis of the literature has also been provided.

Optimal Path Planning for Multiple Goals A multi-tree Transition-based RRT* (Multi-T-RRT*) algorithm has been proposed for solving multi-goal path planning problems, which provides an efficient method for simultaneously computing the paths between all combinations of goals in the problem. The algorithm incorporates continuous cost spaces to account for general cost criteria, scales with the dimensions of the problem, and preserves the asymptotic optimality and probabilistic completeness guarantees of the RRT* algorithm.

Multi-goal Tree-Pruning A heuristic tree-pruning strategy has been proposed to reduce the time and memory complexity of multi-tree-based algorithms

for multi-goal path planning. This approach is able to identify candidate nodes for deletion according to a heuristic measure that determines the usefulness of tree nodes for improving existing solutions. The strategy has been shown to reduce the machine memory required for the Multi-T-RRT* algorithm by up to 97% without impairing its capability to find optimal solutions.

Integrated Task and Path Planning Building upon the Multi-T-RRT* algorithm, a strategy for integrating PDDL task planning with multi-goal path planning has been introduced. This base planner extends the advantages of the Multi-T-RRT* to the level of task planning, providing an integrated solution that scales well with the dimensionality of the problem, optimises plans according to general cost functions, and guarantees probabilistically-complete and asymptotically-optimal task plans that adequately satisfy all planning features that can be represented using PDDL.

Adaptive Task and Path Planning for Mobile Robots Through an investigation on online and adaptive task planning, an integrated task and path planning framework has been proposed to provide fast planning and dynamic re-planning capabilities for mobile robots. The framework enables anytime planning behaviours such that an initial feasible solution can be found quickly for prompt execution while improvements to the task plan is made continuously during execution. The framework supports both low-level path re-planning and high-level task re-planning to repair paths that become invalidated by new obstacles and to maintain high-quality task plans for dynamic applications.

Spatially-Constrained Robotic Task Sequencing A novel clustering-based RTSP algorithm (Cluster-RTSP) has been proposed to overcome the limitations of existing methods for solving spatially-constrained RTSPs, where solving a problem using existing methods either require substantially long computation times or result in sub-optimal task sequences. The Cluster-RTSP algorithm is able to compute near-optimal solutions for sequencing problems involving redundant robots and complex arrangements of obstacles while achieving planning times that are faster than the current state-of-the-art. The algorithm has been verified on problems involving up to 1500 task points and has been shown to always return a solution within several minutes of planning time even when solving very

complex planning problems.

Formal Definition for Dynamic RTSPs This thesis has investigated the dynamic variant of the RTSP, a unique problem involving increased complexity compared to general task planning problems. While methods exist for solving adaptive task planning problems for general robotic domains, these methods do not adequately cope with the specific requirements of robotic task sequencing. Since no existing study has investigated this particular problem, a formal definition is given to the Dynamic RTSP along with a description of key requirements for algorithms to satisfy this class of problems.

Partial Planning in RTSPs The p-Cluster-RTSP algorithm has been introduced as a new variant of the Cluster-RTSP that enables the concept of partial planning, where a complete task is divided into several sub-tasks that are individually solved progressively during execution. By solving only the first sub-task for an executable partial plan prior to execution, the pre-execution planning time can be substantially reduced by approximately 50%-65% (depending upon the complexity of the problem). The p-Cluster-RTSP provides an intermediate step towards extending offline RTSP methods to online planning.

Adaptive Planning for DRTSPs A dynamic variant of the Cluster-RTSP (d-Cluster-RTSP) has been introduced for addressing RTSPs in dynamic environments. The algorithm combines the concept of partial planning with three re-planning routines that re-assess the validity of configurations and trajectories during execution. By integrating trajectory re-planning with sub-task re-planning procedures, the algorithm is capable of maintaining near-optimal solutions under dynamic conditions by adaptively updating a previous solution within seconds. To the best of my knowledge, this work is the first to consider RTSPs in dynamic environments and the evaluations presented in this thesis provides a resource for benchmarking the performance of future developments for DRTSPs.

8.3 Limitations

This work has covered a wide range of considerations within the area of adaptive planning for dynamic environments. However, as the review of the state-of-the-

art in Chapter 2 has shown, seeking optimal solutions efficiently in dynamic environments is a hard problem. Like all the related work in the literature, the developments presented in this thesis possess a number of limitations that have not been addressed within the scope of this research.

First, Chapter 4 presented a multi-goal path planning algorithm intended to overcome the poor planning efficiency of performing multiple path planning queries between several goals in the same environment. However, the experimental results in Fig. 4.5.1 indicated that for some small subset of planning problems involving simple, uncluttered environments, the time saved by reducing the collective number of sampled configurations required to solve the path planning problems does not offset the computational overhead introduced by the multi-tree expansion procedure of the algorithm. For machines equipped with multiple processing cores, it may prove advantageous to run both standard path planning and the Multi-T-RRT* algorithm in parallel much like the CFOREST parallelization framework presented in [54] or the learning-from-experience based motion planning frameworks described in [68, 69].

In Chapter 5, the concept of anytime planning was extended to task planning in the DA-TPP framework. While this approach enables execution to begin quickly and provides continuous improvements to an executed plan, one consequence of committing to the first action of the initial solution is the high likelihood of committing to a plan that leads to a local optima at best. While this does not prevent the robot from completing the task successfully, it is possible for the robot to commit to a very poor first action that leads to sub-optimal solutions despite the continuous improvements made during planning.

Chapter 5 went on to introduce a tree pruning technique developed for multiple goals. While experiments showed that the technique enables the planner to find comparable solutions to standard tree expansion after many iterations, it is important to recognise that the rate of convergence of the algorithm to the optimal solution is slower. Furthermore, the choice of a suitable value for the maximum number of total nodes, N_{max}, across all trees is an important consideration. Too low a value can prevent the algorithm from adequately exploring the environment, causing the planner to converge to local optima, while a large value could lead to lowered memory efficiency. Determining a suitable value for a given problem may not be trivial. Machine learning may provide a solution to this problem. For example, by providing a machine with a diverse range of envi-

ronments labelled with the optimal value for N_{max}, it would be possible to train the machine to map optimal values to the complexity and scale of environments.

Finally, Chapter 5 also addressed the problem of dynamic tasks and motion planning within a 2-dimensional environment. Though the underlying Multi-T-RRT* path planning algorithm scales with the dimensionality of the problem, a limitation of the integrated framework is the difficulty to extend it to arbitrary robots possessing varying DoFs. This is due to the reliance of the re-planning procedures on being able to quickly identify all nodes in the search trees that lie in collision with new obstacles. As highlighted in Chapter 2, mapping obstacles in the Euclidean space to general C-spaces is a non-trivial task. Thus it would not be possible to extend the DA-TPP to robots where the C-space is not equivalent to the Euclidean space without a method for mapping obstacles into the C-space.

In Chapter 6, a key distinction was made between the terms near-optimal and true optimal. Indeed, a key limitation of the Cluster-RTSP algorithm is the failure to guarantee an optimal assignment of configurations to each task point considered within an RTSP. Though the evaluations have shown that this selection always produces high quality solutions across environments of varying complexity, it was evident in the experimental results presented in Section 6.3.3 that for a very high number of configurations per task point, the solutions produced by Cluster-RTSP became marginally poorer than the results obtained by RoboTSP [135]. Further investigation into the use of different metrics for defining the similarity heuristic could provide further insight into what the best performing metrics for different tasks are (recall from [172] that some metrics performed better for expansion tasks while others excelled in contraction tasks). Another limitation of the Cluster-RTSP is the inability to consider true motion costs when solving the task sequencing problem. This remains an ongoing challenge for all methods that account for kinematic redundancy and it is currently unclear whether true motion costs can be accounted for in task sequencing without using a brute force search (the problem is NP-hard).

The d-Cluster-RTSP algorithm introduced in Chapter 7 can be considered the first of its kind, as no previous study on adaptive planning has specifically addressed the problem of robotic task sequencing. Nevertheless, the proposed algorithm possesses a number of shortcomings that ultimately limit the performance of the algorithm. Firstly, each time the algorithm begins planning for a new sub-task, the robot must remain in idle mode until planning for the sub-task

is complete. This is naturally undesirable as it reduces the productivity of the robot and simply does not provide an elegant execution of tasks. As mentioned in the chapter, this could be addressed by simply solving for a complete solution prior to beginning execution, but this comes at the expense of longer pre-execution planning time. Secondly, the algorithm relies heavily upon the assumption that the new optimal solution to a problem following a change in problem variables remains close to the previous optimal solution. Unfortunately when this assumption does not hold, the algorithm fails to maintain a near-optimal task sequence. One way of potentially overcoming this problem involves periodically re-applying the X-means clustering procedure and re-solving the TSP problem at the level of configuration clusters to obtain a new ordered set of sub-tasks that reflects the updates made to configuration assignments. Thirdly, the experimental evaluations in Section 7.4.2.1 have shown that the algorithm can require up to 10 seconds of computation time to re-plan sub-tasks and trajectories. While this is substantially faster than re-planning from scratch, the efficiency of these procedures still remains far from being real-time. Consequently, the d-Cluster-RTSP would not cope with fast-changing environments where the state of the world changes at a much faster rate that the update rate of the algorithm.

Lastly, I wish to acknowledge the narrowed scope of this thesis within the context of adaptive planning. In both Chapters 5 and 7, where the development of algorithms have been extended to dynamic planning, considerations have been limited to dynamically-changing environments. However, general adaptive planning problems could additionally involve dynamically-changing tasks and goals. While the methods developed in this research has not specifically addressed this aspect of adaptive planning, it is possible to extend the presented algorithms to these scenarios. For example, in Chapter 5 I have already demonstrated the capability of the DA-TPP framework to handle new robot starting locations in global re-planning instances where the robot has partially advanced towards a landmark. A similar strategy could be used to introduce a new tree whenever a goal is dynamically added to the problem. Likewise, entire trees could be removed for dynamically removed goals. In the case of the DRTSP, re-planning procedures already provide the functionality to recompute the best configuration for an existing task point and reassign it to the most appropriate sub-task. The same procedures could be used to dynamically add new task points to the plan, whilst the removal of task points closely match the behaviour of encountering

task points that can no longer be reached. Hence, with minimal modification, the d-Cluster-RTSP could also be extended to account for dynamically-changing task points.

8.4 Concluding Remarks and Future Perspectives

This research had set out to investigate the challenges of addressing dynamic task planning problems and to develop adaptive planning algorithms that could adequately compute low-cost plans while achieving high planning efficiency in response to dynamically-changing environments. Until now, this combined problem encapsulating dynamic re-planning and optimal planning in robotics has remained mostly elusive.

Accordingly, the outputs of this research are a number of algorithms and techniques that advance the state-of-the-art by demonstrating the capability to compute near-optimal solutions under dynamic conditions while remaining computationally fast and competitive. Thorough evaluations were conducted through a number of experimental comparisons against baselines, benchmarks and the state-of-the-art to validate and quantify the benefits of these methods.

The work presented in this thesis paves the way for some interesting research directions for future development.

One interesting direction for investigation is the integration of the concepts from the DA-TPP framework with the Cluster-RTSP algorithm to enable solving RTSPs for mobile manipulators (i.e. a mobile base carrying a robot manipulator). To date, RTSPs have been limited to fixed-base manipulators, yet mobile manipulators offer the flexibility and dexterity to perform repetitive tasks on large-scale components that would otherwise be impossible to achieve with a fixed-base manipulator due to limitations with the robot's reachability. This type of problem would introduce a new challenge involving the optimal selection of base placements that enable the manipulator to perform the necessary tasks while minimising the incurred travel cost of the mobile base. One potentially viable approach to solving the mobile manipulator RTSP involves applying a sampling-based strategy in tandem with the *IKFast* module in OpenRave [171] to generate a set of valid base placements and corresponding manipulator configurations to

reach each task point. The heuristic configuration assignment procedure presented in Chapter 6 could then be extended to make an optimal selection of both base placements and manipulator configurations, giving priority to minimising configuration changes in mobile base placements were possible. Further work involving the integration of adaptive planning components, including the concepts of partial planning, anytime planning and dynamic re-planning, would form an interesting long-term study. Future developments in this direction would particularly benefit industries such as oil & gas, nuclear, construction and marine, where inspection of large vessels and building infrastructure are common.

This thesis has also provided a preliminary investigation into the feasibility and potential of addressing dynamic sequencing problems through autonomy. However, a substantial amount of development is still required to begin to realise a physical system truly capable of adapting to changes in a real-world task. As an immediate goal, one area for future work is the development of an integrated physical system, capable of achieve reliable perception, efficient planning and safe yet accurate robot control, to demonstrate this potential of adaptive robotics to adequately cope with dynamic environments. As mentioned in Chapter 7, this requires a cross-disciplinary approach with special considerations given to the real-time capabilities of the system from the perspective of sensing and control.

From the perspective of planning itself, I have introduced the d-Cluster-RTSP as a viable approach to addressing the DRTSP. However, the algorithm possesses a number of shortcomings that limit its effectiveness for some applications, including its inability to achieve near-optimal solutions in the presence of large-scale changes, the compulsory robot idle time when transitioning between sub-tasks, and the response time of up to 10 seconds for re-planning. Consequently, further research and development is necessary to overcome these limitations. This leaves us with an important research question to consider: **to what extent can we enable real-time re-planning in DRTSPs?**

A number of methods in literature have already shown promise for dynamic motion planning, yet further in-depth evaluations are necessary to understand the behaviour of these algorithms in real-world scenarios. Indeed a more comprehensive study to review existing methods in literature that in some way relate to the broader aspects of DRTSP could provide answers and inspiration for new ideas in the quest to realise true real-time adaptive robotics for DRTSPs, and indeed task planning problems in general. Machine learning methods and learning-from-

experience methods in particular have begun to show promise for their potential to quickly develop new solutions by learning from previous problems and corresponding solutions. One of their key limitations is the requirement for substantial training data, which is generally time-consuming to provide. However, conveniently, the RTSP naturally involves a large number of motion planning queries that could be used to quickly generate a large set of training data to feed into methods such as the Lightning [68] or Thunder [69] framework. In return, these algorithms could provide fast trajectory re-planning performance beyond what is currently achievable with d-Cluster-RTSP.

Research in adaptive planning for robotics is still in its infancy and many challenges have yet to be solved. Even within the study of offline task planning for manipulation, many complex problems have proven difficult to solve and in recent years significant research efforts have been directed towards finding *feasible* plans. Thus the development of adaptive task planning for general problems such as manipulation remains elusive. Interestingly, adaptive manipulation is a particular skill that humans have developed great proficiency in. In the fast growing world of modern robotics, demand for adaptive robotics to possess similar capabilities have already appeared in numerous industrial applications. All planning tasks are, after all, plagued by the inescapable effects of uncertainty, variability and dynamics in the real world beyond the safe enclosure of traditional robotic cells.

Bibliography

[1] G. Pierce, K. Burnham, L. McDonald, C. Macleod, G. Dobie, R. Summan, and D. McMahon, "Quantitative inspection of wind turbine blades using uav deployed photogrammetry," in *9th European Workshop on Structural Health Monitoring*, (in Manchester, United Kingdom), July 2018.

[2] G. Rosman, C. Choi, M. Dogar, J. W. Fisher, and D. Rus, "Task-specific sensor planning for robotic assembly tasks," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, (in Brisbane, Australia), May 2018.

[3] M. Crosby, R. P. A. Petrick, F. Rovida, and V. Krueger, "Integrating mission and task planning in an industrial robotics framework," in *27th Int. Conf. on Automated Planning and Scheduling*, 2017.

[4] "Execution summary world robotics 2017 industrial robots." Published in World Robotics 2017 Ed., 2017. International Federation of Robotics.

[5] T. Duckett, S. Pearson, S. Blackmore, and B. Grieve, "Agricultural robotics: The future of robotic agriculture." UK-RAS White papers, 2018. UK-RAS Network.

[6] J. P. Grotzinger, J. Crisp, A. R. Vasavada, R. C. Anderson, C. J. Baker, and R. B. et al., "Mars science laboratory mission and science investigation," *Space Science Reviews*, vol. 170, pp. 5–56, September 2012.

[7] R. R. Murphy, S. Tadokor, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, *Springer Handbook of Robotics*, ch. 50. Search and Rescue Robotics, pp. 1151–1173. Springer, Berlin, Heidelberg, 2008.

[8] F. Negrello, A. Settimi, D. Caporale, G. Lentini, M. Poggiani, and D. K. et al., "Humanoids at work: The WALK-MAN robot in a postearthquake

scenario," *IEEE Robotics & Automation Magazine*, vol. 25, pp. 8–22, September 2018.

[9] B. DuBose, "Drone interest soars: New partnerships aim to expand drone use for maritime asset maintenance and coating inspections," *Materials Performance*, vol. 56, no. 5, pp. 28–31, 2017.

[10] C. Gomez and D. Green, "Small unmanned airborne systems to support oil and gas pipeline monitoring and mapping," *Arabian Journal of Geosciences*, vol. 10, no. 9, pp. 202.1–202.17, 2017.

[11] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *2010 Int. Conf. on Emerging Security Technologies*, pp. 142–147, 2010.

[12] L. D. L. Barker and L. L. Whitcomb, "Preliminary survey of underwater robotic vehicle design and navigation for under-ice operations," in *IEEE Int. Conf. on Intelligent Robots and Systems*, 2016.

[13] O. Khatib, X. Yeh, G. Brantner, B. Soe, and B. K. ang S. Ganguly et al., "Ocean one: A robotic avatar for oceanic discovery," *IEEE Robotics and Automation Magazine*, vol. 23, no. 4, pp. 20–29, 2016.

[14] J. Badger, D. Gooding, K. Ensley, K. Hambuchen, and A. Thackston, "ROS in space: A case study on robonaut 2," *Studies in Computational Intelligence*, vol. 625, 2016.

[15] C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, and M. DeDonato, "What happened at the DARPA robotics challenge, and why?," tech. rep., Carnegie Mellon University, 2015.

[16] J. Wieke and S. Wrede, "Results of the survey: Failure in robotics and intelligent systems," tech. rep., Research Institute for Cognition and Robotics & Center of Excellence Cognitive Interaction Technology, Bielefeld University, Germany, 2017.

[17] "Robotic manipulation for nuclear sort and segregation." European Commission, url: `https://cordis.europa.eu/project/rcn/194336/factsheet/en`. last visited on 12 November 2019.

[18] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[19] M. J. Tsai and Y. H. Chou, "Manipulability of manipulators," *Mechanism and Machine Theory*, vol. 25, no. 5, pp. 575–585, 1990.

[20] T. Yoshikawa, "Manipulability of robotic mechanisms," *The Int. Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.

[21] J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering*, vol. 1, pp. 46–57, March 2015.

[22] B. Burns and O. Brock, "Toward optimal configuration space sampling," in *Robotics: Science and Systems*, 2005.

[23] X. Wu, Q. Li, and K. H. Heng, "An algorithm for construction of discretized configuration space obstacle and collision detection of manipulators," in *ICAR '05 Proceedings, 12th Int. Conf. on Advanced Robotics*, (in Seattle, WA, USA), July 2005.

[24] L. S. C. Pun-Cheng, M. Y. F. Tang, and I. K. L. Cheung, "Exact cell decomposition on base map features for optimal path finding," *Int. Journal of Geographical Information Science*, vol. 21, pp. 175–185, 2007.

[25] T. Arney, "An efficient solution to autonomous path planning by approximate cell decomposition," in *3rd Int. Conf. on Information and Automation for Sustainability*, (in Melbourne, VIC, Australia), December 2007.

[26] M. d Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Visibility graphs*, pp. 305–315. Springer, Berlin, Heidelberg, 1997.

[27] T. L. Pérez and M. A. Wesley, "An algorithm for planning-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[28] Y.-H. Liu and S. Arimoto, "Path planning using a tangent graph for mobile robots among polygonal and curved obstacles: Communication," *The International Journal of Robotics Research*, vol. 11, pp. 376–382, August 1992.

[29] M. H. Overmars and E. Welzl, "New methods for computing visibility graphs," in *SCG '88 Proceedings of the fourth annual Symposium on Computational Geometry*, (in Illinois, USA), June 1988.

[30] M. Babic, L. Hluchy, P. Krammer, B. Matovic, R. Kumar, and P. Kovac, "New method for constructing a visibility graph-network in 3D space and a new hybrid system of modeling," *Computing and Informatics*, vol. 36, no. 5, 2017.

[31] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, pp. 48–57, June 1989.

[32] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, December 1959.

[33] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[34] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal of Artificial Intelligence Research*, vol. 39, pp. 533–579, 2010.

[35] C. W. Warren, "A vector based approach to robot path planning," in *IEEE Int. Conf. on robotics and Automation*, (in Sacramento, CA, USA), April 1991.

[36] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape," *Algorithmica*, vol. 2, pp. 403–430, 1987.

[37] V. J. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 20, pp. 1058–1068, 1990.

[38] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Trans. on Robotics and Automation*, vol. 13, no. 6, pp. 814–822, 1997.

[39] R. A. Langer, L. S. Coelho, and G. H. C. Oliveira, "K-bug, a new bug approach for mobile robot's path planning," in *IEEE Int. Conf. on Control Applications*, (in Singapore), 2007.

[40] J. Pan, S. Chitta, and D. Manochan, "FCL: A general purpose library for collision and proximity queries," in *2012 IEEE Int. Conf. on Robotics and Automation*, (in Sait Paul, MN, USA), May 2012.

[41] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, August 1996.

[42] T. McMahon, S. Jacobs, B. Boyd, L. Tapia, and N. M. Amato, "Evaluation of the K-closest neighbor selection strategy for PRM construction," tech. rep., Texas A&M University, 2011.

[43] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.

[44] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.

[45] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *The Int. Journal of Robotics Research*, vol. 21, pp. 999–1030, December 2002.

[46] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *IEEE Int. Conf. on Robotics and Automation*, (New Orleans, LA, USA), 2004.

[47] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Computer Science Department, Iowa State University, 1998.

[48] L. Jaillet, J. corts, and T. Simon, "Transition-based RRT for path planning in continuous cost spaces," in *2008 IEEE/RSJ Int. Conf. on Intelligent Robotic Systems*, pp. 2145–2150, 2008.

[49] D. Devaurs, T. Simeon, and J. cortes, "Optimal path planning in complex cost spaces with samping-based algorithms," *IEEE Trans. on Automation Science and Engineering*, vol. 15, pp. 415–424, April 2016.

[50] J. J. Kuffner and S. M. LaValle, "RRT-connect: an efficient approach to single-query path planning," in *IEEE Int. Conf. on Robotics and Automation*, 2000.

[51] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," tech. rep., Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2013.

[52] A. Bircher, K. Alexis, U. Schwesinger, S. Omari, M. Burri, and R. Siegwart, "An incremental sampling-based approach to inspection planning: the rapidly-exploring random tree of trees," *Robotica*, vol. 35, pp. 1327–1340, June 2017.

[53] D. Devaurs, T. Simeon, and J. Cortes, "A multi-tree extension of the transition-based RRT: Application to ordering and pathfinding problems in continuous cost spaces," in *2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2991–2996, 2014.

[54] M. Otte and N. Correll, "C-FOREST: parallel shortest-path planning with super linear speedup," in *24th Int. Conf. on Automated Planning and Scheduling*, (Portsmouth, USA), June 2014.

[55] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2383–2388, 2014.

[56] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE Int. Conf. on Robotics and Automation*, pp. 1243–1248, 2006.

[57] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *IEEE Int. Conf. on Robotics and Automation*, pp. 1603–1609, 2007.

[58] M. Otte and E. Frazzoli, "RRT$^{\mathrm{X}}$: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *Int. Journal of Robotics Research*, vol. 35, pp. 797–822, June 2016.

[59] D. Ferguson and A. Stentz, "Anytime RRTs," in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (in Beijing, China), pp. 5369–5375, 2006.

[60] D. Ferguson and A. Stentz, "Anytime, dynamic planning in high-dimensional search spaces," in *2007 IEEE Int. Conf. on Robotics and Automation*, (in Roma, Italy), pp. 1310–1315, 2007.

[61] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *2011 IEEE Int. Conf. on Robotics and Automation*, (in Shanghai, China), May 2011.

[62] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Trans. on Systems, Man and Cybernetics: Systems*, vol. 43, pp. 1141–1153, September 2013.

[63] J. Jiang and J. Xin, "Path planning of a mobile robot in a free-space environment using q-learning," in *Progress in Artificial Intelligence*, pp. 133–142, 2019.

[64] Z. Liu, F. Lan, and H. Yang, "Partition Heuristic RRT Algorithm of path planning based on Q-learning," in *IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference*, (in Chengdu, China), 2019.

[65] J.-H. K. J.-J. Park and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," *Int. Journal of Control, Automation, and Systems*, vol. 5, pp. 674–680, December 2007.

[66] P. Corke, *Robotics, Vision and Control, Fundamental Algorithms in MATLAB*. Springer, second ed., 2017.

[67] R. Meyes, H. Tercan, S. Roggendorf, T. Thiele, C. Buscher, M. Obdenbusch, C. Brecher, S. Jeschke, and T. Meisen, "Motion planning for industrial robots using reinforcement learning," *Procedia CIRP*, vol. 63, pp. 107–112, 2017.

[68] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *IEEE Int. Conf. on Robotics and Automation*, (in Saint Paul, MN, USA), 2012.

[69] D. Coleman, I. A. Sucan, M. Moll, K. Okada, and N. Correll, "Experience-based planning with sparse roadmap spanners," in *IEEE Int. Conf. on Robotics and Automation*, (Seattle, WA, USA), May 2015.

[70] A. Dobson, A. Krontiris, and K. E. Bekris, *Algorithmic Foundations of Robotics X*, vol. 86 of *Springer Tracts in Advanced Robotics*, ch. Sparse Roadmap Spanners. Springer, 2013.

[71] M. F. Abdelwahed, A. E. Mohamed, and M. A. Saleh, "Solving the motion planning problem using learning experience through case-based reasoning and machine learning algorithms," *Ain Shams Engineering Journal*, vol. 11, pp. 133–142, March 2020.

[72] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.

[73] N. Achour and M. Chaalal, "Mobile robots path planning using genetic algorithms," in *7th Int. Conf. on Autonomic and Autonomous Systems*, 2011.

[74] M. Brand, M. Masuda, N. Wehner, and X.-H. Yu, "Ant colony optimization algorithm for robot path planning," in *Int. Conf. on Computer Design and Applications*, (in Qinhuangdao, China), June 2010.

[75] H.-J. Wang, Y. Fu, Z.-Q. Zhao, and Y.-J. Yue, "An improved ant colony algorithm of robot path planning for obstacle avoidance," *Journal of robotics*, 2019.

[76] X. Dai, S. Long, Z. Zhang, and D. Gong, "Mobile robot path planning based on ant colony algorithm with A* heuristic method," *Frontiers in Neurorobotics*, vol. 13, April 2019.

[77] A. Z. Nasrollahy and H. H. S. Javadi, "Using particle swarm optimization for robot path planning in dynamic environments with moving obstacles and target," in *3rd UKSim European Symposium on Computer Modeling and Simulation*, (Athens, Greece), November 2009.

[78] S. C. Brailsford, C. N. Potts, and B. M. Smith, "Constraint satisfaction problem: Algorithms and applications," *European Journal of Operational Research*, vol. 119, pp. 557–581, 1999.

[79] P. Norvig and S. J. Russell, *Artificial Intelligence: A modern approach*, ch. 6, pp. 202–233. Prentice Hall, third ed., December 2009.

[80] S. Minton, A. B. Philips, M. D. Johnston, and P. Laird, "The min-conflicts heuristic: Experimental and theoretical results," tech. rep., NASA Technical Reports Server, September 1991.

[81] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - the planning domain definition language," tech. rep., Yale Center for Computational Vision and Control, 1998.

[82] J. Hoffman and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[83] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[84] J. O. Hoffmann, "The metric-FF planning system: Translating 'ignoring delete lists' to numeric state variables," *Journal of Artificial Intelligence Research*, vol. 20, pp. 291–341, 2003.

[85] A. Gerevini and I. Serina, "LPG: a planner based on local search for planning graphs with action costs," in *6th Int. Conf. on Artificial Intelligence Planning Systems*, pp. 13–22, 2002.

[86] A. Gerevini, A. Saetti, I. Serina, and P. Toninelli, "LPG-TD: a fully automated planner for PDDL2.2 domains," in *14th Int. Conf. on Automated Planning and Scheduling*, (in Whistler, British Columbia, Canada), 2004. International Planning Competition.

[87] Y. Chen, C.-W. Hsu, and B. W. Wah, "SGPlan: Subgoal partitioning and resolution in planning," in *4th Int. Planning Competition: Int. Conf. on Automated Planning and Scheduling*, June 2004.

[88] B. W. Wah and Y. Chen, "Partitioning of temporal planning problems in mixed space using the theory of extended saddle points," in *15th IEEE Int. Conf. on Tools with Artificial Intelligence*, (in Sacramento, CA, USA), 2003.

[89] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.

[90] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[91] S. Srivastava, L. Riano, S. Russell, and P. Abbeel, "Using classical planners for tasks with continuous operators in robotics," in *Intelligent Robotic Systems - Papers from the 2013 AAAI Workshop*, pp. 85–91, 2013.

[92] L. P. Kaelbling and T. Lozano-Perez, "Hierarchical task and motion planning in the now," in *2011 IEEE Int. Conf. on Robotics and Automation*, (in Shanghai, China), pp. 1470–1477, 2011.

[93] J. Ferrer-Mestres, G. Frances, and H. Geffner, "Combined task and motion planning as classical ai planning," tech. rep., Pompeu Fabra University, 2016.

[94] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *International Journal of Robotics Research*, vol. 28, pp. 104–126, January 2009.

[95] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE Int. Conf. on Robotics and Automation*, (in Hong Kong, China), pp. 639–646, 2014.

[96] N. T. Dantum, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *Int. Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[97] N. T. Dantum, S. Chuadhuri, and L. E. Kavraki, "The task-motion kit: An open source, general-purpose task and motion-planning framework," *IEEE Robotics and Automation Magazine*, vol. 25, pp. 61–70, September 2018.

[98] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: An efficient heuristic for task and motion planning," *Algorithmic Foundations of Robotics XI*, pp. 179–195, 2015.

[99] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *Int. Journal of Robotics Research*, vol. 37, pp. 104–136, April 2018.

[100] J. Gerold, "Throughput, productivity top KPI list." Available at: `https://www.automationworld.com/products/control/article/13304138/throughput-productivity-top-kpi-list`, April 2004. Last visited: 15th December 2019.

[101] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[102] S. Alatartsev, S. Stellmacher, and F. Ortmeier, "Robotic task sequencing problem: A survey," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 279–298, November 2015.

[103] S. Srivastava, S. Kumar, R. Garg, and P. Sen, "Generalized traveling salesman problem through n sets of nodes," *CORSE Journal*, vol. 7, pp. 97–101, 1969.

[104] L. L. Abdel-Malek and Z. Li, "The application of inverse kinematics in the optimum sequencing of robot tasks," *Int. Journal of Production Research*, vol. 28, no. 1, pp. 75–90, 1990.

[105] C. Wurll, D. Henrich, and H. Worn, "Multi-goal path planning for industrial robots," in *Int. Conf. on Robotics and Application*, (in Santa Barbara, USA), October 1999.

[106] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, pp. 197–218, 1995.

[107] A. Kovacs, "Integrated task sequencing and path planning for robotic remote laser welding," *Int. Journal of Production Research*, vol. 54, pp. 1210–1224, June 2015.

[108] F. Liu, S. Liang, and X. Xian, "Optimal robot path planning for multiple goals visiting based on tailored genetic algorithm," *International Journal of Computational Intelligence and Systems*, vol. 7, pp. 1109–1122, December 2014.

[109] A. Noormohammadi-Asl, H. D. Taghirad, and A. Tamjidi, "Implementation of multi-goal motion planning under uncertainty on a mobile robot," in *5th RSI Int. Conf. on Robotics and Mechatronics*, (in Tehran, Iran), October 2017.

[110] J. Faigl, V. Vonásek, and L. Přeučil, "Visiting convex regions in a polygonal map," *Robotics and Autonomous Systems*, vol. 61, pp. 1070–1083, October 2013.

[111] Y. Zhang, S. Sreedharan, A. Kulkami, T. Chakraborti, H. H. Zhuo, and S. Kambhampati, "Plan explicability and predictability for robot task planning," in *IEEE Int. Conf. on Robotics and Automation*, (in Singapore), 2017.

[112] C. Galindo, J.-A. Fernández-Madrigal, J. González, and A. Saffiotti, "Using semantic information for improving efficiency of robot task planning," in *IEEE Int. Conf. on Robotics and Automation*, (in Rome, Italy), April 2007.

[113] P. Munoz, M. D. R. Moreno, and D. F. Barrero, "Unified framework for path-planning and task-planning for autonomous robots," *Robotics and Autonomous Systems*, vol. 82, pp. 1–14, 2016.

[114] M. Cirillo, L. Karisson, and A. Saffiotti, "Human-aware task planning for mobile robots," in *Int. Conf. on Advanced Robotics*, (in Munich, Germany), 2009.

[115] L. Karlsson, "Conditional progressive planning under uncertainty," in *17th Int. Joint Conf. on Artificial Intelligence*, pp. 431–436, 2001.

[116] S.-J. Lee, J.-M. Park, D.-H. Kim, and J.-H. Kim, "Adaptive task planner for performing home service tasks in cooperation with a human," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (in Madrid, Spain), October 2018.

[117] M. Leonetti, L. Iocchi, A. G. Cohn, and D. Nardi, "Adaptive human-aware task planning," in *Int. Conf. on Automated Planning and Scheduling*, 2019.

[118] N. Castaman, E. Tosello, and E. Pagello, "Conditional task and motion planning through an effort-based approach," in *IEEE Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots*, (in Brisbane, QLD, Australia), May 2018.

[119] I. A. Sucan and L. E. Kavraki, *Algorithmic Foundations of Robotics VIII*, vol. 57 of *Springer Tracts in Advanced Robotics*, ch. Kinodynmaic motion planning by interior-exterior cell exploration, pp. 449–464. Springer, 2009.

[120] S. Dubowsky and T. Blubaugh, "Planning time-optimal robotic manipulator motions and work places for point-to-point tasks," *IEEE Trans. on Robotics and Automation*, vol. 5, pp. 377–381, 1989.

[121] Y. Edan, T. Flash, U. Peiper, I. Shmulevich, and Y. Sarig, "Near minimum-time task planning for fruit-picking robots," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 1, pp. 48–56, 1991.

[122] L. L. Abdel-Malek and Z. Li, "Robot location for minimum cycle time," *Engineering Costs and Production Economics*, vol. 17, no. 1, pp. 29–34, 1989.

[123] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

[124] E. Kolakowska, S. F. Smith, and M. Kristiansen, "Constraint optimization model of a scheduling problem for a robotic arm in automatic systems," *Robotics and Autonomous Systems*, vol. 62, no. 2, pp. 267–280, 2014.

[125] F. Vitolo, P. Franciosa, D. Ceglarek, S. Patalano, and M. D. Martino, "A generalised multi-attribute task seqeuencing approach for robotics optical inspection system," in *II Workshop on Metrology for Industry 3.0 and IoT*, (in Naples, Italy), June 2019.

[126] P. T. Zacharia and N. Aspragathos, "Optimal robot task scheduling based on genetic algorithms," *Robotics and Computer-Integrated Manufacturing*, vol. 21, no. 1, pp. 67–79, 2005.

[127] W. Jing, J. Polden, P. Y. Tao, C. F. Goh, W. Lin, and K. Shimada, "Model-based coverage motion planning for industrial 3D shape inspection applications," in *13th IEEE Conf. on Automation Science and Engineering (CASE)*, (in Xi'an, China), August 2017.

[128] W. Jing, J. Polden, C. F. Goh, M. Rajaraman, W. Lin, and K. Shimada, "Sampling-based coverage motion planning for industrial inspection application with redundant robotic system," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (in Vancouver, BC, Canada), pp. 5211–5218, 2017.

[129] S. N. Spitz and A. A. G. Requicha, "Multiple-goals path planning for coordinate measuring machines," in *IEEE Int. Conf. on Robotics and Automation*, (in San Francisco, CA, USA), pp. 2322–2327, 2000.

[130] E. K. Xidias, P. T. Zacharia, and N. A. Aspragathos, "Time-optimal task scheduling for articulated manipulators in environments cluttered with obstacles," *Robotica*, vol. 28, no. 3, pp. 427–440, 2010.

[131] P. T. Zacharia, E. K. Xidias, and N. A. Aspragathos, "Task scheduling and motion planning for an industrial manipulator," *Robotics and Computer-Integrated Manufacturing*, vol. 29, pp. 449–462, December 2013.

[132] L. B. Gueta, R. Chiba, J. Ota, T. Ueyama, and T. Arai, "Coordinated motion control of a robot arm and a positioning table with arrangement of multiple goals," in *IEEE Int. Conf. on Robotics and Automation*, (in Pasadena, CA, USA), pp. 2252–2258, 2008.

[133] G. A. Croes, "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, pp. 791–908, December 1958.

[134] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[135] F. Suarez-Ruiz, T. S. Lembono, and Q.-C. Pham, "RoboTSP - a fast solution to the robotic task sequencing problem," in *IEEE Int. Conf. on Robotics and Automation*, (in Brisbane, QLD, Australia), pp. 1611–1616, 2018.

[136] S. L. Villumsen and M. Kristiansen, "A framework for task sequencing for redundant robotic remote laser processing equipment based on redundancy space sampling," *Procedia Manufacturing*, vol. 11, pp. 1826–1836, 2017.

[137] J. Vannoy and J. Xiao, "Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1199–1212, 2008.

[138] H. Mao and J. Xiao, "Real-time conflict resolution of task-constrained manipulator motion in unforeseen dynamic environments," *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1276–1283, 2019.

[139] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[140] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *1st Conference on Robot Learning*, (in Mountain View, USA), 2017.

[141] H. Zhou, H. Min, and Y. Lin, "An autonomous task algorithm based on behavior trees for robot," in *2nd China Symposium on Cognitive Computing and Hybrid Intelligence*, (in Xi'an, China), September 2019.

[142] H. Harman, K. Chintamani, and P. Simoens, "Robot assistance in dynamic smart environments - a hierarchical continual planning in the now framework," *Sensors*, vol. 19, no. 22, 2019.

[143] K. He, A. M. Wells, L. E. Kavraki, and M. Y. Vardi, "Efficient symbolic reactive synthesis for finite-horizon tasks," in *Int. Conf. on Robotics and Automation*, (in Montreal, QC, Canada), May 2019.

[144] Y. Jiang, F. Yang, S. Zhang, and P. Stone, "Task-motion planning with reinforcement learning for adaptable mobile service robots," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (in Macau, China), November 2019.

[145] P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G. v. Wichert, and W. Burgard, "Planning reactive manipulation in dynamic environments," in *IEEE/RSJ*

Int. Conf. on Intelligent Robots and Systems, (in Macau, China), November 2019.

[146] C. R. Garrett, C. Paxton, T. Lozano-Perez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *IEEE Int. Conf. on Robotics and Automation*, 2020.

[147] H. N. Psaraftis, "A dynamic-programming solution to the single vehicle many-to-many immediate request dial-a-ride problem," *Transport Science*, vol. 14, pp. 130–154, 1980.

[148] C. Li, M. Yang, and L. Kang, "A new approach to solving dynamic traveling salesman problems," in *Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 236–243, 2006.

[149] M. M. J. Jurjee, H. M. Sarim, N. H. A. Al-Dabbagh, and E. B. Nababan, "A multi-population harmony search algorithm for the dynamic travelling salesman problem with traffic factors," *Journal of Theoretical and Applied Information Technology*, vol. 95, no. 2, 2017.

[150] R. Tinós, "Analysis of the dynamic traveling salesman problem with weight changes," in *Latin American Congress on Computational Intelligence*, (in Curitiba, Brazil), 2015.

[151] L. Strak, R. Skinderowicz, U. Boryczka, and A. Nowakowski, "A self-adaptive discrete pso algorithm with heterogeneous parameter values for dynamic tsp," *Entropy*, vol. 21, no. 8, 2019.

[152] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors," *Applied Soft Computing*, vol. 13, pp. 4023–4037, 2013.

[153] M. Mavrovouniotis, F. M. Muller, and S. Yang, "Ant colony optimization with local search for dynamic travelling salesman problems," *IEEE Transactions on Cybernetics*, vol. 47, no. 7, pp. 1743–1756, 2017.

[154] Y.-W. Huang, C.-F. Liu, S.-F. Hu, Z.-H. Fu, and Y. Chen, "Dynamic task sequencing of manipulator by Monte Carlo tree search," in *Int. Conf. on Robotics and Biomimetics*, (in Kuala Lumpur, Malaysia), December 2018.

[155] J. Zabalza, Z. Fei, C. Wong, Y. Yan, C. Mineo, E. Yang, T. Rodden, Q.-C. Pham, and J. Ren, "Smart sensing and adaptive reasoning for enabling industrial robots with interactive human-robot capabilities in dynamic environments - a case study," *Sensors*, vol. 19, March 2019.

[156] I. Lopez-Juarez, "Skill acquisition for industrial robots: From stand-alone to distributed learning," in *IEEE Int. Conf. on Automatica*, (in Curico, Chile), October 2016.

[157] T. Kunz, U. Reiser, M. Stilman, and A. Verl, "Real-time path planning for a robot arm in changing environments," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (in Taipei, Taiwan), October 2010.

[158] D. G. Wall, J. Economou, H. Goyder, K. Knowles, P. Silson, and M. Lawrence, "Mobile robot arm trajectory generation for operation in confined environments," *Journal of Systems and Control Engineering*, vol. 229, pp. 215–234, 2015.

[159] C. Mineo, C. Wong, M. Vasilev, B. Cowan, C. MacLeod, S. G. Pierce, and E. Yang, "Interfacing toolbox for robotic arms with real-time adaptive behaviour capabilities," tech. rep., University of Strathclyde, 2019.

[160] T. Lyche and K. Morken, "Spline methods draft," tech. rep., Department of Informatics, Centre of Mathematics for Applications, University of Oslo, May 2008.

[161] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of time-optimal, jerk-limited trajectories," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (in Nice, France), September 2008.

[162] "KR QUANTEC Extra HA operating instructions." KUKA, 2013.

[163] G. R. Grice, R. Nullmeyer, and V. A. Spiker, "Human reaction time: Toward a general theory," *Journal of Experimental Psychology: General*, vol. 111, no. 1, pp. 135–153, 1982.

[164] M. C. Lin, *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California at Berkeley, 1993.

[165] F. Schwarzer, M. Saha, and J. Latombe, *Algorithmic Foundations of Robotics V*, vol. 7 of *Springer Tracts in Advanced Robotics*, ch. Exact Collision Checking of Robot Paths, pp. 25–41. Springer, 2004.

[166] F. Schwarzer, M. Saha, and J.-C. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Transactions on Robotics*, vol. 21, pp. 338–353, June 2015.

[167] A. Koubaa, ed., *Robot Operating System (ROS) - The complete reference*, vol. 4. Springer International Publishing, 2020.

[168] D. F. Elkott, H. A. Elmaraghy, and W. H. Elmaraghy, "Automatic sampling for cmm inspection planning of free-form surfaces," *Int. Journal of Production Research*, vol. 40, no. 11, pp. 2653–2676, 2002.

[169] M. P. Mali and K. H. Inamdar, "Optimization of spot welding process using digital manufacturing," *International Archive of Applied Sciences and Technology*, vol. 4, pp. 27–35, June 2013.

[170] D. Pelleg and A. Moore, "X-means: Extending K-means with efficient estimation of the number of clusters," in *Conference on Machine Learning*, (in Stanford, CA, USA), pp. 727–734, June-July 2000.

[171] R. Diankov and J. J. Kuffner, "OpenRAVE: A planning architecture for autonomous robotics," tech. rep., Robotics Institute, Pittsburgh, PA, 2008.

[172] H. J. Jeon and A. D. Dragan, "Configuration space metrics," in *IEEE/RSJ Int. Conf. on Robotics and Systems*, (in Madrid, Spain), pp. 5101–5108, 2018.

[173] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[174] M. Englert, H. Roglin, and B. Vocking, "Worst case and probabilistic analysis of the 2-opt algorithm for the tsp," *Algorithmica*, vol. 68, no. 1, pp. 190–264, 2014.

[175] J. J. A. Slootbeek, "Average-case analysis of the 2-opt heuristic for the tsp," Master's thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, Netherlands, 2017.

[176] R. E. Kass and L. Wasserman, "A reference bayesian test for nested hypotheses and its relationship to the schwasz criterion," *Journal of the American Statistical Association*, vol. 90, no. 431, pp. 928–934, 1995.

[177] B. Hancock, D. Frank, A. Foglia, and R. Yozzo, "Notes on bayesian information criterion calculation for x-means clustering." Available at: `https://github.com/bobhancock/goxmeans/blob/master/doc/BIC_notes.pdf`, 2012. last visited on 09 January 2020.

[178] J. A. Chisman, "The clustered traveling salesman problem," *Computers & Operations Research*, vol. 2, pp. 115–119, September 1975.

[179] Z. H. Ahmed, "The ordered clustered travelling salesman problem: A hybrid genetic algorithm," *The Scientific World Journal*, February 2014.

[180] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, pp. 245–248, January 2009.

[181] J. v. Leeuwen and A. A. Schoon, "Untangling a traveling salesman tour in the plane," in *Int. Workshop on Graph Theoretic Concepts in Computer Science*, pp. 87–98, 1981.

[182] P. H. Siqueira, S. Scheer, and M. T. A. Steiner, *Travelling Salesman Problem*, ch. A recurrent neural network to traveling salesman problem, pp. 135–156. IntechOpen, 2008.

[183] N. Aras, B. J. Oommen, and I. K. Altinel, "The kohonen network incorporating explicit statistics and its application to the traveling salesman problem," *Neural Networks*, vol. 12, no. 9, pp. 1273–1284, 1999.

[184] M. Budinich, "A self-organizing map neural network for the traveling salesman problem that is competitive with simulated annealing," *Neural Computation*, vol. 8, pp. 416–424, 1996.

[185] K. S. Leung, H. D. Jin, and Z. B. Xu, "An expanding sef-organising neural network for the traveling salesman problem," *Neurocomputing*, vol. 62, pp. 267–292, 2004.

[186] F. C. Vieira, A. D. D. Neto, and J. A. Costa, "An efficient approach to the travelling salesman problem using self-organizing maps," *Int. Journal of Neural Systems*, vol. 13, no. 2, pp. 59–66, 2004.

[187] E. M. Cochrane and J. E. Beasley, "The co-adaptive neural network approach to the euclidean travelling salesman problem," *Neural Networks*, vol. 16, no. 10, pp. 1499–1525, 2003.

[188] G. Reinelt, "TSPLIB - a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[189] R. Longbottom, "Whetstone benchmark history and results," tech. rep., Roy Longbottom's PC Benchmark Collection, July 2017.

[190] "Performance of various computers in computational chemistry." Computer for Science (CFS), url: `http://www.cfs.dl.ac.uk/benchmarks/compchem.html`, February 2005. last visited on 11 April 2020.

[191] "Airbus shopfloor challenge." Robohub. Available at: `https://robohub.org/tag/airbus-shopfloor-challenge/`, 2016. Last visited: 15th January 2020.

[192] C. Mineo, M. Vasilev, C. N. MacLeod, R. Su, S. G. Pierce, C. Wong, and E. Yang, "Enabling robotic adaptive behaviour capabilities for new industry 4.0 automated quality inspection paradigms," in *57th Annual British Conf. on Non-Destructive Testing*, (in Nottingham, United Kingdom), pp. 28–39, September 2018.

[193] "KR AGILUS sixx specification." KUKA, March 2015.

[194] "Gazebo robotic simulator." Open Source Robotics Foundation, url: `http://gazebosim.org/`. last visited on 31 January 2020.

[195] "Teleop twist keyboard." ROS wiki. Available at: `http://wiki.ros.org/teleop_twist_keyboard`, 2015. Last visited: 31st January 2020.

Appendix A

PDDL Domain and Problem Files

PDDL Domain Example File

Listing A.1: MWR Nagivation Domain

```
(define (domain robot-world)
 (:requirements :durative-actions :fluents :duration-inequalities)

 (:functions
  (cost ?from-waypoint ?to-waypoint)
  (total-cost)
 )

 (:predicates
  (waypoint ?waypoint)
  (robot ?robot)
  (task ?task)
  (task-at ?task ?waypoint)
  (task-done ?task)
  (at ?robot ?waypoint)
  (robot_base ?waypoint)
 )

 (:durative-action move_to_landmark
  :parameters
   (?robot
    ?task
```

```
    ?waypoint1
    ?waypoint2)

  :duration
   (= ?durations 5)

  :condition
   (and
    (at start (robot ?robot))
    (at start (task ?task))
    (at start (waypoint ?waypoint1))
    (at start (waypoint ?waypoint2))
    (at start (at ?robot ?waypoint1))
    (at start (task-at ?task ?waypoint2))
    (at start (< (cost ?waypoint1 ?waypoint2) 1000000))
   )

  :effect
   (and
    (at start (not(at ?robot ?waypoint1)))
    (at start (increase (total-cost) (cost ?waypoint1 ?waypoint2)))
    (at end (task-done ?task))
    (at end (at ?robot ?waypoint2))
   )
)

(:durative-action move_to_base
 :parameters
  (?robot
   ?waypoint1
   ?waypoint2)

 :duration
  (= ?durations 5)

 :condition
  (and
   (at start (robot ?robot))
   (at start (waypoint ?waypoint1))
   (at start (waypoint ?waypoint2))
   (at start (at ?robot ?waypoint1))
   (at start (robot_base ?waypoint2))
```

```
    (at start (< (cost ?waypoint1 ?waypoint2) 1000000))
   )

  :effect
   (and
    (at start (not(at ?robot ?waypoint1)))
    (at start (increase (total-cost) (cost ?waypoint1 ?waypoint2)))
    (at end (at ?robot ?waypoint2))
   )
 )
)
```

PDDL Problem Example File

Listing A.2: MWR Navigation Problem

```
(define (problem robot-1)
 (:domain robot-world)

 (:objects
  robotbase
  landmark1
  landmark2
  landmark3
  landmark4
  task1
  task2
  task3
  task4
  robot1
 )

 (:init
  (= (total-cost) 0)
  (= (cost-w1w2) 4523)
  (= (cost-w1w3) 6746)
  (= (cost-w1w4) 1488)
  (= (cost-w1w5) 780)
  (= (cost-w2w3) 4979)
  (= (cost-w2w4) 5922)
  (= (cost-w2w5) 2325)
  (= (cost-w3w4) 8438)
```

```
(= (cost-w3w5) 2103)
(= (cost-w4w5) 2666)

(waypoint1 robotbase)
(waypoint2 landmark1)
(waypoint3 landmark2)
(waypoint4 landmark3)
(waypoint5 landmark4)

(task task1)
(task task2)
(task task3)
(task task4)

(robot robot1)
(at robot1 robotbase)

(can-move robotbase landmark1)
(can-move landmark1 robotbase)
(can-move robotbase landmark2)
(can-move landmark2 robotbase)
(can-move robotbase landmark3)
(can-move landmark3 robotbase)
(can-move robotbase landmark4)
(can-move landmark4 robotbase)
(can-move landmark1 landmark2)
(can-move landmark2 landmark1)
(can-move landmark1 landmark3)
(can-move landmark3 landmark1)
(can-move landmark1 landmark4)
(can-move landmark4 landmark1)
(can-move landmark2 landmark3)
(can-move landmark3 landmark2)
(can-move landmark2 landmark4)
(can-move landmark4 landmark2)
(can-move landmark3 landmark4)
(can-move landmark4 landmark3)

(task-at task1 landmark1)
(task-at task2 landmark2)
(task-at task3 landmark3)
(task-at task4 landmark4)
```

```
 )

 (:goal
  (and
   (task-done task1)
   (task-done task2)
   (task-done task3)
   (task-done task4)

   (at robot1 robotbase)
  )
 )

 (:metric
  minimize (total-cost)
 )
)
```