UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology
School of Electrical, Mechanical and Mechatronic Systems

Fast, Reliable and Efficient Database Search Motion Planner (FREDS-MP) for Repetitive Manipulator Tasks

by

Fouad Sukkar

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Engineering (Research)

Sydney, Australia

2017

Certificate of Authorship/Originality

I certify that the work in this thesis has not been previously submitted for a degree nor has it been submitted as a part of the requirements for other degree except as fully acknowledged within the text.

I also certify that this thesis has been written by me. Any help that I have received in my research and in the preparation of the thesis itself has been fully acknowledged. In addition, I certify that all information sources and literature used are quoted in the thesis.

Signature of Student:

Date:

This research is supported by an Australian Government Research Training Program Scholarship.

Acknowledgements

I am grateful for my family and friends' support and encouragement during my studies. To Pablo Ramon Soria and Wolfram Martens, I am deeply grateful for allowing me to use your work, namely the perception module in my hardware experiments, and for assisting with its integration. A special mention goes to my supervisor A. Prof. Robert Fitch for his unfailing support and assistance.

Fouad Sukkar

Sydney, Australia, 2017.

List of Publications

Journal Papers

J-1. **F. Sukkar** and R. Fitch, "Fast, Reliable and Efficient Database Search Motion Planner (FREDS-MP) for Manipulator Tasks," *IEEE Robotics and Automation Letters*, 2018 **(in preparation)**

Conference Papers

C-1. P. R. Soria, **F. Sukkar**, W. Martens, B. C. Arrue and R. Fitch, "Multi-View Probabilistic Segmentation of Pome Fruit with a Low-Cost RGB-D Camera," *ROBOT'2017 - Third Iberian Robotics Conference*, November 22-24, 2017. **(accepted)**

Contents

List of Figures

ABSTRACT

This thesis presents FREDS-MP, a motion planning framework that leverages state of the art methods for solving a set of practical agricultural manipulator applications. Current methods exhibit unacceptably slow planning and execution times, hence FREDS-MP aims to bridge this gap by speeding up planning times whilst maintaining high reliability and solution efficiency. While only a specific set of applications are explored, FREDS-MP can be adopted for other similar applications seamlessly due to its general interface. FREDS-MP consists of three planning phases: offline, task and online. The offline planner pre-computes trajectories and cost information based on special cases that anticipate the real world. This pre-computed information is used by the task planner to compute accurate heuristics for sequencing tasks. The pre-computed trajectories are used as initial seeds by the online planner which utilises state of the art trajectory optimisers to adapt them in real-time to online tasks. Software simulations are performed to validate FREDS-MP and compare it to other state of the art planners. Further, the suitability of two commercial manipulators, six-DOF and seven-DOF, are compared for the intended applications. Several unconstrained and constrained tasks, commonly seen in agricultural applications, are tested under diverse obstacle configurations. Statistical results based on planner performance metrics are presented. From these results it was found that FREDS-MP significantly outperformed other state of the art planners when using a seven-DOF manipulator. Hence, an active perception experiment was carried out on a real Rethink Robotics Sawyer robot arm which was tasked to seek out apples on an artificial trellis and inspect them individually. The results from these experiments are presented and validate the practicality of FREDS-MP.

Abbreviations

DOF - Degree(s) of freedom

SEM - Standard error of the mean

IK - Inverse Kinematics

FK - Forward Kinematics

RRT - Rapidly-exploring Random Trees

TSP - Travelling Salesman Problem

BIT* - Batch Informed Trees

CHOMP - Covariant Hamilton Optimisation Motion Planner

GPMP2 - Gaussian Process Motion Planner 2

URDF - Unified Robot Description Format

OpenRAVE - Open Robotics Automation Virtual Environment

COLLADA - COLLAborative Design Activity

OMPL - The Open Motion Planning Library

Chapter 1

Introduction

Autonomous robots have been used in many applications in a variety of industries and their prevalence is set to grow. In particular, agriculture stands out as an industry where robotics have yet to be widely adopted [1]. While there have been several successful implementations in the field of agricultural robotics, most of these systems are too slow or unreliable which has led to the eventual rejection of these systems by farmers [2]. A key factor to the success of these autonomous systems will be their ability to plan and execute tasks quickly, reliably and efficiently.

This chapter motivates the need for autonomous robots (more specifically, agricultural manipulators) that exhibit these characteristics. The research objectives, contributions and overview of this thesis are also presented.

1.1 The Need for Fast, Reliable and Efficient Manipulation

The evolution of robotics and intelligent systems is part of the dawn of a fourth industrial revolution, particularly in the manufacturing and mining industries [3]. With the increasing challenges faced by the agriculture industry it may not just be desirable but indeed necessary to follow suit. Some of the current key challenges include: increase in demand due to population growth, decrease in workers, increasingly older workforce, changes in diet in developing economies, constraints on land allocation, high cost water and energy, plus environmental concerns and climate change [1]. In Australia, these challenges pose a significant threat, particularly due the development of very competitive overseas markets and the reduced availability of land water [4]. The issue of rising labour costs and increasing uncertainty surrounding the future availability of farm labour is particularly a concern for production of high-value specialty crops, such as: fruits and vegetables, tree nuts, dried fruits, horticulture, and nursery crops, which are still largely dependent on manual labor [5].

In an attempt to address the above challenges, the application of robotics and automation in agricultural environments is becoming increasingly popular and studied in recent years [6]. By automating typical operations in the agricultural industry, otherwise carried out using manual labour, many future farming issues may be addressed by improving the quality of produce while maintaining high yields; increasing efficiency through high precision resource allocation; enabling cost-effective implementation of non-chemical techniques for crop protection; preserving soil health; relieving humans from repetitive, intensive, fatiguing labour in harsh environments; and extending working time and production timeliness [7]. If the Australian agricultural industry exploits this technology, they may be able to remain competitive globally and become more sustainable in their practices.

The focus of this motivation will be mainly on mobile manipulators used for agricultural applications, but the concepts can be extended to many other applications, not necessarily agricultural. Mobile manipulators are ideal for automating many agricultural tasks, such as harvesting, mechanical weeding, pruning and planting seeds. Other than these traditional tasks, manipulators can be used to automate more abstract tasks such as a human visually inspecting crops and counting yields. A robotic manipulator can facilitate such task automation through active sensing, where a sensor is mounted on the end-effector of the manipulator. Such methods have been proposed as a way to solve the issue of occlusions when sensing for fruit and vegetables in orchards [8].

In order for these robotic manipulators to be feasible alternatives to human labour and current machinery, they should be able to carry out their tasks approximately as fast and ultimately have higher productivity. This means that the autonomous decision making should be able to process information about the environment quickly without compromising the effectiveness and reliability of the system. In agricultural environments, this is particularly challenging due to their physical nature. They are semi-unstructured, have cluttered/ complex geometry, not well controlled and dynamic, i.e. change over time [7].

Efforts have been made to speed up this decision making process [9], however, in this application the manipulator was an industrial bin picking robot operating in a relatively uncluttered and simple environment that was well structured and

controlled. Ideas from industrial applications have been adapted to agricultural applications [10] but they are not quick and reliable enough to be considered as an alternative to current farming methods.

In order for these robotic systems to run for several hours at a time, as humans and machines do, they must be reliable. This means that they must carry out tasks consistently well and without human intervention. Machine break downs happen on farms regularly, however they are generally designed so that farmers can fix them with minimal technical knowledge. It is highly unlikely that a farmer will be equipped with the knowledge to debug and fix these robotic systems should they break down or stop working as they are supposed to. Hence, reliability is critical in ensuring productivity is maintained over extended periods of time.

For machines to operate on farms they need to utilise energy. This energy may take various forms; for example, fuel, fertiliser and chemicals account for 16% of farm costs [11]. The key to reducing the use of these energy sources is increasing operation efficiency. Robotic manipulators have the ability to make precise and informed decisions on how they operate in an effort to reduce the amount of time and energy spent carrying out their tasks.

The work in this thesis addresses the need for fast, reliable and efficient manipulation, particularly in agriculture, and proposes that robotic manipulators provide a better alternative to human labour and machines for a wide range of applications.

1.2 Motion Planning and Manipulation

The need for fast, reliable and efficient manipulation has been motivated. Here, each of these characteristics will be addressed in the context of a motion planning problem.

A common problem in autonomous systems is determining how a robot should move from a start state to a target state in a given environment. Motion planning is the process of determining this movement, often by breaking it down into a sequence of discrete motions so that when executed moves the robot from its current configuration to a goal configuration without colliding into obstacles. Further to avoiding obstacles, another objective of motion planners can be to optimise trajectory costs, such as energy consumption, execution time and even uncertainty [12].

The problem of moving a manipulator from a start to goal state is a relatively simple problem if obstacle collisions are not a concern. Once obstacles are introduced, the problem is no longer trivial and motion planning is needed. In fact it is well known that a complete motion planning algorithm for this problem is proven to be PSPACE-Hard in terms of the number of DOF of the robot system [13] [14]. A complete planner here means for any planning problem instance, the algorithm will either find a solution or will correctly report that no solution exists.

Problem complexity is dependent on the dimension of the configuration space, the set of all possible robot configurations. In this context, the dimension of the configuration space maps to the number of DOF of the manipulator. In order for a manipulator to achieve an arbitrary end effector position in space with a specified orientation, six-DOF are necessary [15]. Hence, manipulators suited for dexterous tasks, such as harvesting, mechanical weeding and active sensing, usually have at least six-DOF [2].

An aim of this thesis is to compare the performance of a six-DOF manipulator against a redundant seven-DOF manipulator for a specific set of practical applications. Redundant manipulators have seven or more DOF and offer greater dexterity and flexibility when avoiding obstacles and even reducing torque, however, they come at the cost of greater kinematic complexity [16]. Further, they can have many or even infinite configurations that achieve the same end-effector pose, significantly increasing the size of the problem.

Another contributing factor to motion planning computational complexity is the process of checking whether a computed trajectory is collision free. This is particularly expensive for a manipulator given its complex mechanical structure and large number of rigid bodies that need to be checked against external obstacles and internal self collisions. Obstacles in agricultural tasks are particularly challenging because they are often cluttered and unstructured, meaning more computationally expensive collision checking is needed when computing trajectories [17].

Path planners that attempt to speed up the search for feasible plans in high-dimensional configuration spaces exist but they either come at the cost of having no trajectory optimality guarantees, or are only fast/ feasible for a certain subset of problems [18]. This thesis aims to mitigate this computational speed versus

trajectory optimality trade-off in even the most difficult problems. Path optimality with respect to execution time will be referred to as trajectory efficiency in this thesis.

Fast motion planning in the context of this thesis means producing a feasible trajectory, or determining if one does not exist, within a time frame that would be close to that which a human would take thinking about the same problem. This is critical for applications where multiple tasks need to be considered, for example examining multiple fruit on a vine with a sensor attached to a manipulator end effector, known as eye-in-hand sensing.

For multiple tasks, trajectory efficiency can be increased depending on the order in which the manipulator executes its tasks. For high-DOF manipulators, many different configurations can be utilised to perform the same task. Hence, all possible trajectory sequences and task configurations need to be considered. To solve such a problem, the planner would run in order $O(x^n)$ time where x is the cardinality of the largest set of possible manipulator configurations for a single task, and n is the number of tasks. Given that the problem of computing a single feasible trajectory is computationally expensive alone, clearly in order to compute multiple trajectories and determine an efficient sequence in feasible time there is a need for a faster planning method.

Lastly, reliability refers to the consistency of the motion planner and its ability to handle a diverse range of problems. But more importantly, the planner should exhibit a high success rate when planning trajectories. Traditionally, motion planners will not realise they failed to find a trajectory until they exhaust all of the search space or a time limit is reached. Clearly, given the constraints and complexities above for multi-task problems, this is not ideal. Hence, this thesis aims to maintain a high planning reliability without compromising computation speed or trajectory efficiency.

1.3 Research Objectives

The aims of this thesis can be summarised as follows:

i. Formulate and implement a motion planning framework that produces efficient solutions to a specific set of agricultural manipulator applications faster and

more reliably than existing state of the art methods.

ii. Facilitate generality such that the motion planning framework can be applied to a wide range of applications and robot manipulators.

iii. Validate and compare the motion planning framework against other state of the art methods in simulation with two commercial robot arms, one with six-DOF and another with a redundant seven-DOF.

iv. Tabulate and analyse results for a diverse set of obstacle configurations of varying complexity and a varying number of unconstrained and constrained tasks.

v. Validate the motion planning framework on a real robot arm by performing active perception and apple inspection on an artificial apple trellis.

1.4 Approach

Purely online planning methods plan trajectories at run-time without considering any prior information about the environment or task. This can be severly limiting because the planner needs to concurrently reason about the environment and produce an efficient, or at least feasible, trajectory. Some online planners, such as trajectory optimisers, use an initial approximation of the trajectory and incrementally improve upon it. However, this initial "seed" is usually naive and does not consider the environment's structure. As a consequence, these planners are sensitive to initialisation and if a bad initial seed is produced they may take a long time or never even converge to an collision free trajectory.

To overcome the computational complexities of manipulator motion planning at run-time, a new planning framework, Fast Reliable and Efficient Database Motion Planner (FREDS-MP) is proposed. FREDS-MP performs an offline computation phase which anticipates the application's environment and tasks. In this way, non-naive trajectories and cost information can be pre-computed and then used as initial approximations during the task and online planning phases. By leveraging state-of-the-art trajectory optimisers, these pre-computed trajectories can be adapted to

online tasks in real-time, significantly increasing convergence time and success rates by mitigating local minima issues that trajectory optimiser are inherently prone to.

One of the main challenges of the offline planner is developing an obstacle and manipulator task representation that is tractable and can be utilised effectively by the task and online planning phases. A key assumption is that, for the applications this thesis is concerned with, the manipulator will be performing repetitive tasks within a reduced sub-volume of its workspace. In this way, the task space that the offline planner needs to consider is reduced significantly. Similarly, obstacle configurations can be generalised by representing them as a union of basic shape primitives, forming a bounded volume in place of the real obstacle.

Task sequencing efficiency is increased by computing optimal manipulator configurations for tasks during the offline phase. Having readily available unique configurations to satisfy tasks at run-time significantly reduces the sequencing problem's complexity. Further, readily available trajectory costs that consider manipulator motion and obstacle avoidance provide informed heuristics which can be utilised during task sequencing for making better informed decisions.

It is important to note that there are some disadvantages when using the proposed method in place of purely online methods. Namely, there are trade-offs between offline and online computation of trajectories, loss of flexibility in the available solutions is one example. These tradeoffs will become evident throughout this thesis and explained in more detail later in the comparison and analysis sections, namely Sections 5.2.5 and 6.4.

Generality is facilitated by leveraging standard robot file formats and motion planning interfaces provided by open source libraries. Hence, FREDS-MP can be easily extended to other similar manipulator models. Further, task space and obstacle definitions are flexible and can be extended other similar applications seamlessly.

Simulating multiple commercial manipulators in varying environments is made possible by an open source simulator that provides seamless robot model integration with its interfaces, such as kinematics libraries, motion planners and collision checkers.

Hardware experiments are carried out on a real Rethink Robotics Sawyer robot arm which provides an SDK developed in ROS. The SDK provides several conve-

nience functions that abstract lower level operations. This allows for rapid development through its easy to use interface.

1.5 Contributions

The main contribution of this thesis is a new motion planning framework, FREDS-MP, that leverages current state of the art planning algorithms and builds on existing methods that utilise a prior offline planning phase. Importantly, FREDS-MP produces motion plans for a specific set of agricultural manipulator applications faster and more reliably than stand alone state-of-the-art methods. Furthermore, these motion plans are more efficient because they yield significantly lower execution times.

Key to FREDS-MP's performance is the sequencing method used. While previous agricultural manipulator applications have only considered heuristics based on workspace distances when sequencing tasks, this framework utilises more informed heuristics that consider non-linear costs such as manipulator motion and obstacle avoidance costs. This allows for well informed decisions when sequencing tasks that would otherwise be influenced by potentially, heavily under-estimated costs.

Another contribution of this thesis is the practical comparison of two commercial manipulators, a six-DOF and seven-DOF, for performing tasks commonly seen in agricultural applications. Comparing different manipulator kinematic structures has been considered previously, however, the comparisons have mostly been based on analysing workspace reachability and manipulability measures. Whilst these are valid, the comparison in this thesis is arguably more useful in that it considers practical implications directly, such as their effect on motion planning and task planning performance. Very few studies have considered this and if they did, the comparisons were carried out for a single obstacle configuration and task scenario with custom ball and stick manipulator models. This thesis performs a comparison over a diverse set of obstacle configurations and task scenarios, unconstrained and constrained, with commercial manipulator models.

Lastly, FREDS-MP is applied to an active perception problem with a real robot arm that localises and performs close up inspection of apples on an artificial trellis. While previous applications have only considered simplified obstacle representations

for such tasks, this thesis considers non-trivial obstacles which FREDS-MP deals with effectively.

1.6 Thesis Roadmap

This thesis is organised as follows:

Chapter 2: This chapter explores real world manipulator applications and identifies gaps in current implementations. A survey of the various types of motion planners and their state-of-the-art algorithm is given. Background knowledge on manipulator motion planning necessary for understanding subsequent chapters is also presented.

Chapter 3: The thesis problem statement is formulated here. Algorithm components used in this thesis are also formulated.

Chapter 4: FREDS-MP algorithm is introduced and its implementation details are presented. A time and space complexity analysis of FREDS-MP and each of its sub-components is performed.

Chapter 5: FREDS-MP is tested in simulation with a Universal Robots UR5 and Rethink Robotics Sawyer manipulator. Tabulated results are presented and discussed. A comparison on the suitability of each manipulator for the intended applications is also given.

Chapter 6: FREDS-MP is validated on a real Sawyer robot arm which performs active perception and inspection of apples on an artificial trellis. Results are presented and discussed.

Chapter 7: A brief summary of the thesis contents and its contributions are given in the final chapter. Additionally, recommendation for future work is discussed.

Chapter 2

Related Work and Background

In this chapter, manipulator motion planning in various real world applications is explored. While the focus of the review is on agricultural applications, relevant ideas from other industries such as industrial applications are discussed. Furthermore, state-of-the-art motion planning algorithms, designed to be application agnostic, are explored and their flaws are identified. Finally, relevant background knowledge on manipulator motion planning is presented which serves as a prerequisite for the technical content in this thesis.

2.1 Manipulators in Agriculture

Various motion planning, task planning and manipulator designs have been proposed in agricultural applications recently. This section reviews the various methods used and analyses areas that need improvement, hence motivating the aims and contributions of this thesis.

2.1.1 Motion Planning

Agriculture has seen a long history of robotics. For the past 30 years there has been active development in this field in an effort to automate various harvesting and horticultural tasks [2]. This thesis is mainly concerned with manipulator applications. Some notable applications include robotic cucumber harvester [19], strawberry harvester [20], kiwifruit harvester [21], sweet-pepper harvesters [22] [23] and apple harvesters [24] [25].

Europe has been a key player in agricultural robotics development in recent years with successful European Union (EU) project initiatives CROPS [26] and SWEEPER [27]. CROPS' main focus was on carrying out research and development in agricultural robotics across a range of plantations and forests. SWEEPER aims to further develop a sweet-pepper harvesting robot [22] from the now ended CROPS

project in an attempt to make it market ready.

In fact, most agricultural robots developed in the past few years have been mainly applied to automating harvesting while there has been less attention to other horticultural tasks, such as pruning, weeding, spraying and crop monitoring. It is no surprise that this is the case; for example, in Washington State, US local apple growers reported that harvesting labour accounts for approximately a third of their annual variable costs - as much as pruning and thinning combined [28].

That is not to say that robotic manipulators would not have a significant impact in horticulture. A major motivator for research into weed destruction technology is the increasing resistance that some weeds, called "super weeds", have built to traditionally used agrochemicals - which have been preferred in the past due to their low cost and high efficiency. Alternative weed killing technologies that has been proposed include; mechanical methods, microwave technology and solarisation [29] [30] [31] [32]. A mobile manipulator could facilitate the use of these technologies by attaching appropriate devices to their end-effector. This would allow for high precision and close proximity application, even when crops are present.

Manipulators have been utilised in the past for automating weeding methods utilising electrocution [33], rotating blades [34] and a combination of chemical application and mechanical cutters [35]. The motion planning methods for these projects relied on simple steering methods without taking into consideration obstacles. Similarly, trajectory optimisation path planners have been used in harvesting applications, such as kiwi fruit [21] and apple picking [25]. However, again these do not take into consideration obstacles.

A paper that focused on manipulator weeding [10] used an approach where a database of paths was pre-computed offline using a sampling based planner and then adapted online. This method was inspired from an industrial bin-picking application [9], however instead of having to tune parameters to determine which path segments to optimise, the work used a more general convex optimisation procedure. While this method proved promising, there was a large variance in success rate, ranging from 50% to 70% for paths that were computed in under 0.5 seconds. On average the computation time for plans that succeeded did not exceed 2.5 seconds, but the variance was still large. Further, computation time for plans that failed were

large and not included in the reported averages. The large variance in the results was due mainly to two reasons: while the offline planner took into consideration obstacles, the online planner did not when adapting trajectories; and secondly the online inputs did not match up well all the time to the database. The latter occurred because the offline planning phase only considered a single IK solution for each candidate location, where as during the online phase the IK solver considered all solutions. Whilst this is true of FREDS-MP, the unique IK solutions are more intelligently computed during the database construction and it is shown that a redundant manipulator helps by providing a higher number of IK solutions, hence increasing the probability of matching closely with the database solution. Interestingly, the authors noted that the planning success was sensitive to initial conditions which further supports this thesis's approach. Nevertheless, the computation speed performance is still very fast when considering previous motion planning approaches could take up to 45 seconds to carry out similar tasks [19]. This thesis draws inspiration from this method and aims to increase the reliability, i.e. success rate, while maintaining consistently low average computation times and without compromising path efficiency.

More recently, a grape vine pruning robot [36], which took into consideration obstacles during motion planning, claimed faster planning times than in [10] due to a novel collision checking algorithm designed specifically for grape vine structures. However, they used a Bi-direction RRT motion planner [37] which resulted in very high execution times. A sweet pepper harvesting robot from the CROPS project [38] also used Bi-directional RRT but was not considered feasible for real-time application due to the long planning times, in the worst case taking about half an hour.

In [39] and [23] it is reported that only self collisions and a simplified planar obstacle placed slightly behind the crop row were considered, instead the manipulator was planned to a pose, offset from the grasping pose with the same orientation, and commanded to move along this direction vector towards the crop. This is acceptable for the final grasping motion, however, point to point movements are still susceptible to collisions, particularly if there happens to be a loose hanging branch in the middle of the workspace, which is common in orchards. Interestingly, a significant amount of harvesting failures were reported due to collisions with such obstacles

while transporting the grasped fruit.

A sweet-pepper harvesting robot [23] used Moveit!, a high performance ROS package for motion planning, however the authors did not report which planners they used. They did report performance, however, which included slow planning (5-10 seconds per capsicum) and execution (10-15 seconds per capsicum) times. Obstructions such as leaves and string from the orchard caused a large portion of the grasping failures.

Unfortunately, in almost all of the reviewed applications, a majority of the reporting was spent on the design of the manipulator and crop environment as well as the perception and localisation subsystems, neglecting the motion planning subsystem. For the small number of applications that did consider obstacles in the motion planning they did not take into consideration path efficiency and if they did they were not considered fast enough to run in real-time due to high planning or execution times.

This trend of neglecting obstacles and trajectory efficiency in the motion planning process is supported in a recent review of fifty harvesting robots [2]. The authors also noted that performance improvements over the last thirty years have been partially constrained by an absence of reporting on test conditions, performance indicators, hardware systems, and software systems. While a relatively large portion of these projects reported their motion planning algorithms, only 4% reported obstacle localisation and 6% task planning. This highlights the complex nature of motion planning in agricultural applications and may be a large contributing factor to its neglect.

In summary, agricultural manipulator applications to date have suffered severely from slow execution times. This can be attributed to their lack of attention to the motion planning problem. Moreover, obstacle avoidance has been neglected due to the assumption that the orchards have been well maintained and hanging branches were "cleaned up" prior to field trials. While authors suggest that optimising the design of the orchards for robotic harvesters will solve a lot of the complexities involved with the motion planning problem, this requires considerable work on the farmers part and it may take a long time, if ever, before farmers adopt such practices universally.

2.1.2 Task Planning

According to the review of fifty harvesting robots in 2014 [2] only 6% of these reported task planning performance. Computing an efficient sequence to execute tasks, such as which order to pick crops, can make a significant impact on the overall execution time. Applications in the past have considered this [25] [40] and set the problem up as a Travelling Salesman Problem (TSP), however they only considered Euclidean work space distance between targets as a cost metric. Similarly, in [39] the sequencing was based on the height of the Sweet Peppers to be harvested. Given the non-linearity of robot arms, this can result in significantly underestimated costs. For example, a small distance between two end-effector positions may not necessarily correspond to a small difference in manipulator configuration. Furthermore, these heuristics do not take into consideration obstacle costs which can be highly non-linear.

The importance of proper sequencing is supported by the results observed from the grape vine pruning robot [36] where while it exhibited fast planning times, the main performance bottleneck was high execution times which was due to the fact that they did not consider optimising the sequence of pruning tasks. While an accurate heuristic can be computed, obtaining this heuristic is non-trivial and often equivalent to solving the original problem [41]. Another issue was that they did not exploit the manipulator's redundancy for sequencing tasks.

The motion planning framework developed in this thesis aims to utilise more informed heuristics that consider non-linear costs such as manipulator motion and obstacle costs. Further, manipulator redundancy will be leveraged for sequencing tasks efficiently.

2.1.3 Manipulator Design

Most of the effective agricultural manipulator applications reviewed thus far have six-DOF or higher. This is due to the fact that the higher-DOF arms provided more dexterity for tasks such as harvesting and pruning. In contrast, an apple harvesting robot in Japan [24] used a three-DOF manipulator to detach apples via their stem, however could only approach the apple horizontally due to this and hence had trouble detaching apples with shorter stems. A recent paper [25] suggests that

a way around this is to optimise the design of the orchards for robotic harvesters. While this may be true, until this is a widely adopted practice by farmer, high-DOF manipulators may be the only option to work around orchards' highly unstructured nature.

According to the review of fifty harvesting robots in 2014 [2] no authors had explained why they had chosen their particular manipulator kinematic structure. More recently, for a capsicum harvesting application [22] authors chose to use a Baxter out of convenience because it had native ROS support and provided an IK service out of the box. While this is valid, the decision was not based on the manipulator's suitability for the intended application.

Some authors optimised the manipulator kinematics based on quantitative measures such as the workspace reachability and manipulability [42] [43] [44] [45] [46] [47], or dexterity, where the latter uses the determinant of the manipulator's Jacobian to determine the magnitude of its singular values. This magnitude is proportional to the amount of motion achievable in each dimension of the Cartesian workspace.

A tomato picking robot paper [48] used a "redundant space" measure, which they describe as the space that the manipulator's middle links can reach while its end effector remains fixed to a 6D pose.

Out of all the above, only two manipulator designs considered practical implications directly, such as obstacle avoidance and motion planning performance, when optimising their design. One of these was a cucumber harvesting robot [42], however, the authors only considered optimising a fixed number of links and did not compare different number of DOF. The second was a sweet pepper harvesting robot [43] and while it did compare six, seven-DOF and eight-DOF, it only did so for unconstrained point-to-point tasks for a single scene with the same three repeated tasks. Further, they did not simulate any real commercial robot arms.

Furthermore, none of the above analysed the effect that different manipulator kinematic configurations have on task planning performance. This thesis aims to compare two commercial manipulators, a six-DOF and seven-DOF, considering practical implications directly for a wide range of obstacles and task scenarios, unconstrained and constrained, while analysing the effect of the number of DOF on motion planning and task planning performance.

2.2 State-of-the-Art Motion Planners

Although in agricultural applications there has not appeared to be significant progress in manipulator motion planning, general motion planners have seen significant break through in the robotics field. In this section relevant state-of-the-art motion planners are explored. Further, discussion is carried out on how they may be utilised for manipulator motion planning in the context of this thesis' focus on agricultural applications.

2.2.1 Motion Planning

There are many algorithms and methods currently being researched that have been active research topics for decades. The textbooks by Choset et al. [12] and LaValle [49] provide a rich overview of fundamental motion planning algorithms. Today there exist several modifications to these algorithms, each with their own benefits and intended applications. Motion planning algorithms can be categorised into several types and sub-types where each have been adapted to deal with particular problems, for example: any-angle, incremental and any-time planners. The various motion planning algorithm types and their modifications are analysed below.

Grid-based Search

Grid-based search planners overlay a grid on to the robot configuration space, where each point in the grid corresponds to a robot configuration. The task is then to connect configurations in the grid, ensuring connections are in free configuration space, such that a path between the start and goal configuration is formed. This can be achieved using many different graph search algorithms which are designed to efficiently search through the grid, or any graph, for a solution. The grid can be constructed in various ways, however its main tuning parameter is the grid resolution. A higher grid resolution will be able to plan through narrower obstacle clearances, but the search time will take longer. Further, the number of nodes in a grid will increase exponentially in the configuration space dimension, making them non-ideal for high-dimensional configuration spaces.

Dijkstra's algorithm [50] is an early solution to the shortest path through a graph problem. Its minimum-priority queue implementation is the fastest known

single source, shortest path algorithm for arbitrary directed graphs with unbounded non-negative weights. It can be used for finding the shortest path between two nodes in a graph, however, it is more efficiently used for finding the shortest path from a source node to all other nodes in a graph. The algorithm works by effectively expanding a wavefront out from the source node until a goal node is found.

A* [51] is a popular graph search algorithm because of its break-through speed and accuracy compared to Dijktra's algorithm, which at the time was the best known graph search algorithm. It is fast because it utilises a heuristic to guide its search, potentially greatly reducing the amount of nodes checked in the graph. Further, it is an informed search, or best-first search, algorithm which means that it will find the lowest cost path in the graph from start to goal configuration. In order for this to hold, the heuristic used by A* must be admissible, which means that it must never overestimate the cost to get to the goal node. Hence, when finding the shortest path between a single source and goal node A* is more efficient to use than Dijkstra's, as opposed to finding the shortest path from a source node to all other nodes. If the heuristic is set to zero, A* becomes Dijkstra.

D* is an incremental heuristic search algorithm that was introduced to deal with dynamic graphs, i.e. those which change over time. Instead of simply re-planning from scratch, D* optimises the number of edge cost changes needed to maintain a lowest cost path. It was observed however, that if the goal node changes then A* is more efficient [52]. D* lite is a modification of D* that only uses one tie-breaking criterion when comparing priorities. Hence, D* lite is significantly simpler and is at least as computationally efficient as D* [53].

Grid-based search algorithms traditionally suffered from the fact that they could only produce paths with heading changes that were constrained to the direction between subsequent grid points. This often results in sub-optimal, jagged paths. Algorithms such as A* Post Smoothing were introduced to smooth these paths, however, they are unreliable because they cannot change what side of a blocked cell is traversed. Hence, any-angle path planners were introduced to allow turns in the path to have any angle when searching over a discrete graph, producing optimal or close to optimal paths. They are advantageous over traditional geometric planning methods, such as searching over a visibility graph, because the number of edges

in the graph grows linearly with number of vertices, as opposed to growing by the number of vertices squared, hence making them much faster.

Field D* is an any-angle planner based on D* and uses interpolation during vertex expansion to allow path movement through a finite number of points on the edges of each node in the grid [54]. Theta* is a simple variation on A* where every time a vertex is expanded, it checks if a line of sight path between its successor and parent exists. If so, then it uses the path directly between them [55]. AP Theta* and Lazy Theta* are optimised versions of Theta* that reduce the cost of these line-of-sight calculations [56]. Block A* uses memoisation to quickly find piece-wise any-angle paths for small parts of the graph [57]. ANYA is a variant that looks at an interval of points rather than a single point as a node [58]. Hybrid A* is an any any-angle planner that takes into consideration differential constraints, such as a non-holonomic vehicle [59].

Sampling-Based Planners

The above grid-based search methods do not scale well in high-dimensional problems. When dealing with high-dimensional problems, such as high-DOF manipulators, a common approach is to use sampling-based algorithms. Unlike combinatorial algorithms, their running time is not explicitly exponentially dependant on the dimension of the configuration space. Further, they avoid explicit construction of obstacles in the state space, hence reducing computation needed compared to complete planners. Another key property is that these planners are probabilistically complete, that is, as the planner continues running, the probability of not finding a solution, if one exists, asymptotically approaches zero. A drawback, however, is that they cannot determine if no solutions exists [60].

One of the first algorithms presented in this area was Probabilistic Roadmaps-(PRM) [61] which were designed with the intention to be used for multiple queries. PRMs produce sub-optimal paths, but the quality of the path can be increased as the number of samples in the roadmap increases, at the cost of longer computation times. The algorithm works by taking random samples, or nodes, from the configuration space of the robot and testing whether they are in free space. A local planner attempts to connect the nodes and if successful produces an edge connection between

them. A graph search is then used to find a path through the produced graph.

Since the introduction of the original PRM algorithm, a number of modifications have been developed to address different issues, such as; reducing the number of nodes needed for the roadmap [62], the ability to find paths through narrow regions [63] and reducing the time spent collision checking [64]. PRMs have also shown to be well suited to problems with complex constraints, including kinodynamic constraints [65].

While the offline database concept in this thesis is similar to the multi-query approach of road-map type planners, it does not explicitly enforce connections between nodes in the graph. Rather, the connections between nodes are implicitly defined and produced only at run-time during the online planning phase.

The Rapidly-exploring Random Trees (RRT) algorithm [37] is another fundamental sampling based planner, however unlike the PRM algorithm, the graph created is a tree type and designed for single query planning. It is generally used in high-dimensional problems where searching over a graph may be computationally expensive. While it possesses theoretical guarantees such as probabilistic completeness, it also suffers from the fact that the best path it finds almost surely converges to a non-optimal value [60].

Later, Expansive Space Trees (EST) were developed in an effort to sample nodes in areas that would be useful when expanding the tree [66], where as RRT simply draws samples uniformly. While EST and Guided Expansive Space Trees (GEST) [67] select nodes for expansion based on their neighbouring nodes, Path Directed Subdivision Trees (PDST) [67] and Kinodynamic Planning by Interior Exterior Cell Exploration (KPIECE) [68] select nodes for expansion based on their coverage, to ensure that expansion is not wasted on already explored areas.

As with PRM, there have been several modifications and improvements to RRT since its conception, such as Single-query Bidirectional Lazy (SBL) planner which applies a lazy collision-checking strategy to reduce the planning time [69]. SBL and RRT-Connect (Bi-directional RRT) [37] use two trees to perform bidirectional search. One tree is rooted at the start, whereas the other is at the goal. The search is complete when the two trees are connected. In doing so, less area is searched improving computational performance. CBiRRT [70] was introduced to deal with

general manipulator constraints and builds on top of BiRRT where the "extend" procedure is modified to include a constraint checking function.

CBiRRT was augmented with the concept of task-space regions (TSR) to focus the search regions of the configuration space [71]. Task-Relevant Roadmaps (TRM) [72] expand on this idea to produce constrained motions in task-spaces that are not limited to a restricted set of projectable constraints and TSRs. Rather the task-spaces can be expressed by arbitrary task-functions subject to arbitrary cost-functions. Moreover, during construction these task-spaces are maximally covered and connected in such a way that a reusable graph is produced for motion planning. For the constrained tasks explored in this thesis, CBiRRT was deemed sufficient, however TRMs are promising for expanding to more general constrained tasks.

Sampling based planners tend to produce jerky and unnecessarily long paths due to their random nature, which is not ideal for manipulator planning when efficient paths are desired. There are smoothing and shortcut algorithms [73] [74], that mitigate this but they require an additional post processing step. Furthermore, considerable computational effort is spent in sampling and connecting samples in portions of the configuration space that might not be relevant to the task.

Apart from theoretical performance improvements, efforts have been made to speed up the implementation of RRT and PRM methods. KD-trees have been used to speed up neighbour hood search when connecting nodes [75]. Further, parallelising these algorithms and running multiple searches has been proven to be effective at speeding up and improving their performance [76] [77] [78].

Any-time Planners

Any-time planners are an interesting area in planning research because they combine ideas from several planning categories, such as grid search, incremental and sampling-based planners. In fact, they are ideal in many real world applications because they allow the robot to safely begin executing plans in real-time and during execution they can continue improving the plan, updating the execution on the fly.

Anytime Dynamic A* (AD*) is a re-planning algorithm that combines the benefits of anytime and incremental motion planning methods. The algorithm plans a sub optimal path quickly initially and improves on it over time as more information

is gathered about the environment [79]. Incremental Phi* is an incremental and more efficient version of Theta* [80].

Optimal planners such as RRT*, PRM* [81] and discretisation-based approaches, R* [82] and ARA* [83], are very promising but are currently computationally inefficient for solving high-dimensional motion planning problems. Informed RRT* is a simple modification to RRT* and provides significantly faster solution convergence for problems where it is not necessary to explore a large portion of the configuration space [84]. FMT* was proven to work well in high-dimensional configuration spaces and provided substantially better solutions than RRT* [85]. It utilises a marching method which orders a search based on cost-to-come but this search must be restarted if a higher sampling resolution is needed. RRT$^{\#}$ improves upon the idea in RRT* by propagating local wiring to the rest of the tree and utilises incremental planning techniques to speed up optimality convergence [86].

Batch Informed Trees (BIT*) balances the benefits of graph search and sampling based planners by processing batches of samples, as in previous optimal sampling-based planners, however it updates the path between samples efficiently by leveraging incremental search techniques, incorporating new samples into the existing search [87]. BIT* proved to find better solutions faster than Informed RRT* and FMT* with faster anytime convergence towards the optimum, especially for higher-dimensional problems.

More recently, it was shown that the probability of sampling a point that improves the solution decreases exponentially with the dimension of the problem [41]. The same authors developed the DRRT algorithm which improved optimality convergence by performing a gradient descent on the produced samples to move them towards the optimal cost. The convergence rate was shown to remain almost constant for increasing problem dimension. While this is a promising result, when applied to a six-DOF manipulator problem with obstacles it suffered from high computation times and low success rates. This suggests that the algorithm works well in an ideal setting with perfect heuristics and no non-linear costs but does not generalise well to real world constrained problems.

2.2.2 Trajectory Optimisation

Trajectory Optimisation is a loose term given for producing trajectories, given some initial, and usually naive, seed trajectory. This seed is then incrementally optimised over time until an optimal or near optimal solution is found. There are several techniques, each with their own benefits and use cases. These techniques are divided into two categories; an optimal control problem that allows a solution via analytical or numeric approach and an approximation to this optimal control problem.

One of the most common approaches is Gradient Descent [88] which is a first-order optimisation algorithm. In the context of planning a trajectory for a manipulator, a local minimum is found by incrementally shifting a given trajectory in the direction proportional to the negative of the gradient of a function at a current point. For some cases Gradient Descent can be limited, for example, for Rosenbrock functions the gradient descent method can behave strangely as it approaches local minimums. Gradient Descent has been very popular for producing smooth, locally optimal trajectories for high-DOF manipulators due to its fast performance and easy implementation, however some major draw backs are that it can get stuck in local minima and without modification it does not take into consideration constraints such as avoiding obstacles.

Despite there being several methods for optimising trajectories, when considering the motion planning problem there are two limiting factors associated with most of these methods. The first is the large computational costs for evaluating objective functions and their high order derivative in high-dimensional spaces, and the second is the presence of local minima due to non-convex problems [89].

Covariant Hamilton Optimisation Motion Planner (CHOMP) [89] attempts to alleviate these problems by using a Hamiltonian Monte Carlo algorithm to perturb the trajectory in order to restart the process and prevent it from getting stuck in local minima. Further, CHOMP produces smooth paths via a smoothness functional and avoids obstacles via a obstacle functional. Instead of computing the cost field in the robot's high-dimensional configuration space, it is computed in the robot's workspace and, in the case of a robotic arm, uses body points along the arm to sum workspace costs. In this way CHOMP exploits the fast computation of Euclidean Distance Transforms which means functional gradients can be computed efficiently

for complex problems to enable replanning. Authors also derive an extension of CHOMP which can handle a wide range of equality constraints.

Few other trajectory optimisers consider obstacle avoidance. One approach that considers obstacle avoidance requires a description of configuration space obstacles which proves to be infeasible for high-dimensional manipulators [90]. Alternatively, some approximately optimal methods exist that only consider very simple representations of obstacles [91].

Other trajectory optimisation techniques have been applied in this field, such as modelling the trajectory as a mass-spring system [92] and performing planning by scanning back and forth along the elastic, while moving one mass particle at a time. The draw back of these methods are that they are sensitive to local minimum and can get "stuck" in them.

Kalakrishnan et al. [93] applies motion sampling techniques in a context similar to CHOMP. However, Stochastic Trajectory Optimization for Motion Planning (STOMP) obtains gradient information using trajectory samples while it is readily available which essentially estimates the gradients, where as CHOMP has readily available gradients from the signed distance field information.

A recent algorithm called TrajOpt [94], proved to compute better paths faster and with a higher success rate than CHOMP. TrajOpt differs from CHOMP in that it uses a second-order, sequential convex optimisation procedure (SQP) and convex-convex collision checking. It is argued that this collision checking method results in not only a better approximation of the configuration space but also better collision avoidance since it attempts to translate the entire link out of collision, rather than individual points along the link, as in CHOMP. TrajOpt can also incorporate various constraints, such as orientation, positional and even differential constraints.

In [94] it was proven that a good initial guess has a significant impact on convergence speed and success rate of trajectory optimisers. It was also noted that they are not guaranteed to find a collision-free solution as the no-collisions constraints in the optimisation problem are non-convex. Hence, this motivates the need for a precomputed feasible path as the initial guess when planning paths for manipulators using trajectory optimisation approaches.

Gaussian Process Motion Planner 2 (GPMP2) [95] recently attempted to speed

up the trajectory optimisation problem by reducing the amount of collision checking along the trajectory by using probabilistic inference to only check configurations that are likely to be in collision. It uses the same collision avoidance method as CHOMP, however, it does not avoid getting stuck in local minima. GPMP2 produced promising results with faster planning times and higher success rates than TrajOpt and CHOMP.

2.3 Background

Fundamentals in motion planning, in particular manipulator motion planning, exists in a wealth of literature. The background material summarised here can be found in textbooks by Lavalle [49] and Choset et al. [12]. The concepts explored here and their formulation are relevant to the motion planning problem in this thesis and setup the background knowledge needed to understand the subsequent chapters, in particular Chapters 3 and 4.

2.3.1 Discrete State Spaces

Let us define a *state* x as a particular situation for the robot to be in within the world it is operating, and the set of all possible states as the *state space* X. For discrete motion planning, this state space should be finite or countably infinite.

The world may be transformed through *actions*, chosen by the planner in this context. A *state transition function*, f, specifies a transition from a current state, x, to new state, x', when an action, u, is applied. Hence a *state transition equation* can be defined as

$$x' = f(x, u).$$

Let $U(x)$ denote the action space for each state x, representing the set of all actions possible from x. The set U is then all the possible actions over all states

$$U = \bigcup_{x \in X} U(x).$$

When there a multiple goals, a set of goals states $X_G \subset X$ is defined. The task then for a planning algorithm is to find a finite sequence of actions that transform

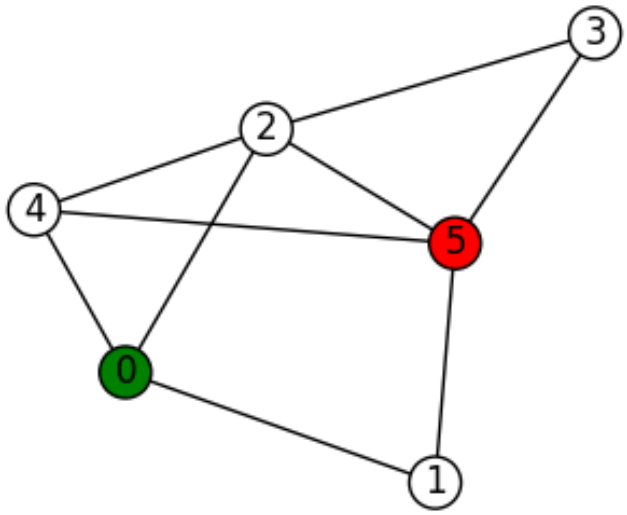


Figure 2.1 : Example of a state transition graph with vertices and edges. The green vertex is the initial state and the red vertex is the goal state.

an initial state x_0 to a goal state in X_G.

This formulation is often visualised as a *state transition graph*, as seen in Figure 2.1. This graph is made up of vertices, i.e. the state space X, and edges between these vertices, meaning $\exists u \in U(x) \mid x' = f(x, u)$.

In the context of this thesis, tasks can be seen as vertices and the edges connecting these tasks are the costs of the trajectories that the manipulator executes to perform a particular task starting from a previous task.

2.3.2 Planning in Continuous Space

In any real world application where a robot must move through an environment without colliding into any obstacles, a continuous state space must be considered. Motion planning refers to planning in continuous state spaces. A motion plan, or trajectory, describes the sequence of motions that a robot makes so that it can transition from its current state to a goal state.

A number of challenges arise when planning in continuous state space. One is dealing with implicit representations of the state space. This is particularly difficult because the state space is uncountably infinite. Another challenge is transforming a continuous model into a discrete one. Later in Section 3.6 various algorithms that deal with these challenges will be presented.

Geometric Representations

The world, or map, in which a motion planning problem is defined within, comprises a robot and obstacles. These geometric objects can be represented in many ways. However, there are generally two ways: using a *boundary representation* and a *solid representation*. A boundary representation would use, for example, an equation of a sphere to represent a spherical obstacle, where as a solid representation would describe the set of points contained in the spherical obstacle. In this thesis, the former will be used.

Let us define the 3D world as $W = \mathbb{R}^3$. Obstacles and robots are then closed subsets of W. Let an *obstacle region* O denote the set of all points in W that lie in one or more obstacles, $O \subseteq W$.

The objective of an obstacle representation is to be expressive yet computationally efficient. For this thesis, obstacles will be represented as a union of convex basic shape primitives

$$O = O_1 \cup O_2 \cup \cdots \cup O_n,$$

where O_i are one of the following basic shape primitives: cylinder, sphere, box, cone, capsule, convex mesh or plane. A robot can be defined similarly, and will be denoted as $\mathcal{A} \subseteq W$.

Kinematic Chains and Transforms

While obstacles, O, remain fixed, motion planning problems require moving the robot, A. A *rigid body transformation* is a function $h : A \to W$, that maps every point of A into W, such that; 1) the distance between any pair of points in A is preserved and 2) the relative orientation of these points in A is preserved. An image of h is defined as

$$h(A) = \{h(a) \in W \mid a \in A\},$$

which indicates all points in W occupied by the transformed robot.

For convenience, a transformed robot will be defined simply as $A(q)$, where q are parameters that define the transformation. For example in the 3D case,

$A(x_t, y_t, z_t, \alpha, \beta, \gamma)$ would be the result of rotating the robot by α, β, γ and then translating it by x_t, y_t, z_t, where α, β, γ is the amount of counter-clockwise rotation about the z-axis, $y - axis$, $x - axis$ respectively. This combined rotation and translation will be represented in the form of a 4×4 *homogeneous transformation matrix*

$$\mathbf{T} := \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0^T} & 1 \end{bmatrix} \in SE(3),$$

where R is a 3×3 rotation matrix and t is a 3×1 translation vector. A rigid body that can translate and rotate in 3D has 6 degrees of freedom, hence T will represent what is called a 6D pose.

A manipulator can be considered as a chain of attached rigid bodies, or *links*, making up a *linkage*. More specifically if the links are attached to make a single chain then the linkage is called a *kinematic chain*. Let $A_1, A_2, \ldots, A_m$ denote a set of m links. For each i such that $1 \leq i \leq m$, link A_i is attached to link A_{i+1} in a way that allows A_{i+1} some constrained motion with respect to A_i. Each of the links are attached by a joint - in this thesis only *revolute joints* will be relevant.

It is convenient at this point to define coordinate frames. When transforming A, it places the robot in W, i.e. it becomes a subset of W. If the origin of W is the *world frame*, then any point in W can be expressed in terms of the world frame. Hence, the robot A is initially expressed in the *body frame* and transforming A converts it from the body frame to the world frame.

This idea of expressing different points of A in terms of a specific frame is crucial when considering kinematic chain transforms. In a kinematic chain, each link has its own coordinate frame and transforms require expressing all links with respect to the world frame. The location in W of any point $(x, y, z) \in A_m$ can be determined by multiplying the transformation matrices

$$T_1 T_2 \ldots T_m \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

The pose of a coordinate frame A is given by the position and orientation of another coordinate frame, B, with respect to A by the affine transform matrix

$$\mathbf{T_{AB}} := \begin{bmatrix} \mathbf{R_{AB}} & \mathbf{t_{AB}} \\ \mathbf{0^T} & 1 \end{bmatrix} \in SE(3),$$

where T_{AB} is a 3×1 translation vector and R_{AB} is a 3×3 rotation matrix. In this way, the pose of each link can be determined by substituting in each of the manipulator's joint angles into the corresponding transform matrix. The collection of transforms describing each link's pose is called the *Forward Kinematics* (FK) of the manipulator.

The *Inverse Kinematics* (IK) is the opposite problem, i.e. given a 6D pose of a link, how should the joints of the manipulator be arranged to achieve this pose. This is a much more difficult problem and requires algebraic manipulation of the corresponding link's, more commonly the last link or "end effector", transform matrix. Some IK solvers use numerical methods to determine solutions, however, analytical methods are preferred when possible because they are able to compute solutions quicker. Both numerical and analytical methods can provide multiple solutions for the same pose. The utility of having redundant solutions will be evident later on in Chapter 4 of this thesis.

Configuration Space

The configuration space or C is the set of all possible configurations of a robot. It can also be seen as the state space for motion planning, or the set of all possible transformations that can be applied to a robot. The concept of C was first developed by Lozano Perez [96] and is important because it acts as an abstraction layer to defining motion planning problems in terms of geometry and kinematics. This abstraction permits many different problems to be solved using the same planning algorithms.

In configuration space, the configuration of the robot is represented as a point. The idea is then to separate the configuration space into two regions, one being all the configurations which the robot is in collision and another where it is not. If a world, W, contains an obstacle region, $O \subset W$, and a rigid robot $A \subset W$, with configuration $q \in C$, then the obstacle region is

$$C_{obs} = \{q \in C \mid A(q) \cap O \neq \emptyset\},$$

and the *free space* region is

$$C_{free} = C \setminus C_{obs}.$$

The motion planning problem then becomes a path planning problem for a point in n-dimensional space, where n is the number of DOF of the robot. For a path to be valid it must be in C_{free}.

More precisely, given; W, $O \subset W$, A, an *initial configuration* $q_I \in C_{free}$ and *goal configuration* $q_G \in C_{free}$, compute a continuous path

$$\tau : [0, 1] \to C_{free} : \tau(0) = q_I,\ \tau(1) = q_G,$$

otherwise, report that such a path does not exist.

Manipulator Configuration Space

The manipulator motion planning problem is an extension of a single rigid body robot to a *kinematic chain*. The C-space of a rigid body that can translate and rotate about a plane is

$$C = \mathbb{R}^2 \times \mathbb{S}^1,$$

where $\mathbb{S}^1$ is a circle

$$\mathbb{S}^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\}.$$

Since the bodies of a manipulator cannot move independently, the C-space cannot simply be combined using Cartesian products. Rather, there is no general method and each manipulator needs to be considered case-by-case depending on its joint properties. The main property we are concerned with, particularly because we are only dealing with a series of links connected by revolute joints, is whether or not the joints are able to swing around to enable any $\theta \in [0, 2\pi)$.

If the joint cannot swing around $\mathbb{S}^1$ then C-space for that joint is homeomorphic to $\mathbb{R}$. That is $C = \mathbb{R}^n$ where n is the number of joints. Where as if they can swing around to achieve any orientation then

$$C = \mathbb{S}^1 \times \mathbb{S}^1 \times ... \times \mathbb{S}^1 = \mathbb{T}^n,$$

where $\mathbb{T}^n$ is an n-dimensional torus. Note that the above assumes the manipulator is stationary. If it can translate about the world origin then C becomes $\mathbb{R}^{n+2}$ for the case where the joint cannot swing around and $\mathbb{R}^2 \times \mathbb{T}^n$ otherwise. The manipulators that will be used in this thesis, and in fact almost all commercial manipulators, have hard joint limits and hence their $C = \mathbb{R}^n$. Additionally, the manipulators will be stationary about the world frame.

Furthermore, the obstacle region for manipulators is different to the single rigid body case. The reason being that because manipulators have multiple bodies, collisions between these bodies need to be avoided. Let P denote the set of collision pairs $(i, j) \in P$ and the manipulator be modelled as a collection, $\{A_1, A_2, ..., A_m\}$, of m links, then the obstacle region becomes

$$C_{obs} = \left(\bigcup_{i=1}^{m} \{q \in C \mid A_i(q) \cap O \neq \emptyset\} \right) \bigcup \left(\bigcup_{[i,j] \in P} \{q \in C \mid A_i(q) \cap A_j(q) \neq \emptyset\} \right),$$

where $A_i(q)$ is the configuration of link i.

The motion planning problem is then the same as the general case, except that the robot, A, is a collection of m links, $\{A_1, A_2, ..., A_m\}$.

Chapter 3

Problem Statement

The motion planning problem described in Chapter 1 will be formulated here by building on the concepts explored in Chapter 2. In summary, the problem can be condensed as follows: for a given set of tasks and a 3D environment representation, compute a motion plan that will complete these tasks in a fast and efficient manner with high reliability. A key problem is to devise a motion planning framework that satisfies the above criteria whilst overcoming the computational complexities associated with planning for a high-dimensional and highly non-linear system, such as a robotic manipulator, for a specific set of practical applications.

In this chapter, the motion planning problem is decomposed into sub-problems where key challenges and constraints are identified and defined. The implications of these challenges and constraints are later realised in the algorithmic components section, where existing state-of-the-art algorithms used in FREDS-MP are formulated.

3.1 Preliminaries

A 6D pose of a rigid body is represented as a 4x4 transform matrix. The pose of a coordinate frame A is given by the position and orientation of another coordinate frame, B, with respect to A by the affine transform matrix

$$\mathbf{T_{AB}} := \begin{bmatrix} \mathbf{R_{AB}} & \mathbf{t_{AB}} \\ \mathbf{0^T} & 1 \end{bmatrix} \in SE(3),$$

where $\mathbf{T_{AB}}$ is a 3×1 translation vector and R_{AB} is a 3×3 rotation matrix. The pose of the manipulator end-effector with respect to its base frame will be denoted as $\mathbf{T_{0E}}$.

3.2 Sub-problem Overview

Fast, reliable and efficient have traditionally been conflicting objectives for manipulator motion planning. For example, sampling-based planners were introduced as a means to reduce computation time at the cost of path optimality. Trajectory optimisers produce fast and efficient trajectories at the cost of reliability. Grid-based planners produce efficient trajectories reliably but are not fast enough to be considered feasible for high-dimensional problems.

It has been observed in Section 2.2.2 that pre-computed paths could be utilised for improving the computation speed and reliability of trajectory optimisers. Further, given a set of manipulator tasks, pre-computed trajectory costs can provide an accurate objective function for task sequencing. Hence, building on these ideas, the motion planning problem will be split into three distinct sub-problems: offline, task and online planning.

The offline planner aims to pre-compute a comprehensive set of trajectories that can later be utilised by the task and online planner for adapting to given manipulator tasks. In order to pre-compute meaningful trajectories, special cases are constructed in an attempt to anticipate what environments the robot will encounter in the real world.

The task planner intelligently selects a suitable sub-set of pre-computed trajectories based on the current manipulator task and obstacle configuration. The trajectories are modified to satisfy the current task and then a suitable order is computed based on pre-computed trajectory cost information.

The online planner takes the sequenced trajectories and adapts them to the current obstacle configuration. Finally, it processes the adapted trajectories for execution on a real robot arm.

3.3 Offline Planner

The offline planner's role is to generate a comprehensive set of trajectories that can be utilised by the task and online planner. The idea is that having informed priors will increase the computation speed and reliability of the task planner and the performance of online planners. Furthermore, cost information from these priors allows for generation of informed heuristics, which can potentially reduce execution

time significantly.

Critical to the offline planning phase is the construction of special cases, used to anticipate real world scenarios. These special cases are parameterised by a unique obstacle configuration. Let us denote the space of special cases, more easily conceptualised as maps, as M, where each map $m_i \in M, i = 1, ..., n$, and n is the number of maps, or unique obstacle configurations.

For each map m_i, there is an associated obstacle configuration and task space where each task is an abstract representation of a real task, such as moving a camera sensor to a new viewpoint or grasping an object. The challenge here is that the task space is continuous and infinitely large. Hence, there is a need to discretise the task space. Let us denote a discretised task space associated to a particular map as $T_{offline} \in m_i$ and a particular task as $t_i \in T_{offline}, i = 1, ..., n$.

The problem is then to devise a suitable discretisation strategy to select $T_{offline}$ that will minimise computation time without compromising reliability and efficiency when adapting the resulting trajectories online. Storage space and construction time constraints need to be considered when pre-computing trajectories. Similarly, the number of possible obstacle configurations is infinitely large, hence a suitable strategy needs to be developed for representing real world obstacles in a way that is tractable. The offline obstacle configuration is denoted $O_{offline} \in m$.

For each task $t_i \in T_{offline}$, the offline planner needs to: compute suitable trajectories; determine a suitable representation of the trajectory; and consider how to handle redundant solutions for the manipulator tasks. Since these trajectories are computed offline, they will need to be stored in order to utilise them later. An important problem for the offline planner is choosing a suitable method for storing and representing these trajectories, while considering implications for the task planner, such as file size, access speed and ease of lookup.

3.4 Task Planner

The task planner operates in real time, hence its input is an online obstacle configuration O_{online} and a set of online tasks, denoted $\hat{t}_{online}$. For convenience, O_{online} will be referred to as a scene and $\hat{t}_{online}$ a scenario. It is important to distinguish that the online tasks are in continuous state space, in contrast to the offline tasks which

are in a discrete state space.

The first problem is to choose a suitable offline map $m \in M$ given O_{online}. The rest of the processes will depend on $T_{offline} \in m$. Given $\hat{t}_{online}$ choose a suitable set of pre-computed tasks $\hat{t}_{offline} \subset T_{offline}$ as candidates to be adapted to $\hat{t}_{online}$. Adpating involves modifying $\hat{t}_{online}$ such that the following constraints are met: end effector is in correct pose and configuration for each task; optional task specific constraints are not violated, such as fixing the end effector orientation to direction vector; manipulator start and end configurations are in C_{free}.

The sequence in which a robot executes multiple tasks can affect the efficiency of the overall trajectory. To sequence these tasks, edge costs need to be determined between the different tasks. For some systems this is simple, for example Euclidean distance between task positions. However, for a highly non-linear system like a robotic manipulator this is not straight forward to compute. Approximations such as taking the maximum joint displacement between two task configurations can provide a good lower bound heuristic, but this assumes that the goal configurations are known in advanced. One can consider all possible configurations but for multiple tasks the size of the sequencing problem can increase significantly.

The task planner's job is to utilise the pre-computed trajectory cost information, computed by the offline planner, to provide an accurate and fast approximation of the edge costs between tasks prior to knowing exactly what trajectory the manipulator will execute for a particular task. Once these costs are known, the sequencing problem is simply a TSP.

Given that we may have n tasks, the optimisation problem becomes minimising a sequence of trajectory, or action, costs. If the objective function $f(\pi_{ij})$ describes the cost of the trajectory between tasks t_i and t_j, then the minimum task sequence is

$$min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} f(\pi_{ij}) x_{ij},$$

where

$$x_{ij} = \begin{cases} 1, & \text{if } \exists \text{ a trajectory from task } i \text{ to } j \\ 0, & \text{otherwise} \end{cases}$$

subject to

$$0 \leq x_{ij} \leq 1 \qquad i, j = 1, ..., n$$

$$u_i \in \mathbb{Z} \qquad i = 1, ..., n$$

$$\sum_{i=1, i \neq j}^{n} = 1 \qquad j = 1, ..., n$$

$$\sum_{j=1, j \neq i}^{n} = 1 \qquad i = 1, ..., n$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n.$$

The above constraints mean that the sequence must be a Hamiltonian Cycle, i.e. need to execute each task once and only once. Additionally, since only one robot is used, a single tour is needed. It should be noted here that the edges between targets can be considered undirected, since the motion planner only considers kinematic costs.

3.5 Online Planner

Let us denote the sequence of trajectories produced by the task planner as $\hat{t}_{ordered}$. The online planner adapts each of these trajectories to the online obstacle configuration, such that the total cost of each modified trajectory is minimised. Further, the online planner ensures that modified trajectories adhere to the constraints defined in Section 3.3 and 3.4. If a particular trajectory breaks these constraints, the online planner must modify the sequence such that there is no significant impact on reliability, efficiency and computation speed.

Once the trajectories are adapted and valid, the online planner needs to process them so that they are ready to be executed on the manipulator, such that the intended trajectory is followed and kinematic constraints, such as joint position and velocity limits, are not violated.

3.6 Components Algorithms

A wide range of state-of-the-art algorithms were analysed in Chapter 1. Those used in this thesis implementation were deemed the most effective at addressing each of the formulated sub-problems. The algorithms used are formulated in this section.

3.6.1 Grid-Based Algorithms

As described in Section 3.1, task planning abstracts high level tasks by ignoring lower level tasks, such as the motion to perform the task itself. The purpose of the grid-based search is to compute these low level motions so that the motion plans can be utilised by the task and online planner.

Let π_K denote a motion plan that is K steps long. The motion plan is made up of actions or manipulator configurations, i.e. a sequence $\hat{q}_K$ $(q_1, q_2, ..., q_K)$ of K actions. Given inputs q_K (goal configuration) and x_I (initial state), a sequence of states $\hat{x}_{K+1}$ $(x_1, x_2, ..., x_{K+1})$ can be derived using a state transition function f.

The cost functional is

$$L(\pi_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_{k+1}(x_{k+1}),$$

where state space X is finite, $l(x_k, u_k)$ is the cost term and

$$l_{k+1}(x_{k+1}) = \begin{cases} 0, & \text{if} \quad x_{k+1} \in X_G \\ \infty, & \text{otherwise} \end{cases}.$$

The problem is then to find a motion plan π_K that minimises L.

Dijkstra's Algorithm

Dijkstra's algorithm is an optimal discrete planning algorithm that searches over a graph and produces single-source shortest paths. It is known to be asymptotically the fastest single-source shortest path algorithm for arbitrary directed graphs with unbounded non-negative weights.

We will define the transform of the manipulator's end-effector with respect to the base frame of the robot, which is always placed at a map m origin, as $\mathbf{T_{0E}}$. A state, x will map to a unique transform. Hence, the state space X here is the space of end-effector transforms. The graph G_m is a discretised grid overlayed on top of the state space of a particular m.

Each state $x \in G_m$ maps to a manipulator configuration θ that is a solution to the corresponding end effector transform. The configuration space C then becomes all the possible $q \in C_{free}$ that correspond to a state $x \in G_m$.

Dijkstra's algorithm will find a set of motion plans $\hat{\pi}_K$ that result in a set of minimum costs, $\hat{L}$, to move from a source state x_I to all other $x \in X$. The cost $l(x_k, u_k)$ is defined by the Euclidean displacement metric

$$\rho(x_k, x'_k) = \sqrt{\sum_{i=1}^{n} \Big(q(x'_k)_i - q(x_k)_i\Big)^2},$$

where $q(x_k)$ is the current manipulator configuration at state x_k, $q(x'_k)$ is the candidate configuration at x'_k and n is the number of joints. Minimising the Euclidean displacement in joint space corresponds to minimising the total joint displacement.

Another problem is ensuring that the action u_k that results in x'_k is feasible. One constraint is that x'_k needs to have at least one valid manipulator configuration $q \in C_{free}$. The other constraint is ensuring continuous time-safety. While obstacles are explicitly encoded into the graph, the later constraint is not. Hence this must be dealt with.

3.6.2 Sampling-Based Algorithms

In sampling based approaches, the motion planning problem can be seen as a search in C. Let us denote a manipulator configuration as $q \in C$. Then the task is to compute a continuous path from an initial configuration, θ_I to a goal configuration, q_K, without any pre-processing. That is, an explicit representation of C_{free} is not available. Rather, given a $q \in C$ we can test whether $q \in C_{free}$.

Time-continuous safety is another constraint that must be dealt with, i.e. not just the sampled configurations are in C_{free} but the entire manipulator motion between these sampled configurations are as well.

Bi-direction RRT (BiRRT)

BiRRT is a single-query sampling based planner. It uses a bi-directional tree method, where two propagating wavefronts, one centred on q_I and the other on q_K, will meet after exploring a portion of the configuration space.

Similar to RRT, BiRRT avoids the discretisation problems of graph-search techniques by randomly sampling the continuous configuration space C. Hence, the problem is to randomly sample configurations $q \in C$ until the two wavefronts meet.

Once they meet, the motion plan π_K is the sequence of configurations $\hat{q}$ connecting q_I and q_K, subject to the constraint $\hat{q} \in C_{free}$.

Batch Informed Trees (BIT*)

BIT* balances the benefits of graph-search and sample-based techniques. It produces batches of samples over the continuous C and runs an incremental graph search over these samples. Moreover, it is an optimal planner, in contrast with BiRRT, which simply focuses on path feasibility. The optimal solution is the path, $\pi_K \in C_{free}$, that minimises a cost function, L, while connecting q_I to q_K.

Constrained Bi-directional RRT (CBiRRT)

For particular manipulator tasks, there may be a need to constrain the motion in some way to meet an objective. Some examples include always ensuring the end effector co-ordinate frame is upright (moving a cup with liquid in it), constraining the end-effector to a manifold (polishing a surface) constraining the end effector to a direction vector (grasping an object from a cluttered workspace).

CBiRRT aims to provide a general framework for implementing a large variety of trajectory constraints. The algorithm builds on top of BiRRT where the "extend" procedure is modified to include a constraint checking function. Whilst this framework can work with a variety of manipulator constraints, the constraint that this thesis is concerned with is constraining the end effector to a direction vector.

Constraints are evaluated as a function of the robot's configuration, q_s. Using forward kinematics, from q_s we can compute end effector transform $\mathbf{T_{0E}}$, with respect to the map origin. A constraint reference transform, $\mathbf{T_{0C}}$, is also needed. In this problem it would be the direction vector we wish to constrain the end effector to. The constraints are then defined as an inequality matrix

$$\mathbf{CO} := \begin{bmatrix} \mathbf{CO_{x_{min}}} & \mathbf{CO_{x_{max}}} \\ \mathbf{CO_{y_{min}}} & \mathbf{CO_{y_{max}}} \\ \mathbf{CO_{z_{min}}} & \mathbf{CO_{z_{max}}} \\ \mathbf{CO_{\psi_{min}}} & \mathbf{CO_{\psi_{max}}} \\ \mathbf{CO_{\theta_{min}}} & \mathbf{CO_{\theta_{max}}} \\ \mathbf{CO_{\phi_{min}}} & \mathbf{CO_{\phi_{max}}} \end{bmatrix}, \tag{3.1}$$

where the first three rows bound the translation and the remaining three bound rotation of $\mathbf{T_{0E}}$ about the $\mathbf{T_{0C}}$ frame.

We can use this representation to intuitively specify constraints. Further, a displacement from constraint function can be defined that is very fast to compute. This displacement is used to ensure that the trajectory does not violate the given constraints. For more details on how this displacement is computed, see [70].

3.6.3 Trajectory Optimisers

A manipulator motion planning problem can be formulated as a non-convex optimisation problem. That is, minimise an objective function *f(x)* subject to inequality and equality constraints

$$g_i(x) \leq 0, \quad i = 1, 2, ..., n_{ineq}$$

$$h_i(x) = 0, \quad i = 1, 2, ..., n_{eq},$$

where f, g_i, h_i are scalar functions.

Since this is a kinematic motion planning problem, the optimisation is performed over a trajectory, i.e. a $K \times N$-dimensional vector, where K is the number of time-steps, or configurations that make up the trajectory, and N is the number of degrees of freedom. For sake of clarity, a configuration at time-step k will be denoted as q_k.

Hence, given a start and end configuration we wish to find a trajectory that minimises an objective function

$$f(q_{1:K}) = \sum_{k=1}^{K-1} l(q_{k+1}, q_k).$$

More recently, trajectory optimisers take into consideration obstacles by adding an "obstacle" cost to the objective function. The way in which these obstacle costs

are computed are usually dependant on the way the obstacles are represented, which varies between each algorithm.

During optimisation, obstacle avoidance is treated as an inequality constraint but the final trajectory must adhere to the equality constraint, $q_{1:K} \in C_{free}$. Another equality constraint is that the trajectory start and end configurations are fixed to given initial and goal configuration. Joint position and velocity limits are treated as inequality constraints.

All the trajectory optimisers used (CHOMP, TrajOpt and GPMP2) follow the same problem definition above, hence they will not be formulated separately. Details on their implementation can be found in [94] [97] [95] [98] [89] [99] and Section 5.1.5.

Chapter 4

FREDS-MP

In this chapter the implementation details of FREDS-MP are explained in detail. Each of the sub-component problems are addressed and rationales are given for their design. Further, any assumptions made about the function of each sub-component are detailed here. Lastly, the time and space complexity of FREDS-MP are analysed.

4.1 Offline Planner

The offline planner pre-computes trajectories given a map. These pre-computed trajectories are stored and indexed in a database. In order for the pre-computed trajectories to be useful, they are computed based on the real applications that the manipulator will carry out. That means the obstacles and tasks need to be anticipated and modelled. In some environments, particularly agriculture, it is not possible to know exactly what these will look like ahead of time. Hence, the offline planner will implement strategies to define these in a useful way. This section will describe how these maps and databases are represented and constructed.

4.1.1 Obstacle Representation

Obstacles will be represented as a union of basic 3D shape primitives. For example, in Figure 4.1 an apple trellis can be represented as a box. The shape and size is chosen such that a bounded volume is formed around the real obstacle. This bounded volume is designed such that it forms a tight fit. A tighter fit will result in a larger C-free region and hence more room for the manipulator to work around. Further, the pre-computed trajectories are guaranteed to be collision free when adapted to the real obstacle.

In the real world the obstacles will vary, so one strategy could be to choose different discrete sizes of the volumes and distance offsets. Each map $m \in M$ will then correspond to a particular discretised obstacle configuration. In this thesis'

Figure 4.1 : Example of using bounded volumes to represent real world obstacles.

implementation the bounded volumes will be defined manually for a range of obstacle configurations. However, a strategy could be developed to create these procedurally.

4.1.2 Task Space Representation

An offline discretised task space, $T_{offline} \in m$ is constructed by overlaying a grid over X, defined to be the space of end effector transforms. For the tasks we are concerned with in this thesis, X will be defined as a 2D manifold. For example, for the active perception application, X is chosen to be a unit sphere, $\mathbb{S}^2$, and $T_{offline}$ is a grid of normal vectors on $\mathbb{S}^2$. This is visualised in Figure 4.2. The result is a set of poses where the end effector, typically a camera sensor, is always pointing towards a common centre point. This is a reasonable representation, since typically in an active perception problem the aim is to sample different views of a common object.

Another application this thesis is concerned with is inspecting fruit on a trellis, where the end effector will move a sensor close up to the fruit. The state space in this case would be a 2D plane parallel to the trellis and $T_{offline}$ would be a grid of vectors, facing into the trellis, as can be seen in Figure 4.2(a). Because the end effector will be in close proximity to the trellis, a common strategy is to approach the fruit using a "stroking" motion where the end effector is given a pose offset from its target and then moves "in and out" in a linear motion along a direction vector. Since the trellis is cluttered with leaves and branches, the stroking motion will mitigate the arm and end effector being caught on parts of the trellis, such as branches and wire. It will also avoid explicit and complicated modelling of the

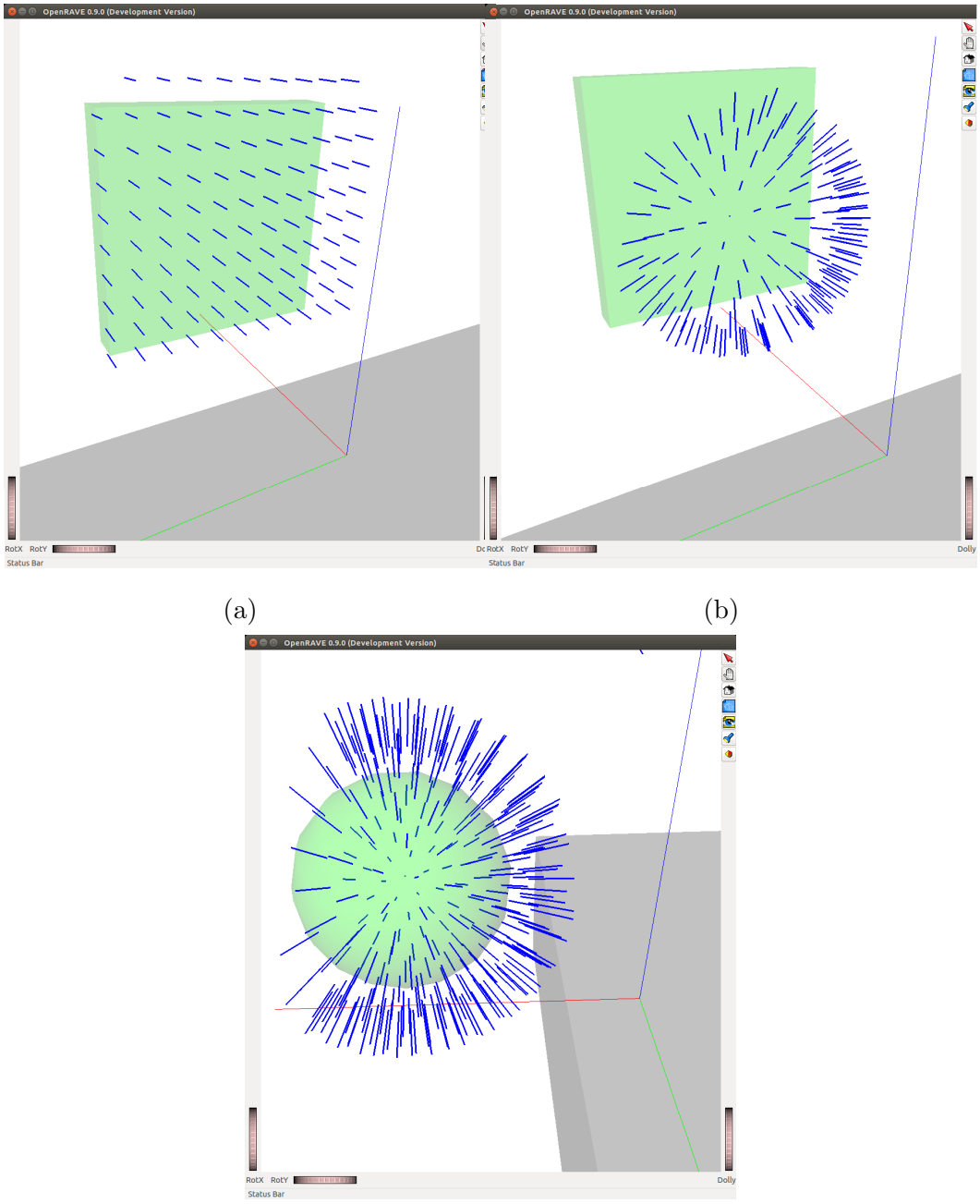


Figure 4.2 : Task space discretisation for various obstacles and manifolds: (a) normal vectors on a plane parallel to a planar obstacle; (b) normal vectors on a spherical manifold centred about a planar obstacle; (c) normal vectors on a spherical manifold centred about a spherical obstacle

cluttered trellis.

This type of task can still be encoded by the grid, where the normal vector is the offset described above and the direction of the vector is the direction of the

stroking motion. The constrained stroking motion needs to be encoded into the task as well. To facillitate generality, the constraint will be specified using the inequality matrix in equation 3.1. In this way, any task that can be specified using an end effector transform and the inequality matrix will integrate seamlessly into FREDS-MP. In the case of the fruit inspection task, the inequality matrix can be described as follows: first the object frame is placed at the end effector with the z axis pointed in the direction of the end effector; then an epsilon is assigned to each c_{min}, c_{cmax}, except for $c_{z_{min}}, c_{z_{max}}$ which is assigned the stroke distance.

In this thesis' implementation the stroke distance will be constant, however many different tasks could use different stroke distances, for example grasping tasks could be encoded in the same way with a slightly longer stroke distance. These stroking tasks are what have been referred to as constrained tasks.

4.1.3 Database Structure

The database is structured such that looking up candidate paths in the task planner is efficient and produces the relevant information needed to complete the task and online planning phase. Hence, a dictionary type data structure is used for looking up trajectories. The trajectory dictionary structure is visualised in Figure 4.3(a).

The first problem is determining how to index each task. Because the task space is discretised into a grid it make sense to index each task as its node in the grid. Abstracting tasks as nodes makes looking up tasks quicker and easier and is a useful, compact representation for other tasks, such as task planning.

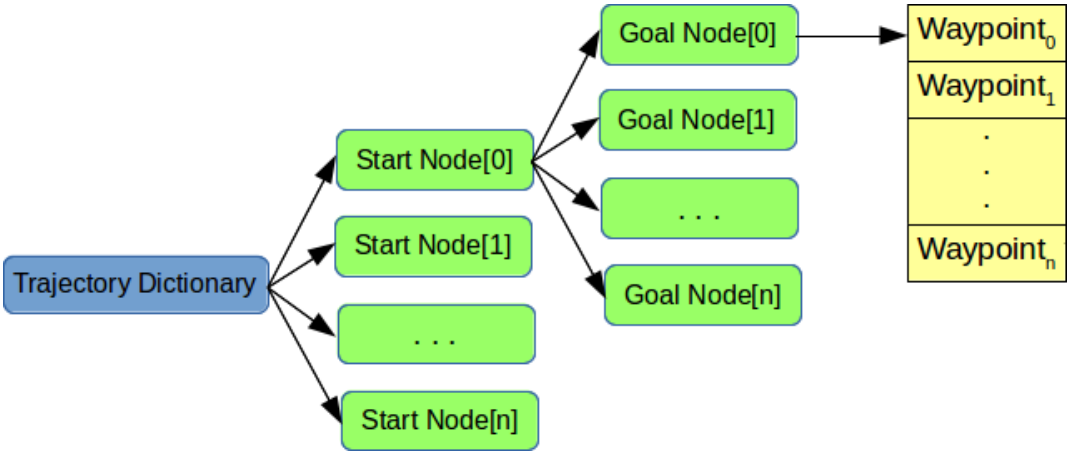
Each node is then mapped to a task which is encoded as a workspace postion, "Task Position", and a point along the direction vector, "Look At", as visualised in Figure 4.3(b). This is useful because it provides more information about the task rather than just simply an orientation. For example the Look At point encodes both the transform rotation as well as task specific details, for example a camera focus point or a stroking distance for constrained tasks. If a task is constrained, this will be encoded as a constraint matrix, see Equation 3.1, in "Constraints".

For each node, or task, in the grid there is a list of trajectories, made up of waypoints or in the context of this thesis, manipulator joint angles, describing how to execute a desired task, "Goal Node", upon completing a previous task, "Start

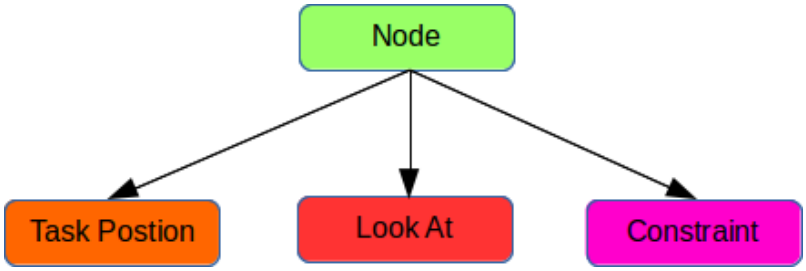
Node". If a particular task cannot be executed starting upon the completion of another task then there will be an empty trajectory for that task pair. In addition to trajectory look up, costs corresponding to those trajectories need to be looked up. Hence, the cost dictionary has the same structure as the trajectory dictionary except instead of a list of waypoints for each start and end node pair, the cost of those trajectories is stored.

4.1.4 Database Construction

The idea behind the offline planner is to remove redundant configurations for each task in the task space $T_{offline} \in m$ or $v \in G_m$ by finding a unique solution for each v that minimises the total edge costs of G_m. This is done implicitly by pruning off configurations, according to line 8 in MOD_DIJKSTRA. The rationale behind



(a)



(b)

Figure 4.3 : Database structure, (a) shows the trajectory dictionary structure and (b) shows the task encoding for each node.

generating a set of unique solutions is to greatly reduce the computation time that would otherwise be required by the task planner.

Another objective is to minimise the number of disconnected nodes. Here a disconnected node does not necessarily mean that a trajectory could not be found, rather an edge could not be created due to the constraint in line 8 in MOD_DIJKSTRA, i.e. there was no manipulator configuration that was close in C. There are two reasons for this objective; the first is that disconnected nodes will have a large cost and the second is that computing these trajectories is time consuming because it needs to be planned with an online motion planner to ensure time continuous safety. If every trajectory needed to be computed this way, then the offline construction would be infeasible.

One implication of this construction method is that for the trajectories generated by MOD_DIJKSTRA there is no guaranteed time-continuous safety. Rather there is a "soft" guarantee that no adjacent configuration in the trajectory will be far from each other in C, due to the constraint in line 8. Further, the nodes in G are designed to be no further apart, in *SE(3)*, than the width of the manipulator's end effector link. This is deemed to be acceptable because the paths are later adapted online where time-continuous safety is guaranteed. In this way, the trajectories and costs constructed in the offline database are intended to act as heuristics rather than be used in their raw form.

Another implication of this method is that when planning trajectories in the task and online planning phase, the manipulator configurations are limited to those unique solutions found during the offline planning phase. While this effectively removes the benefits of redundancy at runtime, it greatly reduces the complexity of the task planning which arguably has more of an impact on performance. This trade-off between task planning efficiency and manipulator redundancy during online planning will be realised later in Chapter 5.

The GENERATE_DATABASE procedure is the entry point to the offline planner and takes as input: a map m, containing the task space graph G_m, and obstacle representation O_m. The procedure then performs the following:

i. **Lines 2-4:** Compute all valid IK solutions, $\hat{q}_v \in C_{valid}$, for T_{0E} at each node v in the graph G_m, where C_{valid} means the configurations are in C_{free} and do

not violate any given custom constraint CO.

ii. **Lines 5-9:** For each valid IK solution, $q_v, v \in G_m$, perform MOD_DIJKSTRA with q_v as starting configuration.

iii. **Line 10:** Find q_v that minimises the number of disconnected nodes:

$$\min_{q_v, v \in G_m} \sum_{i \in G_m} \sum_{i \neq j, j \in G_m} x_{ij},$$

where

$$x_{ij} = \begin{cases} 0, & \text{if } \exists \text{ an edge from node } i \text{ to } j \\ 1, & \text{otherwise} \end{cases}.$$

iv. **Lines 11-13:** For each $v \in G_m$, using the modified database D resulting from optimal q_v from Line 10, perform MOD_DIJKSTRA to compute trajectories to all other nodes in G_m. The trajectories and their corresponding costs are stored in the database.

v. **Lines 14-18:** For each disconnected $v \in G_m$, compute trajectories, $\pi_K \in C_{free}$ between v and every other node $u \in G_m$ using an online motion planner.

The MOD_DIJKSTRA procedure takes as input: a Graph G, a database of IK solutions D, a start node v_i, and a start configuration q_i. The procedure then performs the following:

i. **Lines 2-3:** Initialises priority queue, Q, with v_i and creates a copy of D.

ii. **Line 4:** Replaces all IK solutions for start node in D_{copy} with q_i

iii. **Line 6-8:** For each neighbour v of u, find the configuration $q \in \hat{D}_{copy}[v]$ with minimum "Maximum-Coordinate-Difference Distance" cost to node u with upper bound ϵ. This minimum q is stored in $q_m in$ and the corresponding cost is stored in $alt[v]$.

iv. **Lines 9-12** If $alt[v]$ is lower than current best score, $cost[v]$, then update parents and cost. Also, replace $D_{copy}[v]$ with q_{min}.

Algorithm 1 Offline Planner Main

1: **procedure** GENERATE_DATABASE(m)
2: **for** *each* v *in* $m.G$ **do**
3: $D \leftarrow \hat{q}_v \in C_{valid}$
4: **end for**
5: **for each** v in $m.G$ **do**
6: **for each** q in $D(v)$ **do**
7: $solution[v, q] \leftarrow$ MOD_DIJKSTRA(m.G,D,v,q)
8: **end for**
9: **end for**
10: $min\_solution \leftarrow$ solution with min disconnected nodes
11: **for each** v in $m.G$ **do**
12: $database[v] \leftarrow$ paths(MOD_DIJKSTRA($m.G, min\_solution.D, v, min\_solution.D[v]$))
13: **end for**
14: **for each** disconnected v in $m.G$ **do**
15: **for each** u in $m.G, u \neq v$ **do**
16: $database[v] \leftarrow$ plan(v, u)
17: **end for**
18: **end for**
19: **end procedure**

v. **Line 16:** The *parents* and modified D data structures are returned. The *parents* data structure is used to trace back the paths generated. The modified D data structure contains the unique solutions that were computed.

Algorithm 2 Modified Dijkstra

1: **procedure** MOD_DIJKSTRA(G, D, v_i, q_i)
2: $Q \leftarrow v_i$
3: $D_{copy} \leftarrow D$
4: $D_{copy}[v_i] \leftarrow q_i$
5: **while** !Q.empty() **do**
6: $v \leftarrow$ node in Q with min $cost[v]$
7: **for each** neighbour u of v **do**
8: $q_{min}, alt[u] \leftarrow \arg\min_{q \in D_{copy}[u]} \max(\| \theta_i - \theta_j \| \textbf{ for } \theta_i, \theta_j \in D_{copy}[v], q \mid \forall \| \theta_i - \theta_j \| < \epsilon\})$
9: **if** $alt[u] < cost[u]$ **then**
10: $D_{copy}[u] \leftarrow q_{min}$
11: $cost[u] \leftarrow alt[u]$
12: $parents[u] \leftarrow v$
13: **end if**
14: **end for**
15: **end while**
16: **return**$(parents, D_{copy})$
17: **end procedure**

4.2 Task Planner

The task planner takes as input a scene O_{online}, and scenario $\hat{t}_{online}$, then chooses an appropriate $m \in M$. This thesis assumes that map selection will be performed by another process. Hence, the appropriate pre-computed database will be chosen manually. Further, the scene will simply be the offline obstacle configuration, $O_{offline} \in m$. These assumptions have no impact on the validity of the results, since $O_{offline}$ are bounded volumes.

The task planner has three main functions: searching the database for candidates $\hat{t}_{offline}$ that match closely with $\hat{t}_{online}$, modifying $\hat{t}_{offline}$ to satisfy $\hat{t}_{online}$ constraints and additional costs, and lastly computing an efficient task sequence. These functions are explained in detail below.

4.2.1 Database Search

The database search tries to find the best candidate tasks from the pre-computed database, given some online tasks $\hat{t}_{online}$. In this context "best" means the task that minimises a heuristic that quanitfies the difference between the tasks being matched. The database search methods in [10] [9] use the current manipulator configuration and the set of IK solutions for a particular task as the input. This is reasonable for singleton tasks, however, since the aim is to sequence multiple tasks, we do not know what the subsequent start configurations are until we know the sequence.

To overcome this issue, the input to the search is the task's coordinates in SE(3), since this is independent the sequence of tasks. Further, because the offline database has already optimised the configuration for each task, the sequencing problem is greatly reduced. The search finds n candidates $\hat{t}_{offline} \subset T_{offline}$ that are nearest to $t_{online} \in \hat{t}_{online}$ in SE(3). From this subset the best candidate is picked based on which corresponding unique configuration $q_{t_{offline}}$ has the minimum Euclidean distance, in C, to any of the IK solutions $Q_{t_{online}}$.

The SEARCH procedure takes as input a set of online tasks $\hat{t}_{online}$, and for each task within this set performs the following:

i. **Line 3:** Store all IK solutions for current task t_{online} into $Q_{t_{online}}$.

ii. **Line 4:** Using $T_{offline} \in m$ find n nearest neighbours of t_{online} using distance in $SE(3)$ as a heuristic.

iii. **Line 5:** Of these nearest neighbours, find the one with the smallest Euclidean distance in C to any configuration $q_{t_{online}} \in Q_{t_{online}}$. Store nearest neighbour into *candidates* and corresponding $q_{t_{online}}$ into *min_IK*.

Algorithm 3 Database Search

1: **procedure** SEARCH($\hat{t}_{online}$, m)
2: **for each** t_{online} in $\hat{t}_{online}$ **do**
3: $Q_{t_{online}} \leftarrow \text{IK}(t_{online}) \in C_{free}$
4: $\hat{t}_{offline} \leftarrow \text{nearest}(t_{online}, m.T_{offline}, n)$
5: $best\_candidates[t_{online}], min\_IK[t_{online}] \leftarrow \arg\min_{t_{offline} \in \hat{t}_{offline}} \parallel q_{t_{online}} - q_{t_{offline}} \parallel$ **for** $q_{t_{online}} \in Q_{t_{online}}$
6: **end for**
7: **return** $best\_candidates, min\_IK$
8: **end procedure**

4.2.2 Task Modification

After the best candidate tasks have been retrieved, they need to be modified to meet the constraints of the online tasks. This means that the database trajectories and their corresponding costs also need to be modified. Up to this point, the best candidate offline task for each online task has been found, but the order in which to perform each task is still not known. In order to retrieve a trajectory from the database, two tasks are needed: the next task and the current task it is transitioning from. Because we do not know the sequence yet, this means that every possible sequence needs to be considered. The MOD_CANDIDATES procedure details this modification process.

4.2.3 Sequencing

Since each task has a single configuration, we can use a standard TSP solver to optimise the sequence. Of course, this is an NP-hard problem, hence for large problems the solution will be an approximation within some factor of the optimal. However, in the context of this thesis where the largest problem is 20 tasks, exact solutions are feasible. The TSP solver takes as input a weighted adjacency matrix, where an entry (i, j) corresponds to the cost of getting from task i to task j. Because of the modification of the database trajectories and the fact that the database trajectories themselves are not time-continuously safe, the task assumes that a valid trajectory

Algorithm 4 Modify Candidates

procedure MOD_CANDIDATES($best\_candidates, min\_IK$)
  for each u in $best\_candidates$ **do**
    for each $v \neq u$ in $best\_candidates$ **do**
      $paths[u, v] \leftarrow$ concatenate($min\_IK[u], database.trajectory[u, v], min\_IK[v]$)
      $costs[u, v] \leftarrow$ joint_diff($u, database.trajectory[u, v][0]$) + $database.cost[u, v]$ + joint_diff($v, database.trajectory[u, v][-1]$)
    end for
  end for
  return $paths, costs$
end procedure

exists between each of the tasks. The reason is that it would be computationally expensive to check this for every task-to-task trajectory combination. This is a reasonable assumption given that the configurations appended to database trajectories are collision free and chosen such that they are close in C. This process is summarised in the SEQUENCE procedure.

Algorithm 5 Sequence Tasks

procedure SEQUENCE($\hat{t}_{online}, paths, costs$)
  for each u in $\hat{t}_{online}$ **do**
    for each $v \neq u$ in $\hat{t}_{online}$ **do**
      $adjacency\_mat \leftarrow costs[u, v]$
    end for
  end for
  $sequence \leftarrow$ TSP_solve($adjacency\_mat$)
  return $sequence$
end procedure

4.3 Online Planner

The online planner has three main objectives: the first is to ensure that the modified trajectories from the task planner are time-continuously safe, the second is to optimise the modified trajectories and the third is to ensure that the trajectory is executable on a real manipulator. In this section, path adapting describes how the first two objectives are achieved and trajectory execution describes how the third is achieved.

4.3.1 Path Adapting

The path adapting iterates through each of the adjacent task pairs in the sequence produced by the task planner and optimises their trajectories. It does so by passing these trajectories as initial "seeds" to a suite of trajectory optimisers. In constrast to [10] [9], these trajectory optimisers attempt to "push" the trajectory away from obstacles whilst optimising their cost. Hence, they will not only adapt to real-time obstacles but they will also have a higher chance of succeeding, even if the online tasks are very different to the selected offline tasks. The resulting optimised trajectories are then checked for time-continuous safety. If a trajectory is in collision, a sampling based planner is used as a last resort, which plans from scratch.

Algorithm 6 Path Adapting

```
procedure ADAPT(sequence, paths)
    for each index in sequence do
        adapted_path ← optimise(sequence[index], sequence[index + 1])
        if adapted_path ∉ C_free then
            sequence[index + 1].remove()
            goto 3
        end if
        trajectory.append(adapted_path)
    end for
    return trajectory
end procedure
```

Because time-continuous safety was only assumed in the task planner, it is possible that when adapting the paths the resulting trajectories can be in collision. If this occurs, then the computed sequence is no longer valid. The online planner deals with this by removing the task in the sequence that failed and attempts to use the next one instead, remembering that the *paths* data structure contains trajectories for all the task combinations. The path adapting process is summarised in the ADAPT procedure.

4.3.2 Trajectory Execution

The motion plans generated by the path adapter are geometric, that is they have no timing associated with them. Hence, these trajectories are time parameterised so that none of the manipulator's individual joint velocity limits are exceeded. Further, the trajectories are usually sparse, which is necessary for reducing computation time, however in order for the manipulator to follow the trajectory closely it should be given a more dense set of waypoints. Hence, the trajectory is upsampled before being sent to the manipulator. Time parameterisation is computed assuming max constant joint velocity between waypoints. Upsampling is performed by linearly interpolating between each of the waypoints.

The control input to a manipulators' joints is torque. The translation of desired joint configurations into torque commands is dependent on the manipulator, however, most modern commercial manipulators, including the Sawyer and UR5, have software that abstracts this control layer by taking a list of time-stamped joint configurations and interpolating between them to produce a smooth torque response. Hence, the final output from the online planner is a list of time-stamped waypoints that is used to execute the tasks on a real manipulator.

4.4 Complexity Analysis

The time and space complexity of FREDS-MP will be analysed in this section. The analysis will be performed per sub-component.

4.4.1 Offline Planner

The Offline Planner has four distinct phases: computing all valid IK solutions for each node in the grid; finding the best unique IK solution for each nodegrid; computing and storing minimum trajectory solutions into a database; and generating and storing trajectories for "disconnected" nodes.

Generating IK solutions for each node in the grid is performed prior to running Dijkstra's algorithm. This speeds up the construction process dramatically, since the grid never changes and the IK solutions will be the same for every run of Dijkstra's algorithm. Hence, when Dijkstra's algorithm is run it uses memoization to quickly lookup valid IK solutions when computing path costs. The IK generation step's worst case time complexity is $O(|V||K|)$ where $|V|$ is the number of vertices in the grid and $|K|$ is the cardinality of the largest IK solution set. The space complexity is also $O(|V||K|)$ in the worst that case every IK solution for every node is valid.

Generally, the most computationally expensive step in Modified Dijkstra is finding the closest solution for every neighbouring set of IK solutions. For each node, the first time it explores its neighbours, it will check all possible valid IK solution, which for a redundant seven-DOF manipulator can be a large number of solutions. Since Dijkstra's algorithm uses a min-priority queue implementation, the worst case time complexity for each unique start configuration is then $O(|E||K| + |V| \log |V|)$ where $|E|$ is the number of edges in the graph. Since Dijkstra's algorithm is run for every valid unique IK solution for every node in the grid, the total time complexity is $O((|E||K| + |V| \log |V|)|V||K|)$. Since only the best Dijkstra's algorithm solution is kept for every run, the worst case space complexity is the same as for a single run of Dijkstra's algorithm which is $O(|V|^2)$ where $|V|^2$ is for the cost matrix.

When the best unique IK solutions for each node has been found, then $|K|$ simply becomes one for each node. The worst case time complexity is then simply $O((|E| + |V| \log |V|)|V|)$. Since every trajectory and its corresponding cost gets stored into the database, the worst case space complexity is $O((|V||T|)^2 + |V|^2)$ where $|T|$ is the largest number of waypoints of any single trajectory.

For the final phase, the best time complexity is simply $\Omega(|D||V|)$ where D is the number of disconnected nodes. The best space complexity is then $\Omega(|D||V||T| + |D||V|)$.

In summary, the time complexity of the offline planner $O((|E||K|+|V|\log|V|)|V||K|)$ because this dominates the other phases. It should be noted that if the initial computation of IK solutions phase needs to consider task constraints then this can cause a significant increase in computational cost. Secondly, if there is a large number of disconnected nodes, then the final phase of the offline planner can also significantly increase computational cost. The total space complexity is $O(|V||K|) + O(|V|^2 + |V|) + O((|V||T|)^2 + |V|^2) + \Omega(|D||V||T| + |D||V|)$

4.4.2 Task Planner

The Task Planner has three distinct phases: database search, task modification and sequencing. For the database search phase, the nodes in the grid and all their unique solutions are stored in a KD Tree data structure. Hence the lookup time for n nearest neighbours is $O(n\log|V|)$. Similarly, to find the best candidate out of these nodes with the closest IK solution is $O(|K_{online}|\log|K_{offline}|)$. To find the best candidates, the IK solutions for the online tasks need to be computed at the time of the search. Therefore, the total time complexity for the database search is $O((n\log|V| + \log|K_{offline}|)|t| + |K_{online}|)$ where $|t|$ is the number of online tasks. Space complexity is $O(|V| + |K_{offline}||V| + |K_{online}|)$ since the KD Trees for all grid nodes are generated at the time the database is loaded.

The task modification process is quite fast because all the IK solutions and best candidates have been computed in the the search phase. Hence, the task is to simply retrieve the corresponding trajectories from the database, which is almost instantaneous because it is pre-loaded into RAM, and insert the computed IK solutions into the trajectories. The time complexity is then $\Omega(|t|^2)$ since every sequence needs to be considered up to this point. The space complexity is $O(|T||t|^2 + |t|^2)$ which considers storage of both task trajectories and costs, where $|T|$ is the largest number of waypoints for any single task trajectory.

The sequencing phase first requires populating the adjacency matrix which is space and time complexity $\Omega(|t|^2)$. The time and space complexity of the TSP solver is dependent on the implementation, however, it will generally always scale based on the size of the adjacency matrix.

Depending on the TSP solver used, it is usually the dominating factor for the

time complexity of the Task Planner. However, if $|t|$ is small then the database search dominates, $O((n \log |V| + \log |K_{offline}|)|t| + |K_{online}|)$. The total space complexity is $O(|V| + |K_{offline}||V| + |K_{online}|) + O(|T||t|^2 + |t|^2) + \Omega(|t|^2)$. It should be noted that the total space complexity should also include the size of the database, since it can be pre-loaded for fast lookup. Databases are typically tens of megabytes in size and should easily be pre-loadable into RAM.

4.4.3 Online Planner

The Online Planner time complexity scales off the number of online tasks, since a single sequence has been determined, and is hence $O(|t|)$. Similarly, the space complexity is $O(|t||T|)$.

Chapter 5

Simulation Experiments

In this chapter, the simulation environment setup in OpenRAVE will be explained. Further, the simulation experiments for both the Sawyer and UR5 arms will be described, including how FREDS-MP will be benchmarked against state-of-the-art motion planners BIT* and RRTConnect. Results will be presented and the performance of FREDS-MP will be analysed. Further, it will be determined which manipulator is more suitable for the intended application.

5.1 Simulation Environment

The Sawyer and UR5 arms and their environment are simulated in OpenRAVE [100] [101], an Open Robotics Automation Virtual Environment for developing and testing motion planners. OpenRAVE can import common robot model file formats, such as COLLAborative Design Activity (COLLADA) and Unified Robot Description Format (URDF), allowing seamless integration with all of its interfaces, including motion planning libraries, inverse kinematics solvers and collision checkers. The motion planners used are OpenRAVE planner plugins: TrajOpt [94] [97], GPMP2 [95] [98], CHOMP [89] [99], CiBRRT [70] [102] and OMPL's [103] RRTConnect [37] and BIT* [87]. Further, one of the strengths of OpenRAVE is its ability to robustly run multiple environments simultaneously in the same process. Hence, parallel motion planning is well supported.

This thesis utilises PrPy [104], a Python library developed by the Personal Robotics Laboratory at Carnegie Mellon University. This library provides robot-agnostic utilities that help implement the above planners as well as run them in parallel with minimal effort.

5.1.1 Robot Model

Both the Sawyer and UR5 have available open source mesh models and URDF files [105] [106]. These are used to visualise the robot geometry in OpenRAVE as well as describe the kinematic and dynamic properties of the robots, however dynamic properties are not used in the simulations. The robot description files also specify the collision model of the robot, which are mainly made up of cylinders and spheres that bound the various links of the robots.

The URDFs were modified to include the Intel Realsense camera and mount geometry. The end effector frame is also defined to be in the Realsense camera frame so that the IK solver produces the correct solutions. An OpenRAVE plugin [107] is used to import the URDF files into OpenRAVE since they are not natively supported.

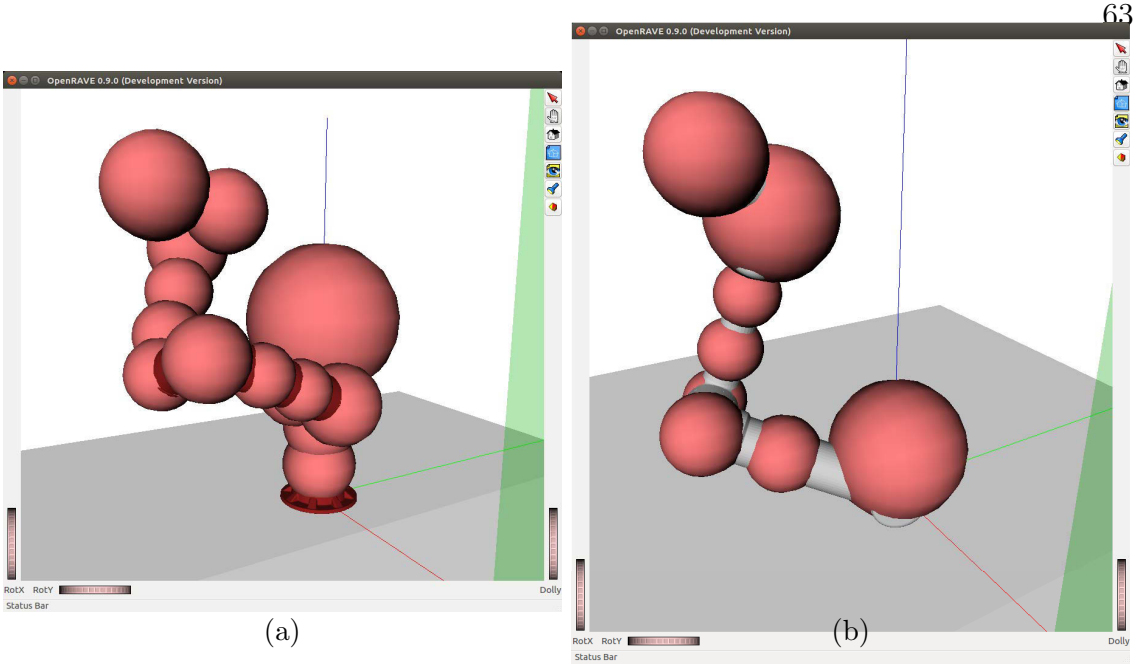
5.1.2 Robot Kinematics

OpenRAVE tracks the state of the robot through its KinBody class which is a kinematic body of links and joints, defined in the URDF file. The KinBody class provides several useful functions, such as setting and getting joint values, setting and getting transformations of all links, and getting velocities of each joint or link.

OpenRAVE provides an analytical IK solver called IKFast which natively supports any imported manipulator configuration up to seven-DOF. Hence, once the URDF is imported into OpenRAVE, IKFast can be run with minimal setup effort to generate a database which can be used during motion planning for fast IK solution queries.

5.1.3 Obstacle Representation

Obstacles are defined via OpenRAVE's XML file format which can define a range of basic shape primitives, such as boxes, cylinders and spheres. The translation and rotation of the obstacles relative to the world frame can be set in the XML file or during runtime within the OpenRAVE environment. They are imported into the OpenRAVE environment as an instance of the KinBody class and hence provide all the functionality described for the robot kinematics.



(a) (b)

Figure 5.1 : Visualisation of (a) Sawyer and (b) UR5 link-sphere approximations for CHOMP and GPMP2 planners in OpenRAVE.

5.1.4 Collision Checking

OpenRAVE comes with a range of built in collision checker plugins. However, the Flexible Collision Library (FCL) [108] is used because it is the fastest out of them and supports the shape primitives, described in Section 2.3.2, that are used to define obstacles and the robot collision model.

5.1.5 Motion Planner Pipeline

The CHOMP and GPMP2 motion planners utilise a signed distance field (SDF) representation of the environment. To produce this SDF, the link-sphere approximations need to be manually defined. The link-sphere approximation for the Sawyer and UR5 arms are visualised in Figure 5.1.

For TrajOpt, CBiRRT, BIT* and RRTConnect, no additional modelling was required. Rather, once the robot is imported into OpenRAVE these motion planners worked out of the box.

5.1.6 TSP Solver

The TSP described in Section 4.2.3 is solved using OR-Tools vehicle routing library [109]. This library provides a constraint programming solver designed for the

general vehicle routing problem or more specifically in our case the TSP. The default solvers and heuristics for the routing model are used.

5.2 Experiments

In this section the aim of the experiments will be explained with reference the thesis' research objectives. Further, the methodology will be described and results will be presented and discussed. The manipulator most suitable for the intended application will be determined in this section.

5.2.1 Aim

The purpose of the software simulation tests is to validate the performance of FREDS-MP by benchmarking it against other state-of-the-art motion planners (Research Objective i). By doing so in simulation, a large number of test scenarios can be evaluated more quickly than if they were to be run on the real hardware. Further, multiple robot arms can be tested and evaluated without needing the actual hardware (Research Objective ii and iii).

Multiple environments with different obstacle configurations can also be set up and interchanged seamlessly to test how the planners perform in varying environment complexities (Research Objective iv). A varying number of tasks per scene will also be tested to see how FREDS-MP scales compared to the other planners (Research Objective iv).

To facilitate generality, one set of simulations will not be application specific (Research Objective ii). Rather, the task of the planner will be to move the manipulator to a sequence of 6D poses without any constraints, other than kinematic and environmental, on how it should move there. The second set will consist of constrained tasks, described in Section 4.1.2.

It should be noted that any arbitrary constraints would be straight forward to implement with the current FREDS-MP framework (Research Objective ii). That is, the planners used in the current implementation's pipeline can incorporate various constraints, such as fixing the end effector to a 2D plane. One could even swap out/ add any constraint planner of choice.

5.2.2 Assumptions

The scenes tested in the experiments are based on challenging agricultural environments and exhibit typical conditions where trajectory optimisers fail due to getting "stuck" in local minima. Hence, they were not used in the simulations to benchmark against FREDS-MP. In contrast, BIT* and RRTConnect are probabilistically complete and can handle these challenging environments.

Further, the placement of the robot arm base relative to the obstacles in the experiments is assumed to be reasonable given the width of a typical orchard and the size of an autonomous agricultural robot.

5.2.3 Methodology

In these experiments FREDS-MP is tested against state-of-the-art motion planners BIT* and RRTConnect. These planners will not use any prior database information when planning. An implication of not having prior trajectory costs is that there is no simple or efficient method for the task planner to generate accurate weights between tasks for sequencing. The reason is that, unlike FREDS-MP, an optimised single start and end manipulator configuration for a task pair is not known in advanced. Hence, every IK solution for each task would need to be considered at runtime. Further, even if the Euclidean configuration space difference between tasks were to be used as edge weights, this would not be as accurate as using the actual path costs. Considering the fact that the Sawyer arm can have hundreds of unique IK solutions for a single task, this would be unacceptably slow to compute.

Therefore, it was decided that using the Euclidean workspace distance as edge weights between tasks was a suitable and fair way to compare the performance of the planners. Further, the manipulator configuration used for each task is computed differently to FREDS-MP. The task and online planning method for these planners are described below.

i. Take as input a list of tasks and compute IK solutions for each.

ii. Remove any tasks that have no valid IK solutions.

iii. Compute weighted adjacency matrix using Euclidean workspace distance between task coordinates as edge weights.

iv. Assign a single IK solution for each task by minimising the Euclidean distance in configuration space between adjacent tasks in the sequence, starting with the robot arm's current configuration and iterating through the sequence.

v. Iterate through the task sequence and compute trajectories between adjacent tasks using the start and end configurations computed in Step iv.

vi. If computing a path fails for a task pair, remove the task with the greater index in the sequence.

The scenes used in the simulations are visualised and labelled in Figures 5.2 and 5.3. The scenes were chosen to test how FREDS-MP performs in a diverse range of obstacle configurations and task space manifolds. Figure 5.2 (a) is a simple scenario, where as Figures 5.2 (b) and (c) are difficult because they have narrow passages that need to be planned through. Figures 5.2 (a) and (b) are representative of the intended application which will be evident in Chapter 6, where as the intent of figure 5.2 (c) is to simulate another realistic application and to demonstrate that FREDS-MP can work with any arbitrary obstacle configuration and task space manifold. The "Branch" configuration was inspired by the fact that many manipulator harvesters reviewed in Section 2.1 failed their tasks because the end effector collided with obstructing branches.

For each scene, scenarios are generated using uniformly random sampled normal vectors over a continuous manifold depending on the task, as described in Section 4.1.2. The tests start with two tasks and are gradually increased in order to test how well the planners scale. Example scenarios are visualised in Figures 5.4 - 5.9 for each scene.

The above process is repeated for constrained tasks, however the random generation of these tasks is different. These tasks are parameterised by a 6D pose and a direction vector. Poses are uniformly random sampled normal vectors over a continuous plane parallel to the obstacle. The direction vector is then the direction of this normal vector.

The constrained task is broken into two motions. The first is moving the manipulator from its current pose to the 6D pose described above. For this first motion there is no constraint on how the manipulator should get to there, other than being

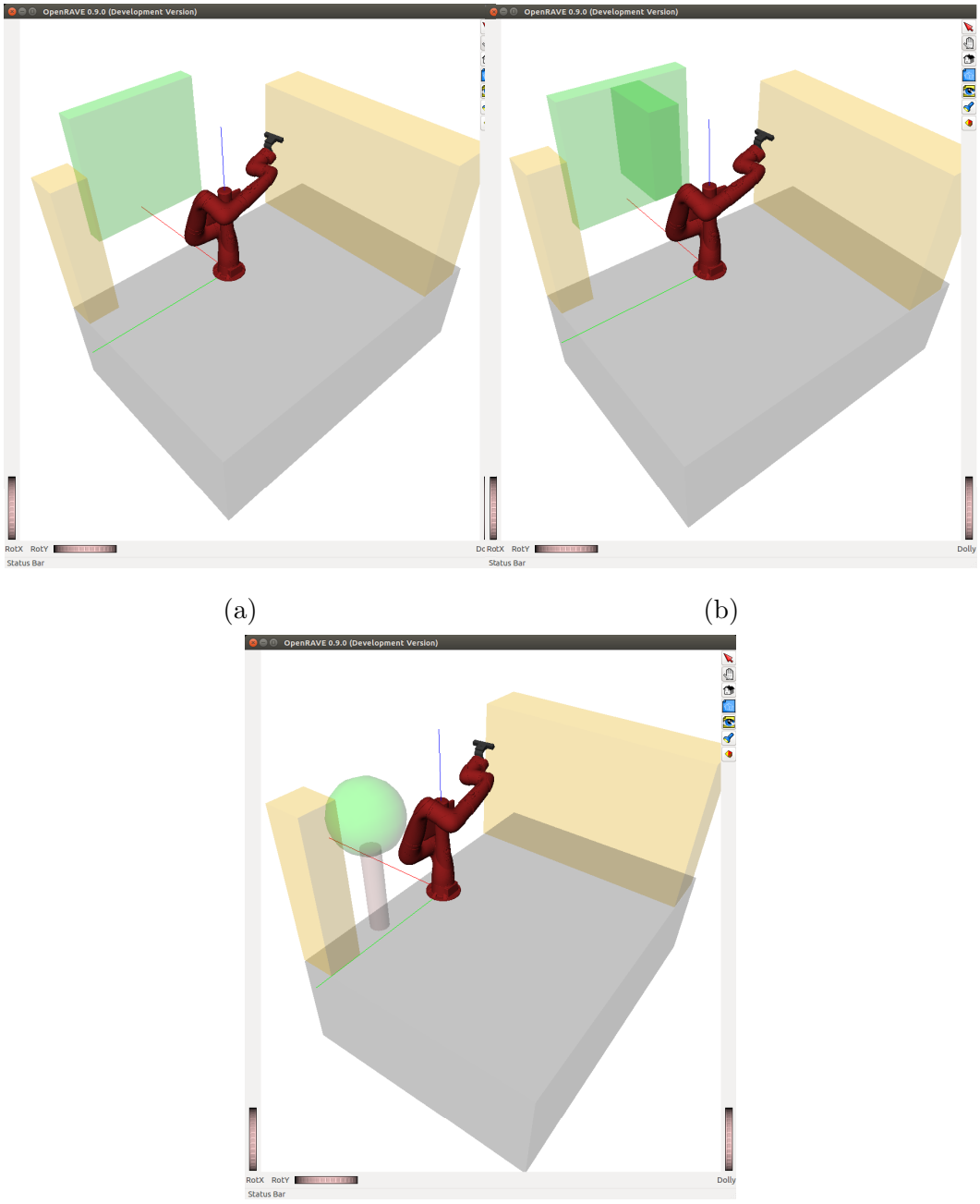
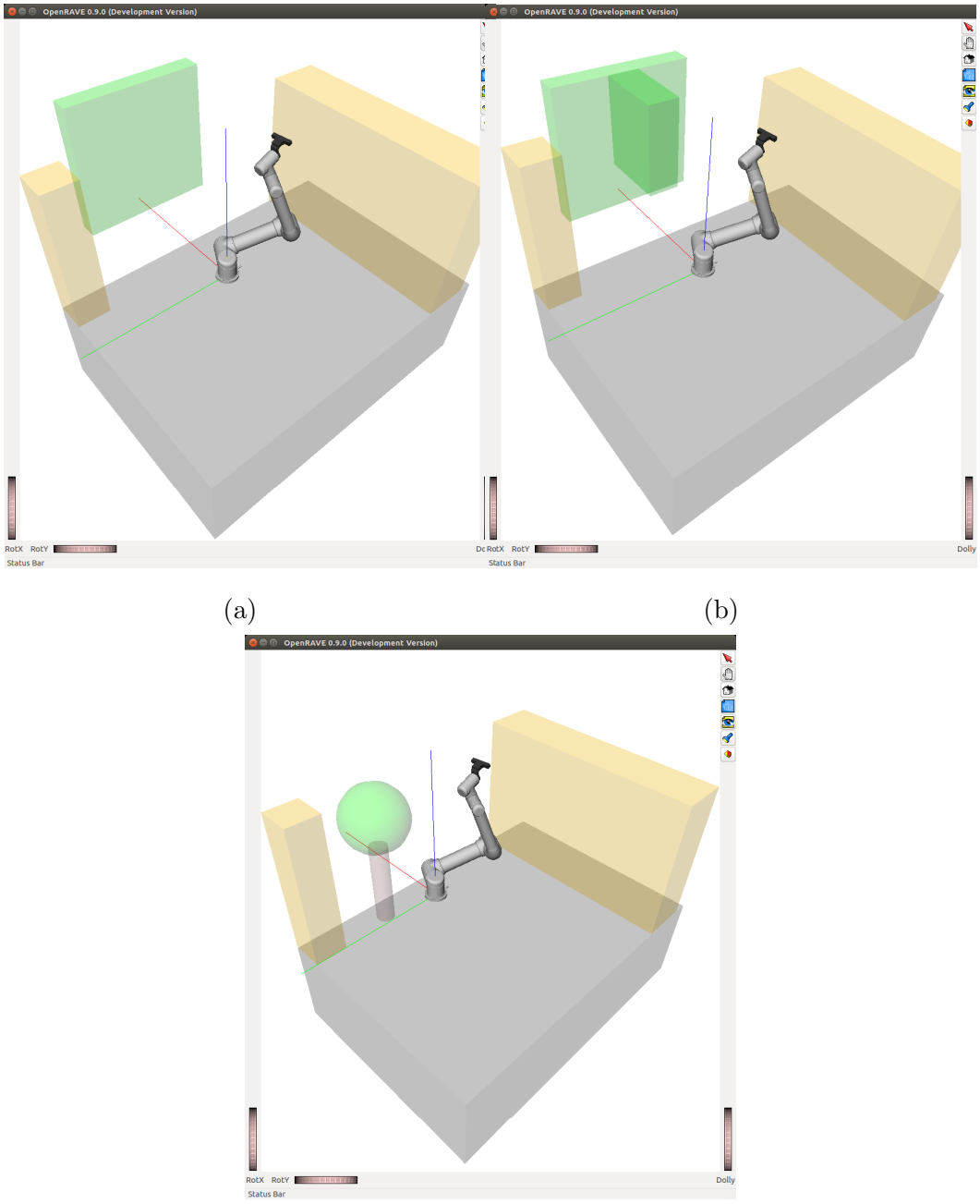


Figure 5.2 : Environment setup for Sawyer simulation tests: (a) “Sawyer Flat”, (b) “Sawyer Branch”, (c) “Sawyer Tree”

in C_{free} and respecting the manipulator’s kinematic constraints. For the second motion, the current pose of the arm is offset along the current direction of the end effector. For this motion the end effector is constrained to move along this direction vector, so that a “stroking” motion is achieved. This is visualised in Figures 5.4 - 5.9 (b).

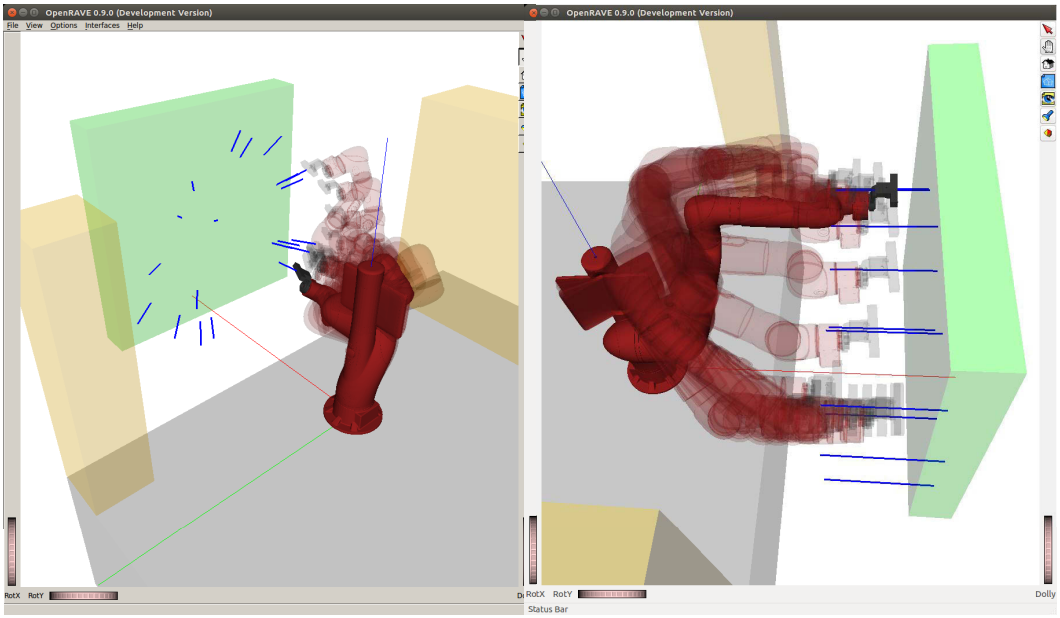


(a) (b)

(c)

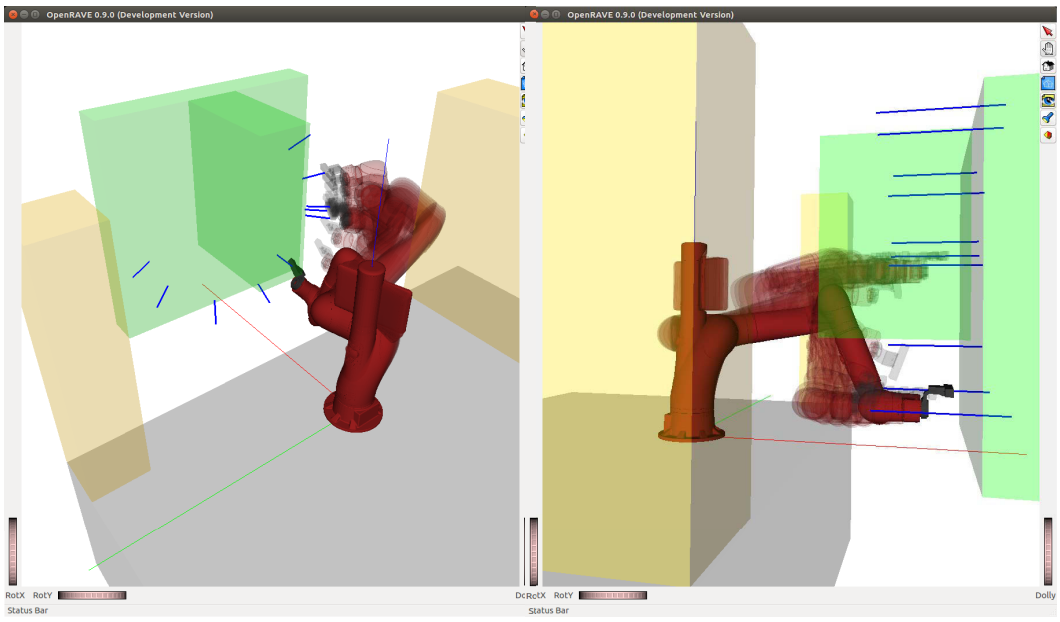
Figure 5.3 : Environment setup for UR5 simulation tests: (a) "UR5 Flat", (b) "UR5 Branch", (c) "UR5 Tree"

Further, when performing the constrained tasks for the "Branch" scene, only the trellis is disabled, hence the branch obstacle region is still avoided by the arm in the motion planner. Similarly, for the "Tree" scene only the trunk of the tree is disabled.



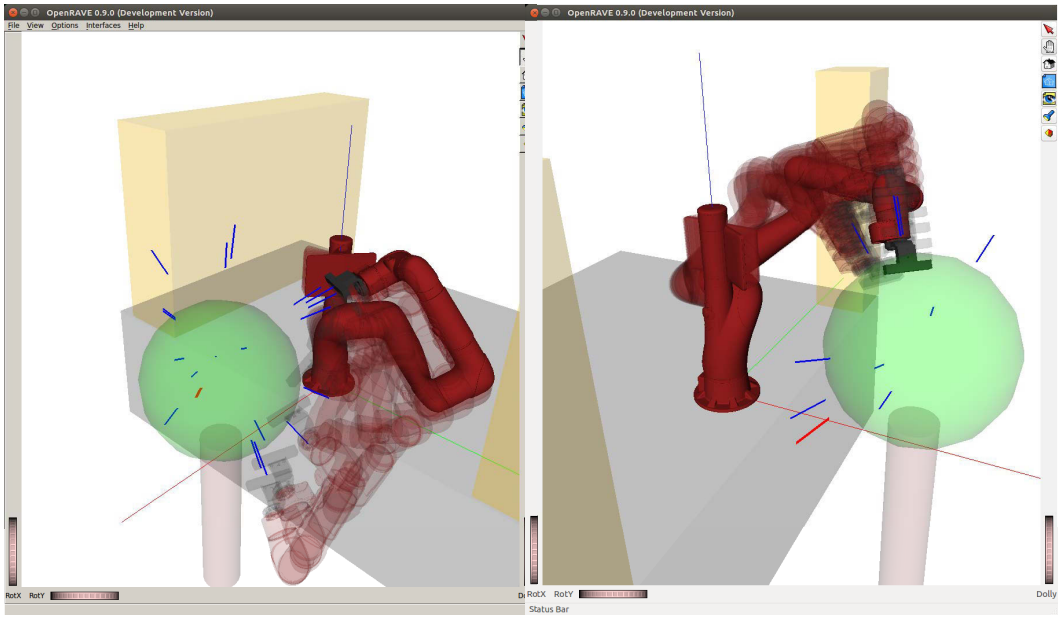
(a) (b)

Figure 5.4 : Visualisation of randomly generated scenarios for Sawyer "Flat" simulations, blue vectors indicate the intended task pose: (a) are unconstrained tasks, (b) are constrained tasks.



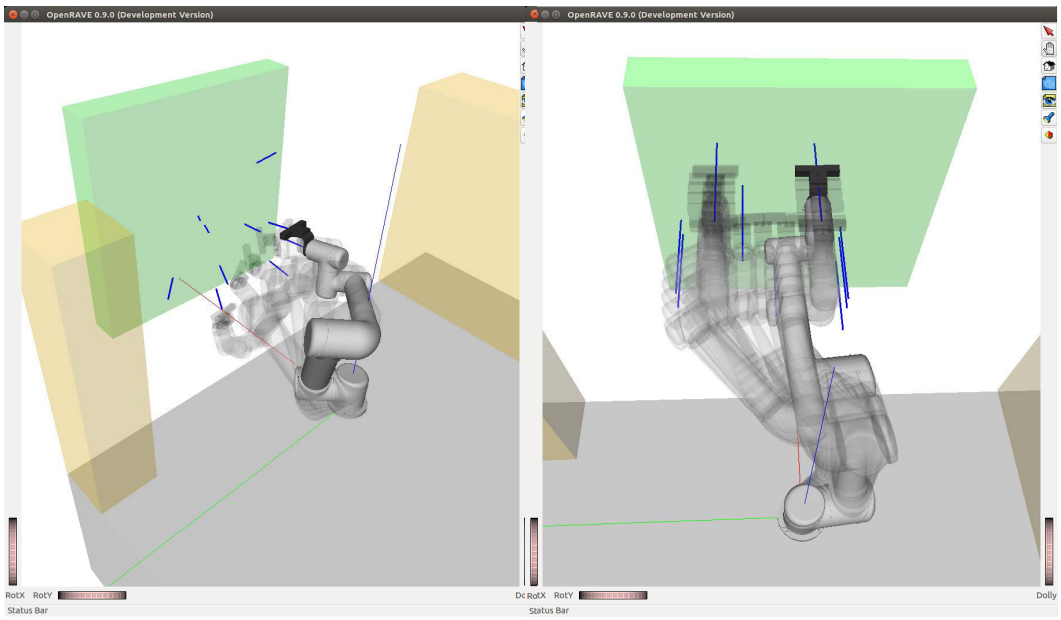
(a) (b)

Figure 5.5 : Visualisation of randomly generated scenarios for Sawyer "Branch" simulations, blue vectors indicate the intended task pose: (a) are unconstrained tasks, (b) are constrained tasks.



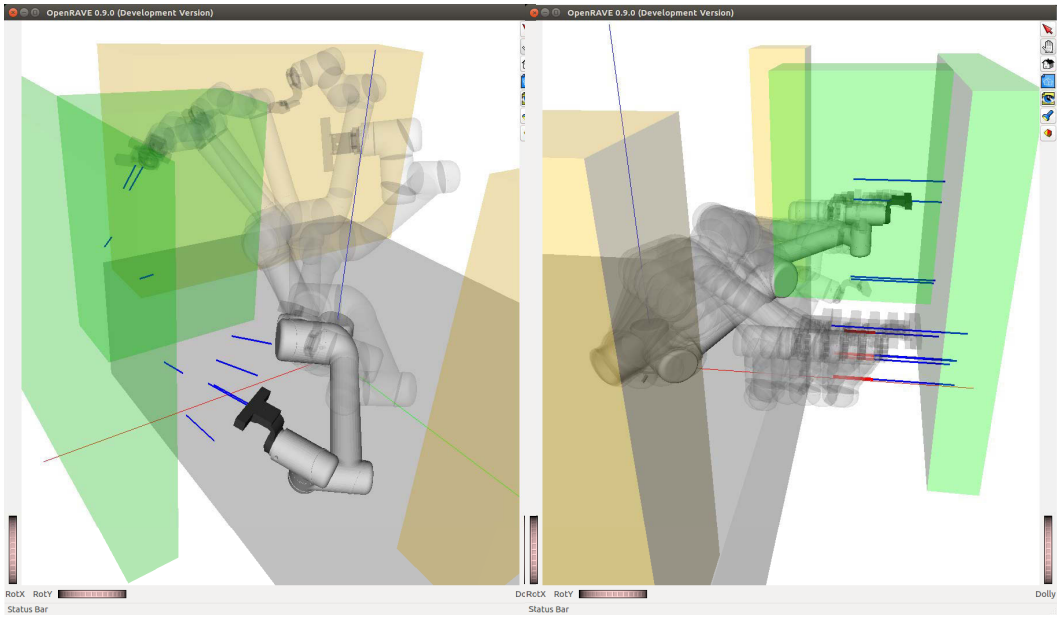
(a) (b)

Figure 5.6 : Visualisation of randomly generated scenarios for Sawyer "Tree" simulations, blue vectors indicate the intended task pose and red vectors indicate the task is unreachable: (a) are unconstrained tasks, (b) are constrained tasks.



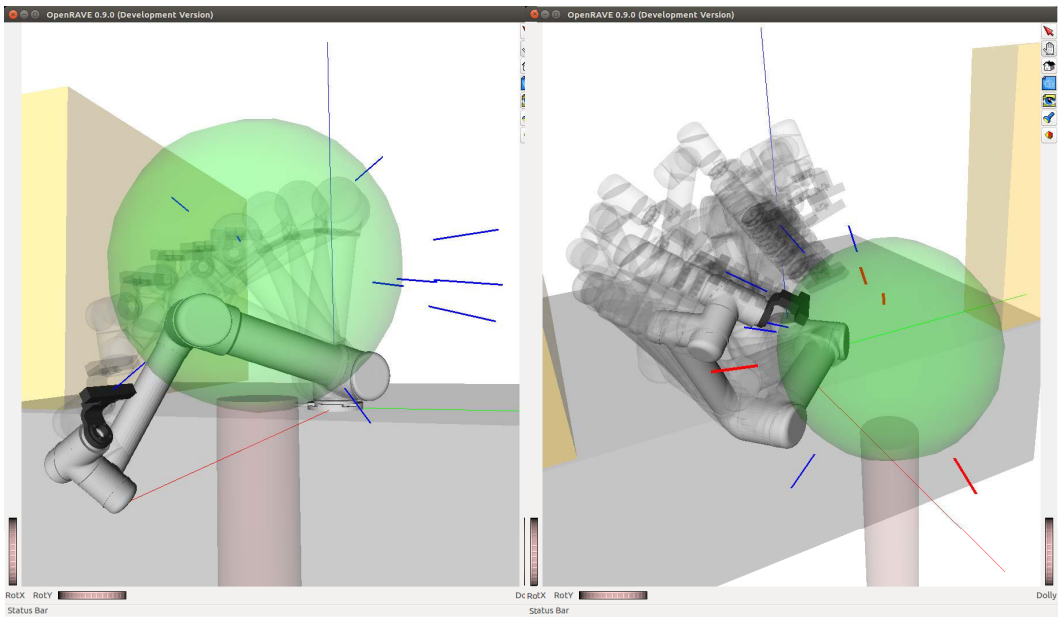
(a) (b)

Figure 5.7 : Visualisation of randomly generated scenarios for UR5 "Flat" simulations, blue vectors indicate the intended task pose: (a) are unconstrained tasks, (b) are constrained tasks.



(a) (b)

Figure 5.8 : Visualisation of randomly generated scenarios for UR5 “Branch” simulations, blue vectors indicate the intended task pose: (a) are unconstrained tasks, (b) are constrained tasks.



(a) (b)

Figure 5.9 : Visualisation of randomly generated scenarios for UR5 “Tree” simulations, blue vectors indicate the intended task pose: (a) are unconstrained tasks, (b) are constrained tasks.

To assess the performance of FREDS-MP, several metrics are measured per test scenario:

- **Online Time (s):** The total time for the online motion planning phase.
- **Task Time (s):** The total time for the task planning phase.
- **Average Execution Time Per Task Pairs (s):** The total execution time divided by the number of successfully planned task pair. The average is used to account for plan failures, which do not contribute to the total execution time.
- **Plan Success (%):** The percentage of tasks that were successfully executed.

For each scene, results are divided into test groups, where each group is categorised by the number of tasks generated per scenario. For each test group, the planners are run through 20 randomly generated scenarios. The above metrics are tabulated for each test group and the mean and SEM for each metric's sample population is plotted.

To provide a fair comparison, both planners use the same starting manipulator configuration and target poses for each test iteration. In addition, for each test scenario the arm starts and finishes at a known home position. That is, the task planner will consider the best sequence of target poses, taking into consideration the fact that the arm will travel from a home position, visualised in Figure 5.2, to the first target and then back again from the last target. The intent is to simulate a real world scenario, where the arm will be mobile and as it travels along a row of crops, it will need to tuck itself into a safe position at the completion of a scenario before moving onto the next one. The above tests are carried out on a Sawyer, a redundant seven-DOF, and a UR5, non-redundant sxi-DOF, robot arm.

The test procedure can be summarised as follows:

---

for *each robot arm* **do**
  for *each scene* **do**
    for *each test group* **do**
      for 20 *iterations* **do**

generate scenario
run $FREDS - MP$
run $BIT*$
run $RRTConnect$
Computed and store metrics
end for
end for
end for
end for

The simulation experiments were run on an Intel NUC PC with Intel i7 quad core CPU, 32GB RAM and 256GB SSD.

5.2.4 Simulation Results

In this section the results of the Sawyer and UR5 simulations are presented as plots, grouped by each of the measured metrics so that the results for the different robot arms, tasks and scenes may be compared side by side.

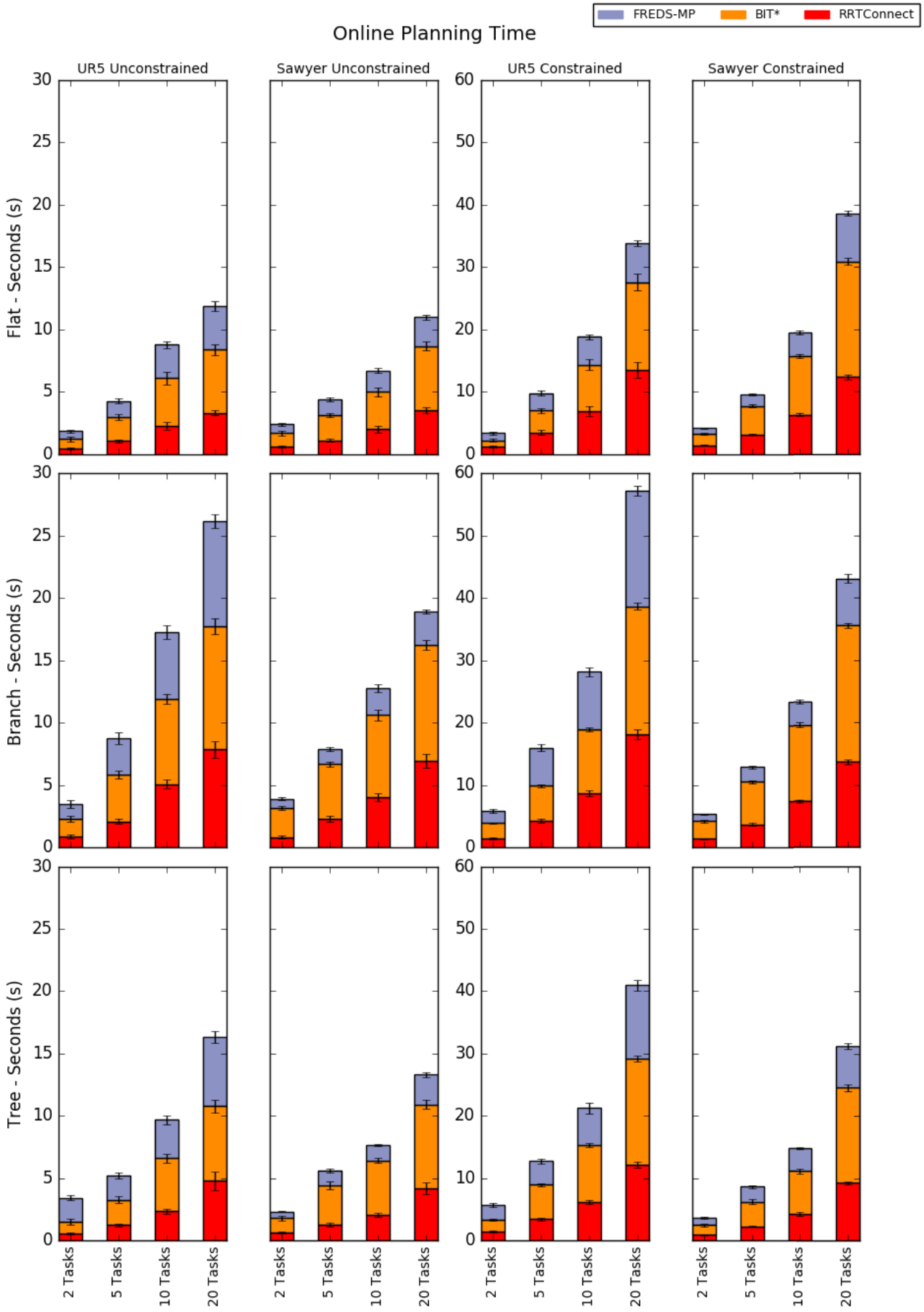


Figure 5.10 : Online planning time plots, grouped by robot arm configuration/ task type horizontally and experiment scene vertically.

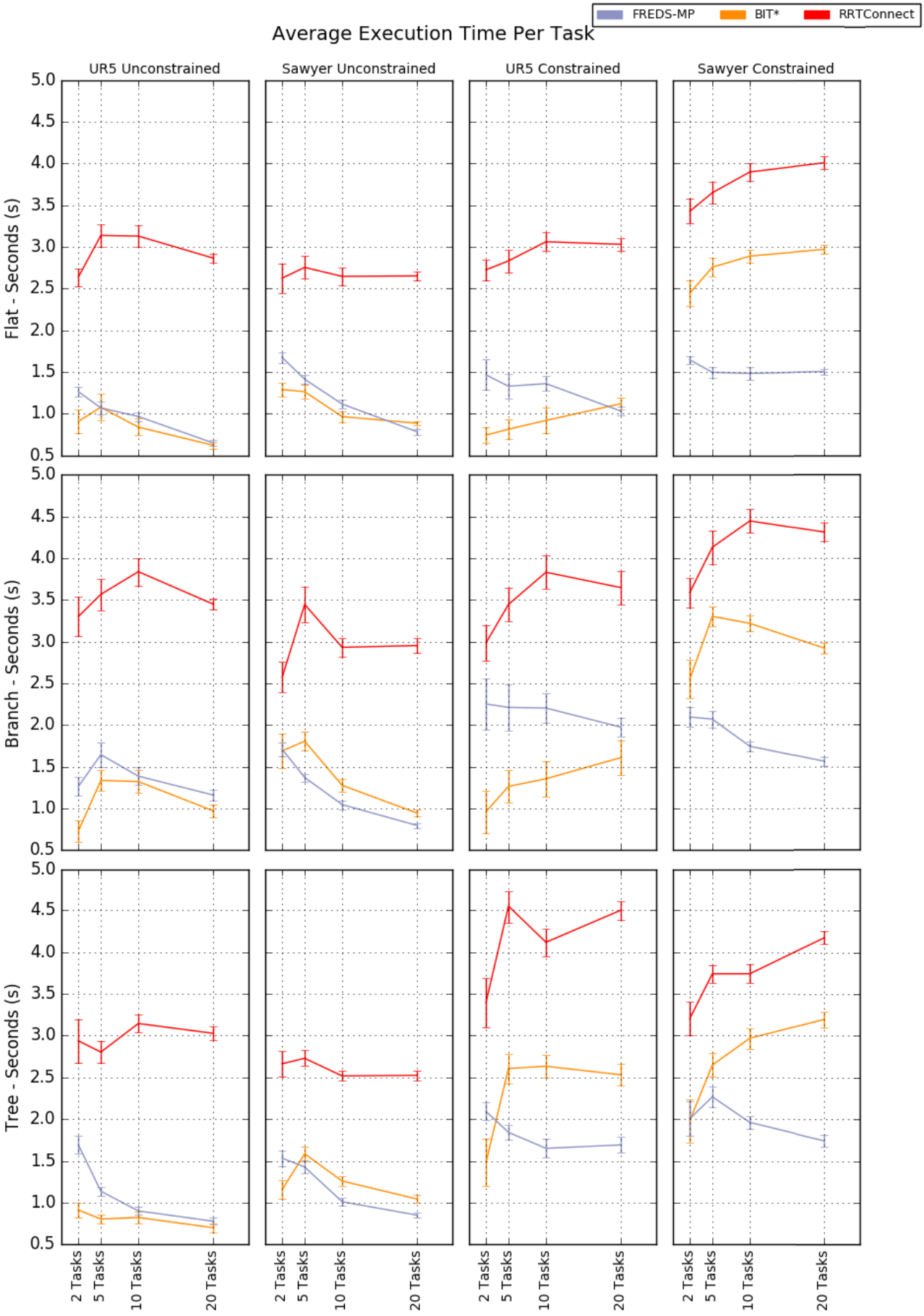


Figure 5.11 : Average task execution time plots, grouped by robot arm configuration/ task type horizontally and experiment scene vertically.

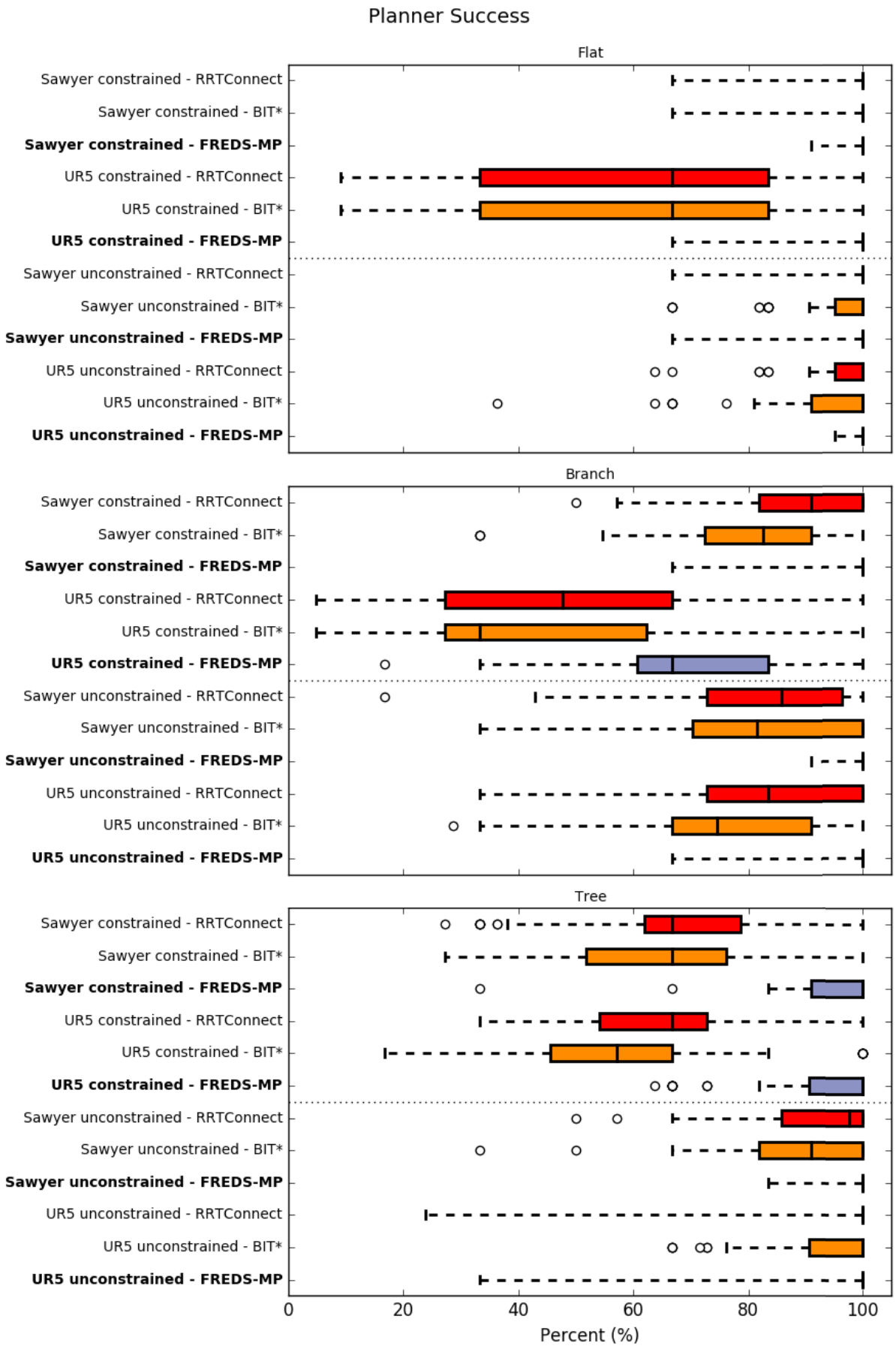
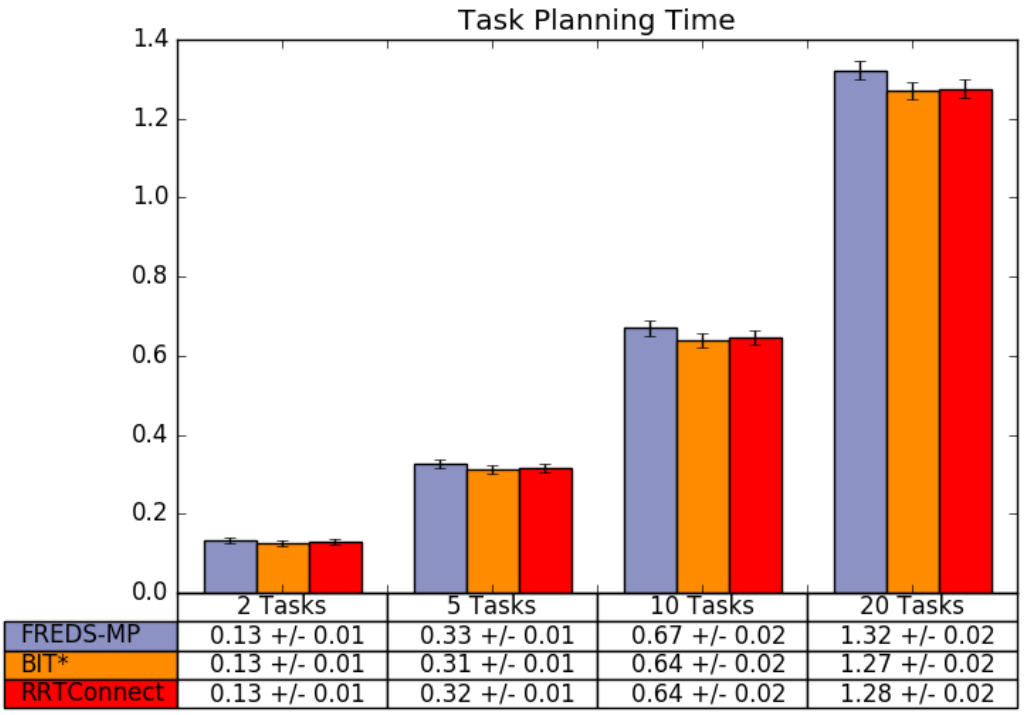


Figure 5.12 : Planner success plots, each plot is split by constrained and unconstrained tasks and then grouped vertically by each experiment scene.



	2 Tasks	5 Tasks	10 Tasks	20 Tasks
FREDS-MP	0.13 +/- 0.01	0.33 +/- 0.01	0.67 +/- 0.02	1.32 +/- 0.02
BIT*	0.13 +/- 0.01	0.31 +/- 0.01	0.64 +/- 0.02	1.27 +/- 0.02
RRTConnect	0.13 +/- 0.01	0.32 +/- 0.01	0.64 +/- 0.02	1.28 +/- 0.02

Figure 5.13 : Task planning time plot, showing the task planning times for each planner type and task scenario averaged over all robot arm configurations, task types and experiment scenes.

5.2.5 Discussion

In this section the results of FREDS-MP for the Sawyer and UR5 arms will be compared individually to BIT* and RRTConnect. Then the two arms will be compared with respect to their performance and suitability for the given tasks.

Sawyer Experiments

As can be seen from the results FREDS-MP almost always out performs BIT* and RRTConnect in every measured metric, in some cases by a significant margin. For example, in Figure 5.10, the online planning time for FREDS-MP in the 20 task group was on average about 3.5 times faster than BIT*. For unconstrained tasks, FREDS-MP does not show much of an improvement in average execution time over BIT*, however there was a large improvement in online planning time.

For scenarios with a low number of unconstrained tasks, RRTConnect and FREDS-MP exhibit roughly the same planning times, however for higher task groups, FREDS-MP significantly outperformed all planners. While RRTConnect exhibited fast plan-

ning times, it had significantly higher average execution times. For constrained tasks, FREDS-MP performs significantly better than RRTConnect and BIT* in both online planning and execution time.

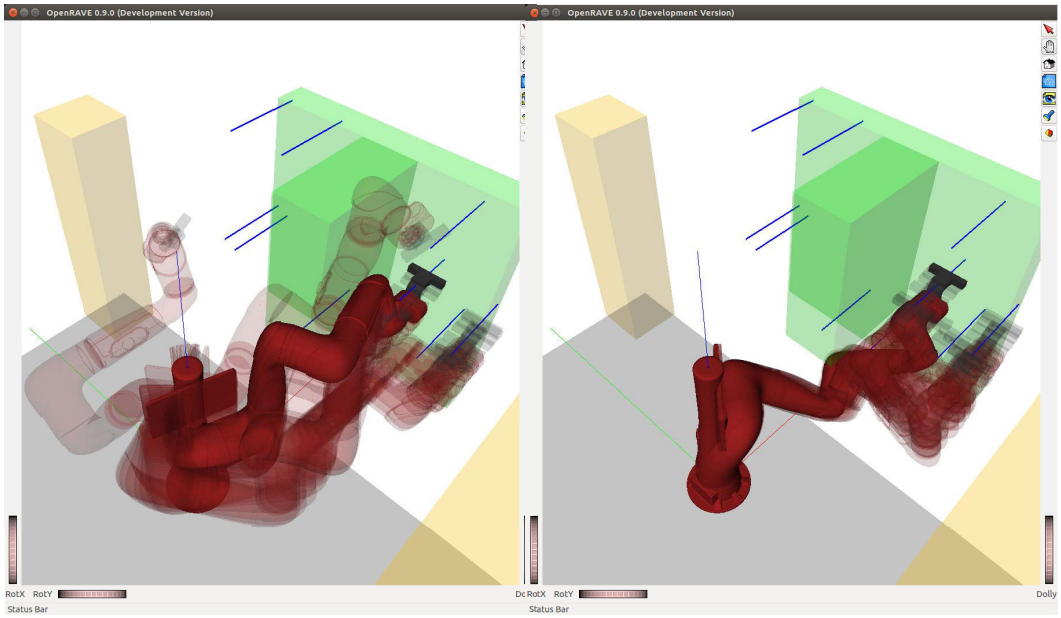
It is clear that FREDS-MP exhibited consistently high success rates and hence is highly reliable. In particular, for the "Branch" and "Tree" scenes, the high success rate can be attributed to the fact that the offline trajectory priors aided in guiding the manipulator through narrow passages when adapting them online. The consistently lower planning times also suggests that providing a non-naive seed for the trajectory optimisers results in quicker convergence.

Another interesting result observed is that as the number of targets increase, the average execution time decreases consistently for FREDS-MP. This is due to the targets becoming more densely spaced as the number of samples increases. This is expected behaviour but only exhibited in FREDS-MP's results which confirms its higher accuracy from a task planning perspective. Further, FREDS-MP's results have smaller SEM which suggests that it is more consistent in its performance and hence more reliable.

It should also be noted that for the average execution time results, the time difference for larger target groups can be significant in the real world. For example, in Figure 5.11 for the 20 target group, FREDS-MP's average execution time is 2.75 seconds faster than RRTConnect. For 20 tasks that is a savings of 55 seconds per scenario. One can imagine that in a real world scenario where these tasks are repeated over many days or even weeks, this could mean significant increase in throughput.

For the unconstrained tasks, average executions times were similar for both flat and branch trellis configurations, however, the success rates were significantly lower for BIT* and RRTConnect in the latter case, which may have contributed to their lower average execution times. In contrast, online planning times for the branch configuration were approximately double when compared to the flat configuration's for BIT* and RRTConnect, where as FREDS-MP achieved approximately the same online planning times in both configurations even though the branch configuration was significantly more complex.

Interestingly, there is not a large difference in results between the "Flat" and



(a) (b)

Figure 5.14 : Visualisation comparing sequencing between (a) BIT* and (b) FREDS-MP

"Branch" constrained task experiments which suggests that the time required for planning the constrained motions dominates the planning of unconstrained motions. Nonetheless, FREDS-MP strongly outperforms BIT* and RRTConnect in both cases, which suggests that for even the simple flat trellis configurations, utilising offline trajectory priors has a significant impact on performance for constrained tasks.

One contributing factor to FREDS-MP's lower execution times is that it can utilise more accurate heuristics when sequencing tasks. For BIT* and RRTConnect, tasks were sequenced based on Euclidean workspace, $SE(3)$, which can greatly underestimate the true cost of the trajectory. Further, BIT* and RRT* choose an IK solution to connect each consecutive task in the computed sequence based on their Euclidean difference in C which also may be a significantly underestimated heuristic. These factors can be attributed to the large difference in the resulting execution times, particular when the manipulator is trying to manoeuvre through narrow passages or needs to execute tasks at the extremes of the manipulator's workspace, particularly for the branch trellis configuration.

Another contributing factor for FREDS-MP's lower execution times is due to the optimised IK solutions pre-computed for each task. This is particularly evident in the constrained tasks where there is much less flexibility in what manipulator configurations are available to choose from when connecting tasks. This, coupled with the random nature of CBiRRT can result in sub-optimal starting configurations for the constrained motions. FREDS-MP considers this at the offline stage and hence can optimise what these start configurations should be and hence provide a better heuristic for sequencing constrained tasks. This is evident in Figure 5.14, where FREDS-MP connects two constrained tasks with much less manipulator movement than BIT*.

The above observations were consistent with the tree configuration, however, for the constrained tasks the path success rate was much lower for BIT* and RRTConnect compared to other experiments, where as FREDS-MP still maintained a very high success rate. This further proves the reliability of FREDS-MP in even the most complex scenarios, particularly in those where narrow passages need to be planned through.

Finally, as can be seen in the task planning times for all scenarios, the overhead due to the database search and task modification in FREDS-MP is negligible.

UR5 Experiments

In the unconstrained UR5 experiments, FREDS-MP roughly performed on par with BIT*. FREDS-MP generally achieved better online planning times with more noticable improvements for the higher task number groups, however, BIT* achieved slightly better average execution times. While RRTConnect achieved faster online planning times, its average execution time was significantly worse, in some cases 4.5 times slower than FREDS-MP. One metric where FREDS-MP excelled in was the planning success rate, it consistently achieved high success rates, more specifically it never dropped below 96%, where as BIT* and RRTConnect achieved success rates lower than 80% for some task groups.

In the constrained experiments, the results follow the same trend, except there was an even more significant separation in the planning success rate of FREDS-MP compared to BIT* and RRTConnect. While, BIT* and RRTConnect dropped as

low as 30%, FREDS-MP never dropped below 60%. The side effect of having a much higher success rate meant that FREDS-MP average execution times suffered, most likely due to the successful trajectories being complex and requiring large movements.

It should be noted at this point that one might expect FREDS-MP to have at least as fast average execution time as BIT* since it uses it as a backup planner in the case that all the trajectory optimisers fail. There are a few reasons why this assumption does not hold: first, due to the difference in sequencing strategy FREDS-MP may choose different start and end configurations for each task; secondly, recall BIT* is probabilistic in nature, hence there is no guarantee it will produce the same result every time; thirdly, trajectory optimisers can still produce highly sub-optimal paths depending on the initial seed and obstacle configuration.

Finally, as can be seen in the task planning times for all scenarios, the overhead due to the database search and task modification in FREDS-MP is negligible.

Comparing Manipulators

The difference in FREDS-MP's performance between the UR5 and the Sawyer experiments are surprisingly significant. Aside from the fact that FREDS-MP performed much better in the Sawyer experiments, the overall results of all the planners was worse in the UR5 experiments. This suggests that the Sawyer is better at performing the tasks for the intended application.

The Sawyer's better performance is attributed to the greater dexterity resulting from having redundant seven-DOF. In particular, the Sawyer arm can manoeuvre itself through tight passages more easily than the UR5. Figure 5.15 shows an example of a difficult pose in a tight passage that the Sawyer was able to achieve but the UR5 could not due to a singularity. Moreover, FREDS-MP performed significantly better in the Sawyer experiments than the UR5's because it was able to utilise the redundancy of the arm effectively for task sequencing.

Interestingly, there was not a significant difference in the results between the UR5 and Sawyer for BIT* and RRTConnect for unconstrained tasks which suggests that the time-complexity of the planners were not impacted by the increasing number of DOF. However, there was a significant difference in the success rate for

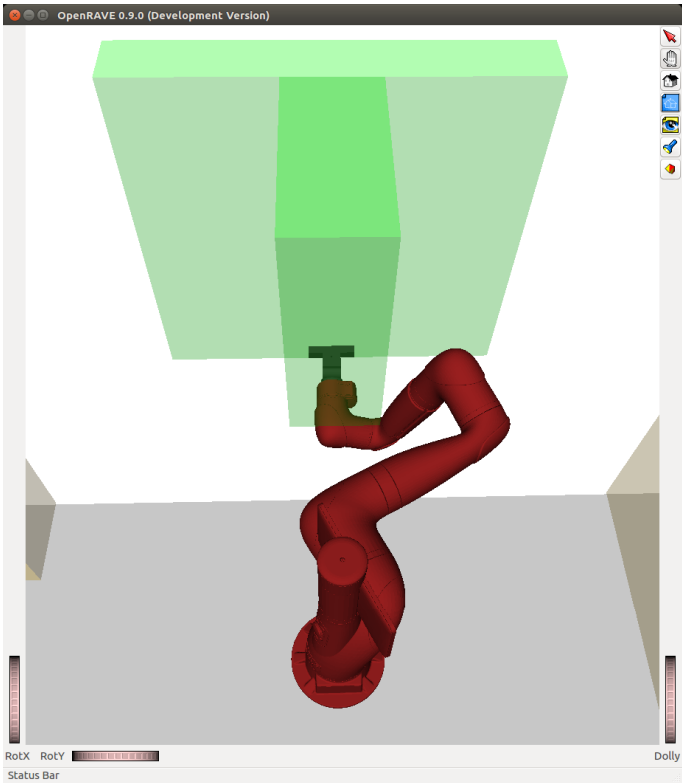


Figure 5.15 : Example where the Sawyer arm is dexterous enough to achieve a difficult pose where as the UR5 cannot due to a singularity.

the constrained tasks, with the UR5 exhibiting much worse success rates, even for FREDS-MP. This suggests that the Sawyer is much better at performing constrained tasks due to its redundancy.

Interestingly, for the constrained experiments BIT* exhibited much worse average execution times and slightly higher online planning time for the Sawyer than the UR5. This may be attributed to the increasing problem size due to the Sawyer's redundancy, however, as mentioned earlier the high failure rate of the UR5 may have caused this result.

Another similar trend between the arms is that for both arms, the difference in the online planning and average execution time between scenes for constrained tasks was not as significant as those for the unconstrained tasks. This further confirms the fact that generating plans for the constrained motions dominated unconstrained motions.

Chapter 6

Hardware Experiments

In this chapter, FREDS-MP is tested on a real Sawyer robot arm. The intended application is active perception, where apples are first localised on a trellis and then inspected individually. The experimental setup, methodology and results are presented in this chapter. Lastly, the results of the hardware experiments are analysed and discussed.

6.1 Hardware Setup

From the simulated experiments, it was shown that the redundant Sawyer manipulator was more effective than the UR5 at performing its tasks, particularly constrained tasks. Therefore, the experiments in this chapter were carried out on the Sawyer. The focus application will be active perception and close up inspection of apples on a trellis. A $1.0m \times 1.0m$ section of an apple trellis, pictured in Figure 6.1 (b), was constructed using artificial vines and apples, which very accurately represents what a real apple trellis looks like.

The Sawyer arm is fixed to a stationary platform, pictured in Figure 6.1 (a), with obstructing structures to emulate a typical chasis of a mobile robot. Since commerical mobile robots in agriculture are limited, the chasis structure used in these experiments has been inspired by the "Swarmbot" from Swarmfarm Robotics [110], the only existing commerical robot that would be viable for the intended application. The chasis in this experiment only represents part of the Swarmbot's structure that would be within the vicinity of the Sawyer arm's workspace and includes lateral obstacles such as the rear wheel post and e-box.

Mounted on the Sawyer end effector is an Intel Realsense SR300 camera, as shown in Figure 6.1 (a). The SR300 is used to capture RGBD images which are used by the perception module, described later in Section 6.2, to segment and localise the apples as well as take photos when performing close-up inspection of the apples.

(a) (b)

Figure 6.1 : Photo of Sawyer robot arm mounting (a) and artificial apple trellis (b).

The software used for the hardware experiments consists of two distinct modules: a perception module and a planner module. The perception module performs all the perception tasks, such as; map building, viewpoint planning and apple localisation. The planning module contains FREDS-MP and all the other necessary processes to execute tasks on the real Sawyer.

These two modules run on separate PC's, both consisting of Intel i7 quad core CPU's, 32GB RAM and 256GB SSD's. They communicate to each other via TCP for sending commands. The Sawyer arm also runs an SDK on its own PC and communicates with the rest of the system via the ROS framework. The planner module uses ROS topics to query the state of the Sawyer and to also send trajectories for execution.

To allow communication between the perception and planner module, both provide a TCP server front end which acts as the interface to the rest of their respective systems. They listen on a TCP socket for messages, containing serialised commands, and once received deserialise and parse them for execution.

To control the robot arm, its SDK provides a built-in Joint Trajectory Action Server (JTAS) which facilitates commanding the arm through multiple waypoints. The JTAS takes as input a list of timestamped waypoints and then determines appropriate joint torque commands, through interpolation, to send to the arm so that the given trajectory is followed.

6.2 Methodology

The experiments are split into two closely coupled phases. The first is the active perception, where multiple viewpoints of the apple trellis are captured using the Realsense camera mounted on the Sawyer. As each viewpoint is taken, the perception module stitches the RGBD images together to create a unified map and performs probabilistic segmentation on this map to localise the apples. Once the apples have been located on the trellis, the Sawyer commences the second phase, the inspection phase, where it takes a close up RGB photo of each apple.

For the active perception phase, the sequence of commands is as follows. First the perception module sends a request to the planner module for the current pose of the Realsense camera frame relative to the Sawyer base. Then the perception module computes a target pose based on the current pose and map information, which is then sent to the planner module for planning and execution. The planner module then sends an acknowledgement to the perception module indicating whether the execution succeeded or failed. This process is repeated until no more viewpoints are needed.

The inspection phase commences once no more viewpoints are needed. To inspect the apples, the planner module must generate inspection tasks. To do so, the planner requests the point clouds of each individual apple and the centre of the apple from the perception module. Using this information, the centroid of the point cloud is computed by the planner and a "look at" vector is produced by taking each apple's centroid and extending a vector towards the centre of the corresponding apple. This is visualised in Figure 6.2.

Due to the cluttered nature of the trellis, to avoid having any part of the end effector get caught on a branch the inspection task is broken into two motions. The first is moving the manipulator from its current pose to a pose that is facing the

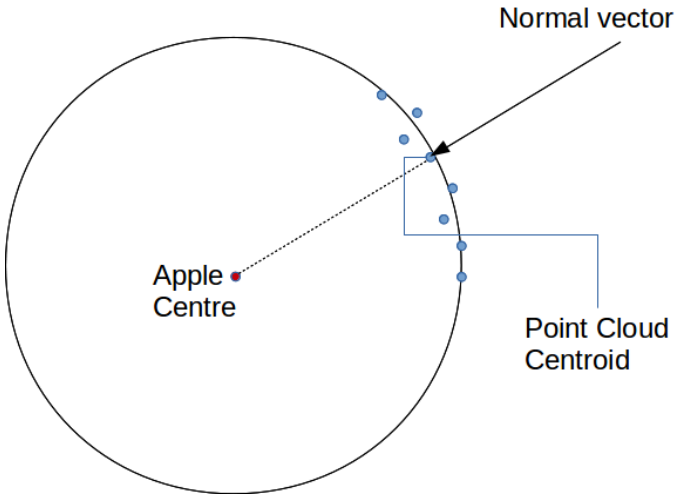


Figure 6.2 : Illustration of how the inspection pose is generated using apple centre and point cloud information from the perception module.

apple and not within a close proximity to the trellis. For this first motion there is no constraint on how the manipulator should get to this pose, other than being in C_{free} and respecting the manipulator's kinematic constraints. For the second motion, the current pose of the arm is offset along the current direction of the end effector towards the apple. For this motion the end effector is constrained to move along this direction vector, so that a "stroking" motion is achieved. These two motions make up the inspection task which is visualised in Figure 6.3. It should be noted that this inspection task could easily be adapted for grasping and picking the apples from the trellis.

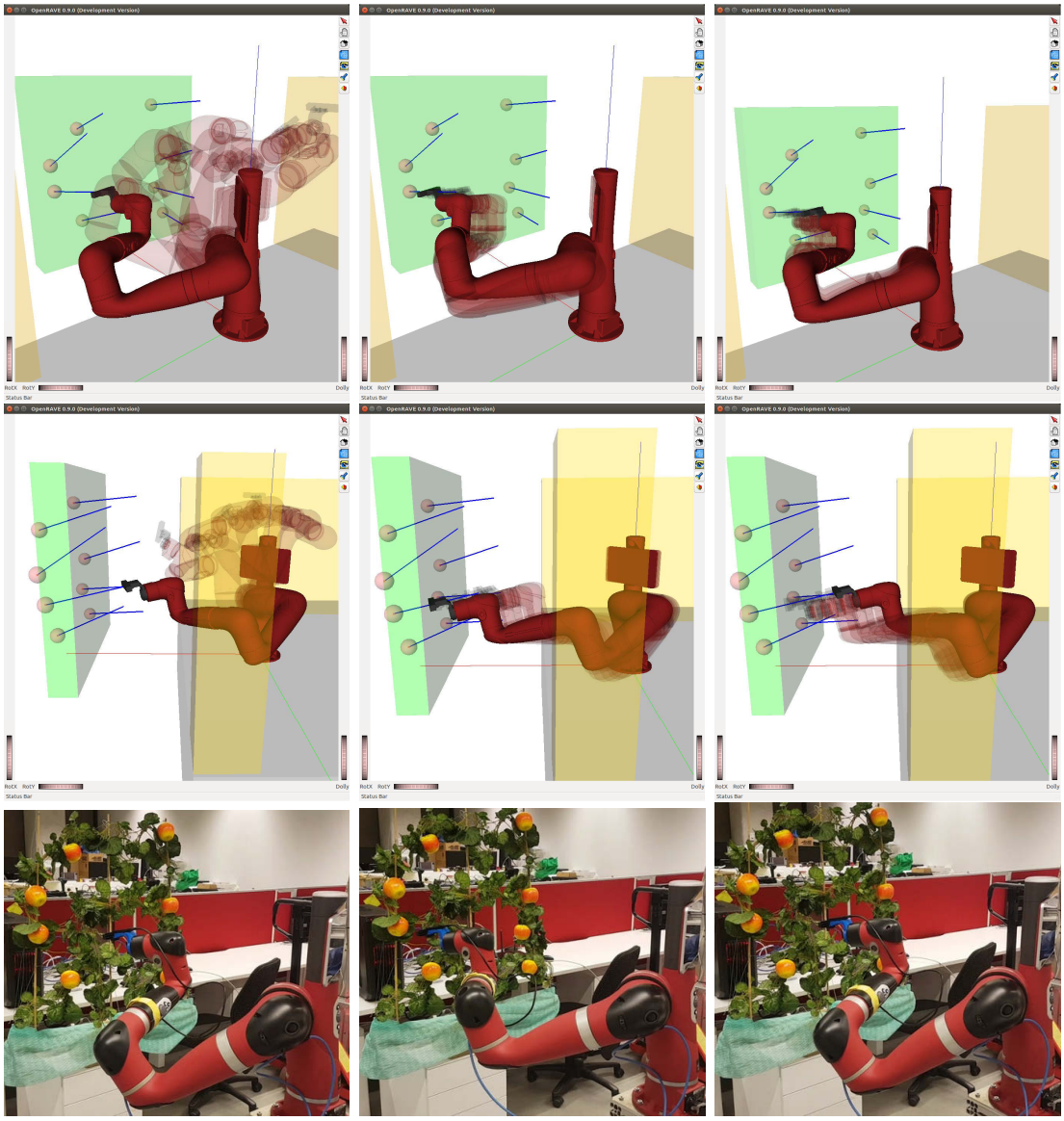


Figure 6.3 : Visualisation of constrained motion for inspecting fruit; columns correspond to the same motion from different perspectives.

The perception module tasks, including: probabilistic segmentation, map building and viewpoint pose generation are assumed to be performed by another process. However, the inspection pose generation is performed by the motion planning module and is in the scope of this thesis.

The experiments are run with two different trellis setups. The first is a flat trellis and the second is a trellis with a box shaped obstacle to represent a protruding branch - a typical scenario in a real apple orchard. To assess the robustness of the planner, each experiment is varied five times by placing the apples in different configurations and then performing the active perception and apple inspection five

times on each configuration. The different trellis types and their apple configurations are labelled in Figures 6.4 and 6.5. The reason that these tasks are performed five times on the same trellis configuration is because of the probalistic nature of the active perception algorithm, that is the generated inspection poses will be slightly different every time.

As with the simulation experiments, to measure the performance of FREDS-MP, it is compared with BIT* and RRTConnect. Further, the same metrics used in the simulated experiments will be used here. For the active perception phase, the viewpoint poses are generated on single step horizons and therefore no sequencing is required. For this reason, the performance of these tasks is not reported in the results section. In contrast, for the inspection phase all the apple locations are known at plan time and sequencing is required. Hence, performance of the inspection tasks are reported.

(a)

(b)

(c)

(d)

(e)

Figure 6.4 : Trellis configurations used for hardware experiments: (a), (b), (c), (d), (e) “Flat 1-5”.

(a) (b)

(c) (d)

(e)

Figure 6.5 : Trellis configurations used for hardware experiments: (a), (b), (c), (d), (e) “Branch 1-5”.

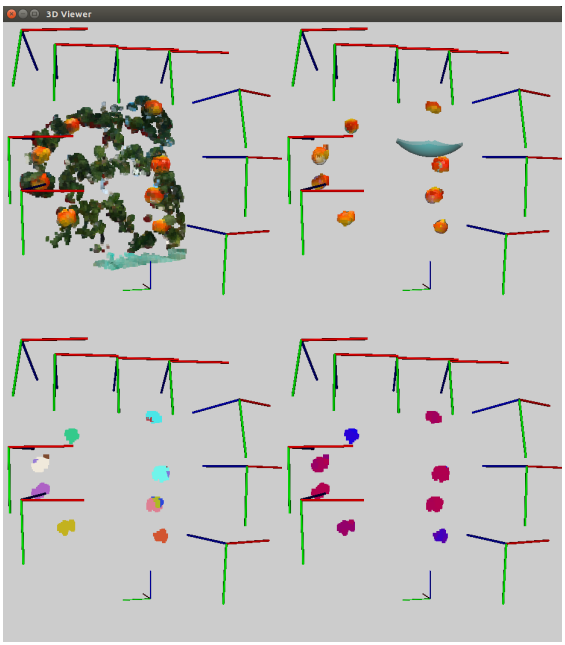
6.3 Results

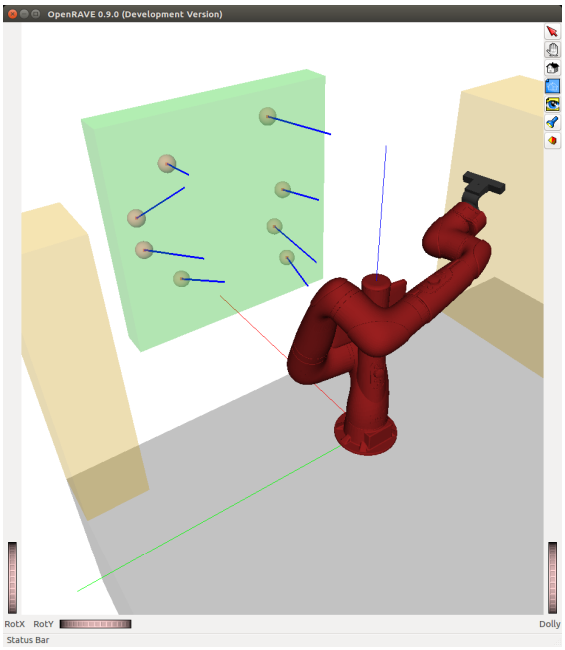
The first trellis configurations tested were “Flat 1-5”. The experimental setup for “Flat 1” is visualised in Figure 6.6 (a). For all other experiments, the setup remained the same except for the trellis’s apple locations.The close-up inspection photos of each individual apples is shown in Figure 6.7. Similarly, Figures 6.8 (a) and 6.9 show the setup and close-up inspection photos for “Branch 1-5”.

The output of the perception and planner modules is visualised in Figures 6.6/ 6.8 (b) and (c) respectively. The perception module output shows the stitched point cloud map in the top left of the GUI, the segmented apples in the top right, the labelled point clouds by colour in the bottom left and the association entropy map in the bottom right which shows the certainty of the segmented object being an apple. The planner module output shows the segmented apples within the context of the real scene with corresponding inspection poses visualised as blue vectors. To view perception and planner outputs for the remaining hardware experiments see Appendix A.1 and A.2.

Once the apples are located, the planner module plans for the resulting inspection tasks. The performance of these plans are plotted in Figures 6.10 - 6.12, where each measured metric is compared side by side for the Flat and Branch setups.

(a)



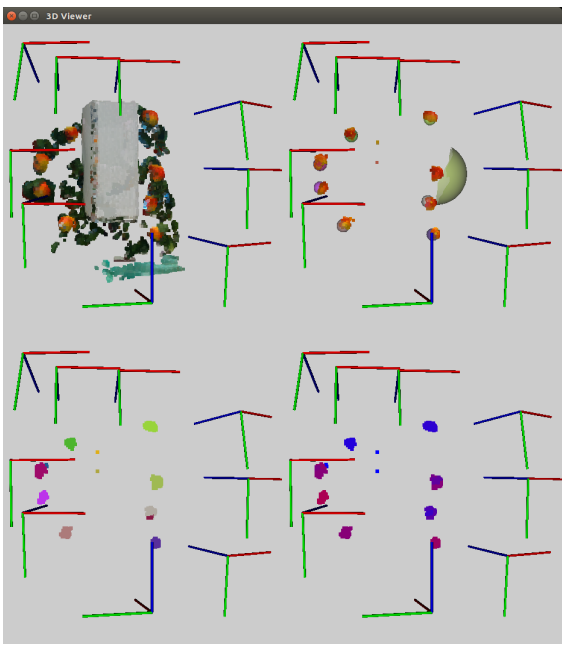


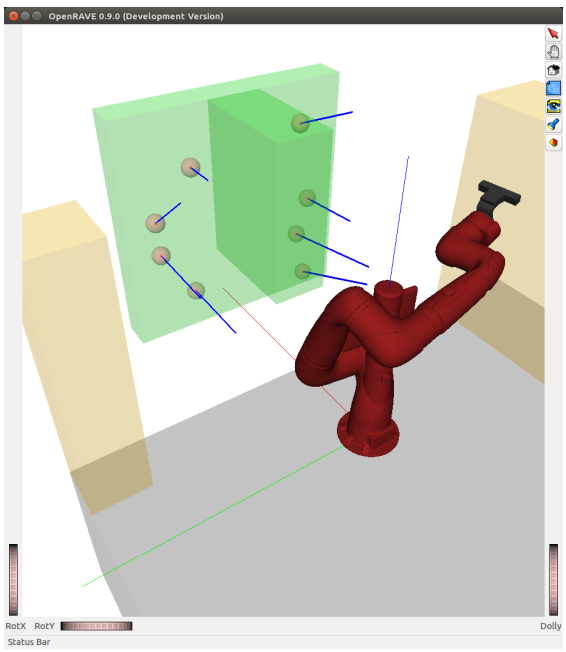
(b) (c)

Figure 6.6 : "Flat 1" experiment with planner and perception module visualisation: (a) hardware setup, (b) output from perception module, (c) output from planner module.

Figure 6.7 : Close-up images of each apple, captured during fruit inspection for "Flat 1" experiment.

(a)

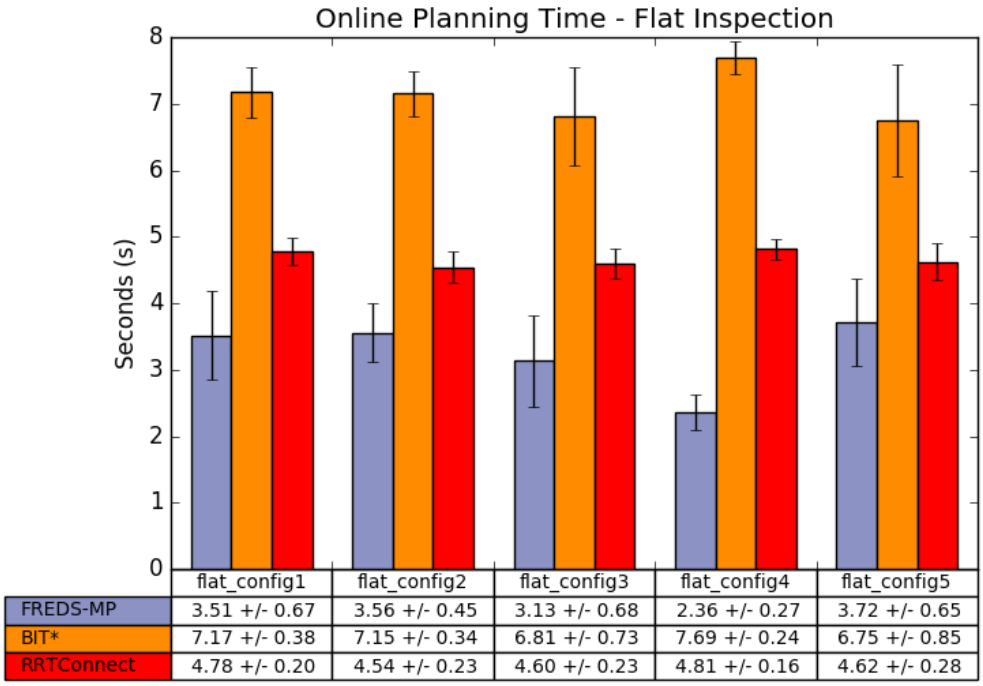




(b) (c)

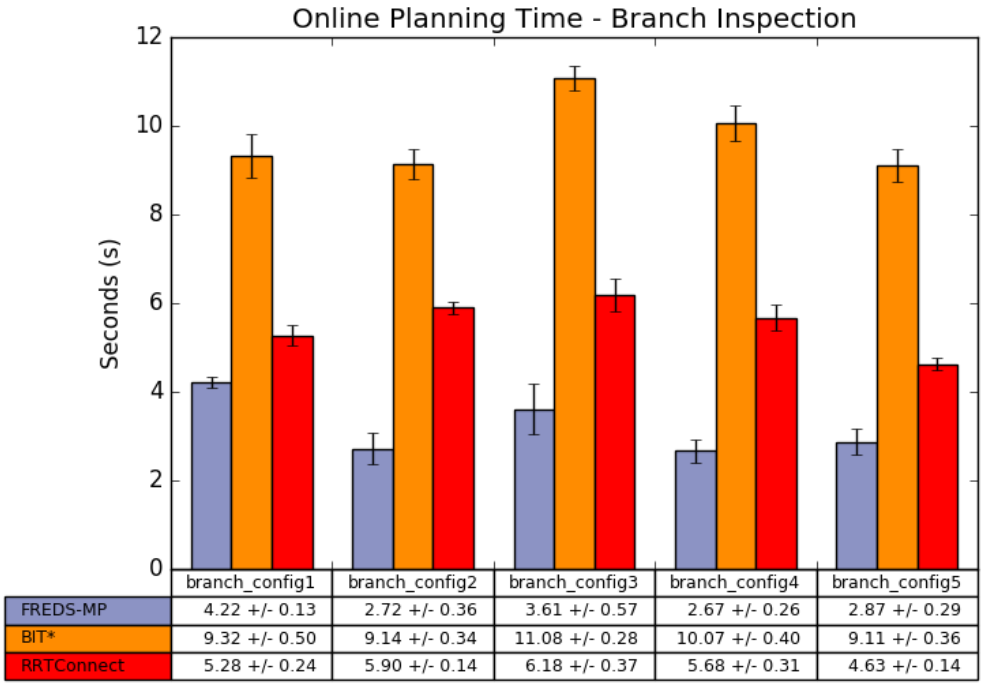
Figure 6.8 : Visualisation of “Branch 1” experiment with branch trellis type: (a) hardware setup, (b) output from perception module, (c) output from planner module.

Figure 6.9 : Close up images of each apple, captured during fruit inspection for "Branch 1" experiment.



	flat_config1	flat_config2	flat_config3	flat_config4	flat_config5
FREDS-MP	3.51 +/- 0.67	3.56 +/- 0.45	3.13 +/- 0.68	2.36 +/- 0.27	3.72 +/- 0.65
BIT*	7.17 +/- 0.38	7.15 +/- 0.34	6.81 +/- 0.73	7.69 +/- 0.24	6.75 +/- 0.85
RRTConnect	4.78 +/- 0.20	4.54 +/- 0.23	4.60 +/- 0.23	4.81 +/- 0.16	4.62 +/- 0.28

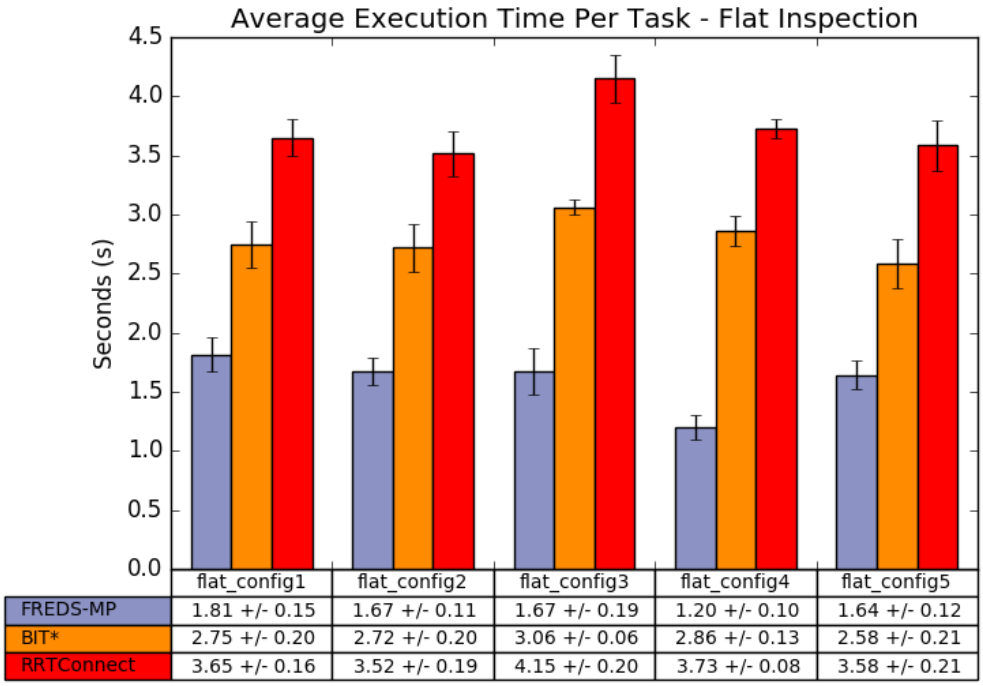
(a)



	branch_config1	branch_config2	branch_config3	branch_config4	branch_config5
FREDS-MP	4.22 +/- 0.13	2.72 +/- 0.36	3.61 +/- 0.57	2.67 +/- 0.26	2.87 +/- 0.29
BIT*	9.32 +/- 0.50	9.14 +/- 0.34	11.08 +/- 0.28	10.07 +/- 0.40	9.11 +/- 0.36
RRTConnect	5.28 +/- 0.24	5.90 +/- 0.14	6.18 +/- 0.37	5.68 +/- 0.31	4.63 +/- 0.14

(b)

Figure 6.10 : Online planning time mean and SEM plots: (b) “Flat 1-5” trellis configurations, (b) “Branch 1-5” trellis configurations.



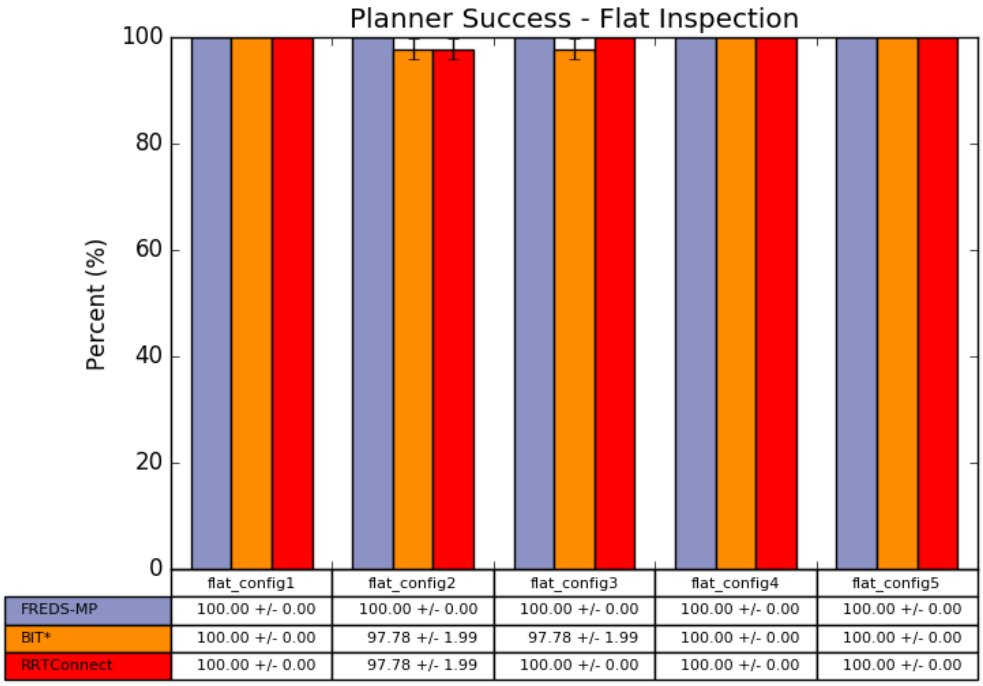
	flat_config1	flat_config2	flat_config3	flat_config4	flat_config5
FREDS-MP	1.81 +/- 0.15	1.67 +/- 0.11	1.67 +/- 0.19	1.20 +/- 0.10	1.64 +/- 0.12
BIT*	2.75 +/- 0.20	2.72 +/- 0.20	3.06 +/- 0.06	2.86 +/- 0.13	2.58 +/- 0.21
RRTConnect	3.65 +/- 0.16	3.52 +/- 0.19	4.15 +/- 0.20	3.73 +/- 0.08	3.58 +/- 0.21

(a)

	branch_config1	branch_config2	branch_config3	branch_config4	branch_config5
FREDS-MP	1.63 +/- 0.08	1.43 +/- 0.10	1.58 +/- 0.16	1.32 +/- 0.05	1.47 +/- 0.07
BIT*	2.95 +/- 0.16	2.89 +/- 0.26	3.04 +/- 0.10	3.12 +/- 0.18	2.96 +/- 0.14
RRTConnect	3.72 +/- 0.22	4.03 +/- 0.22	4.56 +/- 0.26	3.78 +/- 0.20	4.02 +/- 0.12

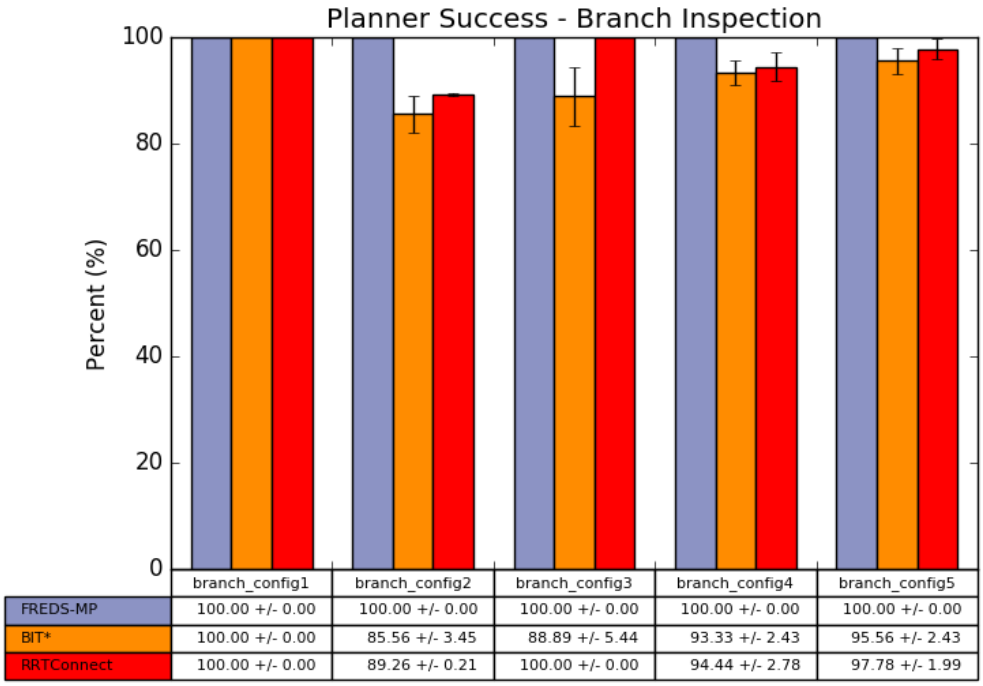
(b)

Figure 6.11 : Average task execution time mean and SEM plots: (a) “Flat 1-5” trellis configurations, (b) “Branch 1-5” trellis configurations.



	flat_config1	flat_config2	flat_config3	flat_config4	flat_config5
FREDS-MP	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00
BIT*	100.00 +/- 0.00	97.78 +/- 1.99	97.78 +/- 1.99	100.00 +/- 0.00	100.00 +/- 0.00
RRTConnect	100.00 +/- 0.00	97.78 +/- 1.99	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00

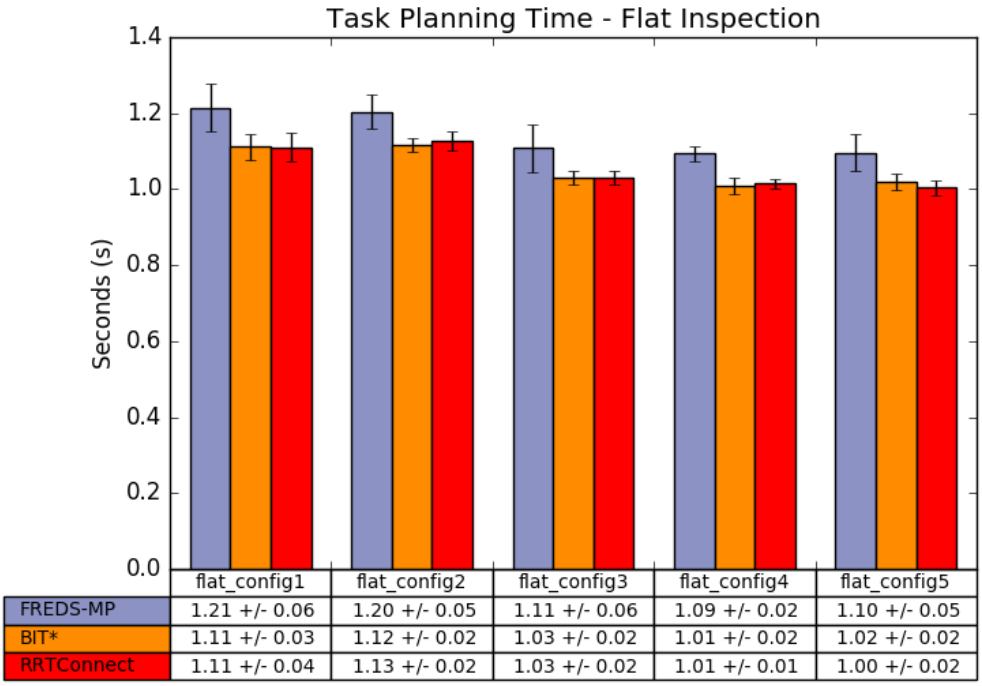
(a)



	branch_config1	branch_config2	branch_config3	branch_config4	branch_config5
FREDS-MP	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00	100.00 +/- 0.00
BIT*	100.00 +/- 0.00	85.56 +/- 3.45	88.89 +/- 5.44	93.33 +/- 2.43	95.56 +/- 2.43
RRTConnect	100.00 +/- 0.00	89.26 +/- 0.21	100.00 +/- 0.00	94.44 +/- 2.78	97.78 +/- 1.99

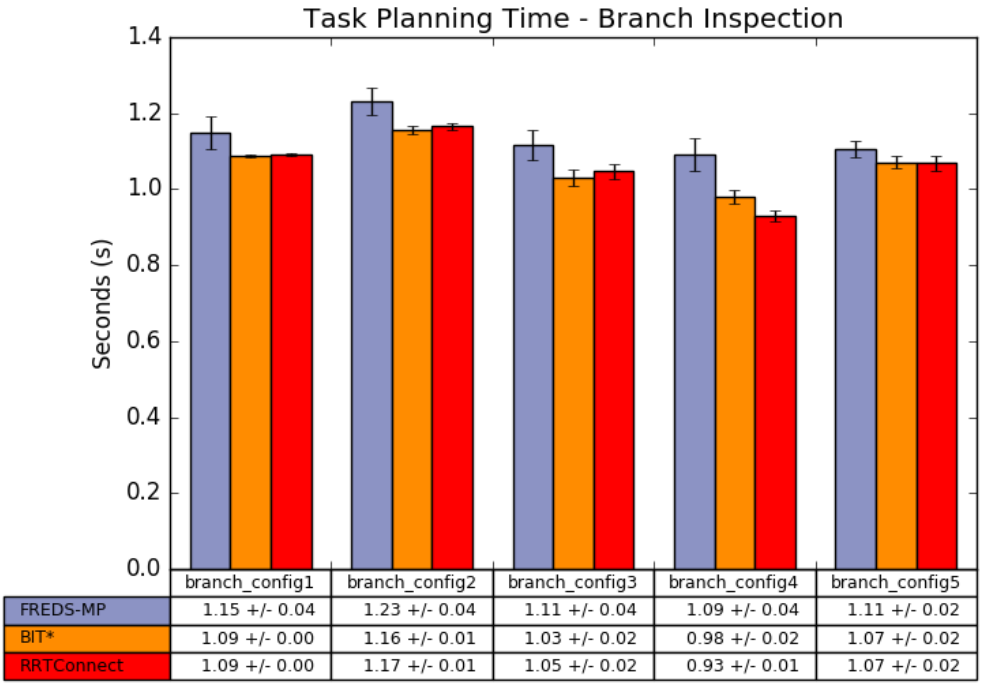
(b)

Figure 6.12 : Planner success mean and SEM plots: (a) “Flat 1-5” trellis configurations, (b) “Branch 1-5” trellis configurations.



	flat_config1	flat_config2	flat_config3	flat_config4	flat_config5
FREDS-MP	1.21 +/- 0.06	1.20 +/- 0.05	1.11 +/- 0.06	1.09 +/- 0.02	1.10 +/- 0.05
BIT*	1.11 +/- 0.03	1.12 +/- 0.02	1.03 +/- 0.02	1.01 +/- 0.02	1.02 +/- 0.02
RRTConnect	1.11 +/- 0.04	1.13 +/- 0.02	1.03 +/- 0.02	1.01 +/- 0.01	1.00 +/- 0.02

(a)



	branch_config1	branch_config2	branch_config3	branch_config4	branch_config5
FREDS-MP	1.15 +/- 0.04	1.23 +/- 0.04	1.11 +/- 0.04	1.09 +/- 0.04	1.11 +/- 0.02
BIT*	1.09 +/- 0.00	1.16 +/- 0.01	1.03 +/- 0.02	0.98 +/- 0.02	1.07 +/- 0.02
RRTConnect	1.09 +/- 0.00	1.17 +/- 0.01	1.05 +/- 0.02	0.93 +/- 0.01	1.07 +/- 0.02

(b)

Figure 6.13 : Task planning time mean and SEM plots: (a) “Flat 1-5” trellis configurations, (b) “Branch 1-5” trellis configurations.

6.4 Discussion

The results from the hardware experiments follow the trends seen in the simulation experiments. In particular, there was not a significant difference in results between the flat and branch trellis setup. The main difference was that the branch configurations suffered slightly lower success rate for BIT* and RRTConnect while FREDS-MP retained 100% success rate for all configurations.

The was some variance across the different apple configurations which suggests that FREDS-MP may be sensitive to the resulting grasp pose, particularly if the orientation is very different from the normal vectors used to create the task space for the offline planner. However, considering the high success rate, this suggests that FREDS-MP is effective at handling errors in the selection of pre-computed trajectories in contrast to previous methods that suffered from such scenarios.

Importantly, FREDS-MP still significantly outperformed BIT* and RRTConnect which validates its application to a real agricultural application.

Chapter 7

Conclusion and Future Work

This chapter concludes the thesis with a summary of why a redundant manipulator and FREDS-MP are suitable for agricultural manipulation tasks. Moreover, the contributions of this thesis will be summarised here in the context of agricultural applications and potentially other applications. Lastly, future work will be suggested in an effort to highlight aspects of FREDS-MP that could be improved to achieve better performance.

7.1 Summary

The initial problem stated that there is a need for a faster, more reliable and efficient method for motion planning in agricultural manipulator applications. FREDS-MP has proven to perform better than standalone current state-of-the-art methods, BIT* and RRTConnect, in all three of these characteristics, both in simulation and on a real robot arm in a real world application. In particular, FREDS-MP showed a significant advantage when computing plans for constrained tasks.

While FREDS-MP performed better than purely online methods overall, it is evident that there are trade-offs between offline and online computing of trajectories. For example, there were some instances when FREDS-MP performed worse. In particular, for scenarios with a small number of unconstrained tasks and simpler obstacles, the online methods performed better due to greater flexibility in available solutions. Further, extra time and effort is required to construct the offline priors and compute the databases. The hardware required to store and load the databases may impose additional constraints.

FREDS-MP was first tested on a simulated Sawyer and UR5 arm in various complex environments. It was shown that the Sawyer, a redundant seven-DOF manipulator, is more well suited to the intended tasks than the UR5, a non-redundant six-DOF manipulator. Hence, FREDS-MP was tested on a real Sawyer arm which

performed active perception and fruit inspection tasks on an artificial apple trellis. FREDS-MP successfully completed all the tasks with 100% success rate and performed consistently better than BIT* and RRTConnect in all measured metrics, thus validating its practicality.

Moreover, the demonstrated unconstrained and constrained tasks are common to other agricultural applications and indeed other applications outside of agriculture, thus FREDS-MP can be adapted to them with ease.

Further to being extended to other applications, FREDS-MP is compatible with any manipulator configuration by leveraging and basing its interfaces on OpenRAVE which can handle various standardised robot model formats seamlessly. Manipulators can be swapped out by simply providing different compatible robot model files, not requiring any further modification.

Importantly, the benefit of FREDS-MP is truly realised in scenarios where a large number tasks need to be performed. This is largely attributed to the fact that FREDS-MP can leverage pre-computed data to make more informed decisions over long planning horizons.

7.2 Contributions

FREDS-MP, a new motion planning framework that leverages current state-of-the-art planning algorithms has been presented. It has been proven to be faster, more reliable and efficient than BIT* and RRTConnect for a range of constrained and unconstrained tasks, in particular for active perception and fruit inspection tasks. Moreover, in contrast to previous agricultural applications, FREDS-MP is a viable method for real-time manipulator planning in complex agricultural environments that considers non-planar obstacles and accurate heuristics for sequencing of tasks. Lastly, a practical comparison has been given for commercially available redundant and non-redundant manipulators, comparing their suitability for performing unconstrained and constrained tasks seen in common agricultural applications.

7.3 Future Work

While efforts have been made to ensure FREDS-MP performs to its potential, there are some obvious but non-trivial improvements that could further increase its perfor-

mance. One improvement would be to better deal with cases when path adapting fails during the online planning phase. In the current implementation the online planner simply discards the task that failed in the sequence and moves onto the next. This is inefficient because the task may have been more easily reached from another task. While this would be more critical in real world applications where 100% success rate is desired, given that FREDS-MP exhibited a generally high success rate, for the purpose of this thesis it was not a priority and would not have had a significant influence on the overall performance.

In terms of evaluation, it would be interesting to try different parameters when constructing the database and analyse their effect on the online planning performance. For example, varying the resolution of the offline task space grid and trying different objective functions when minimising the path costs during the offline construction. It would be valuable to understand the importance of correct selection of pre-computed costs during task planning, possibly by carrying out specific tests to try find correlations between poor performance and large errors in pre-computed cost selection.

Further to making improvements, there are several interesting extensions that could be built on this thesis's work. It would be interesting to model real-time obstacles using sensor data and see how FREDS-MP performs with non basic shape primitives. Further, it was assumed that choosing an appropriate database given a real world obstacle was performed by another process. Potential future work could be to implement this process.

In this thesis, it was assumed for the simulations and hardware experiments that the world was static. In real orchards, the environment is not so well controlled, for example wind can move the trellis. Dealing with uncertainty and dynamic obstacles would be an interesting extension of this work.

Another extension could be to use multiple manipulators to cooperatively cover a larger workspace. Having two arms could provide greater flexibility when sequencing tasks. Moreover, using offline priors could help overcome the significantly increased time complexities associated with multi-coordination planning. An obvious extension would be try different agricultural applications, for example; mechanical weeding, pruning, grasping or even the same hardware experiments on a real apple

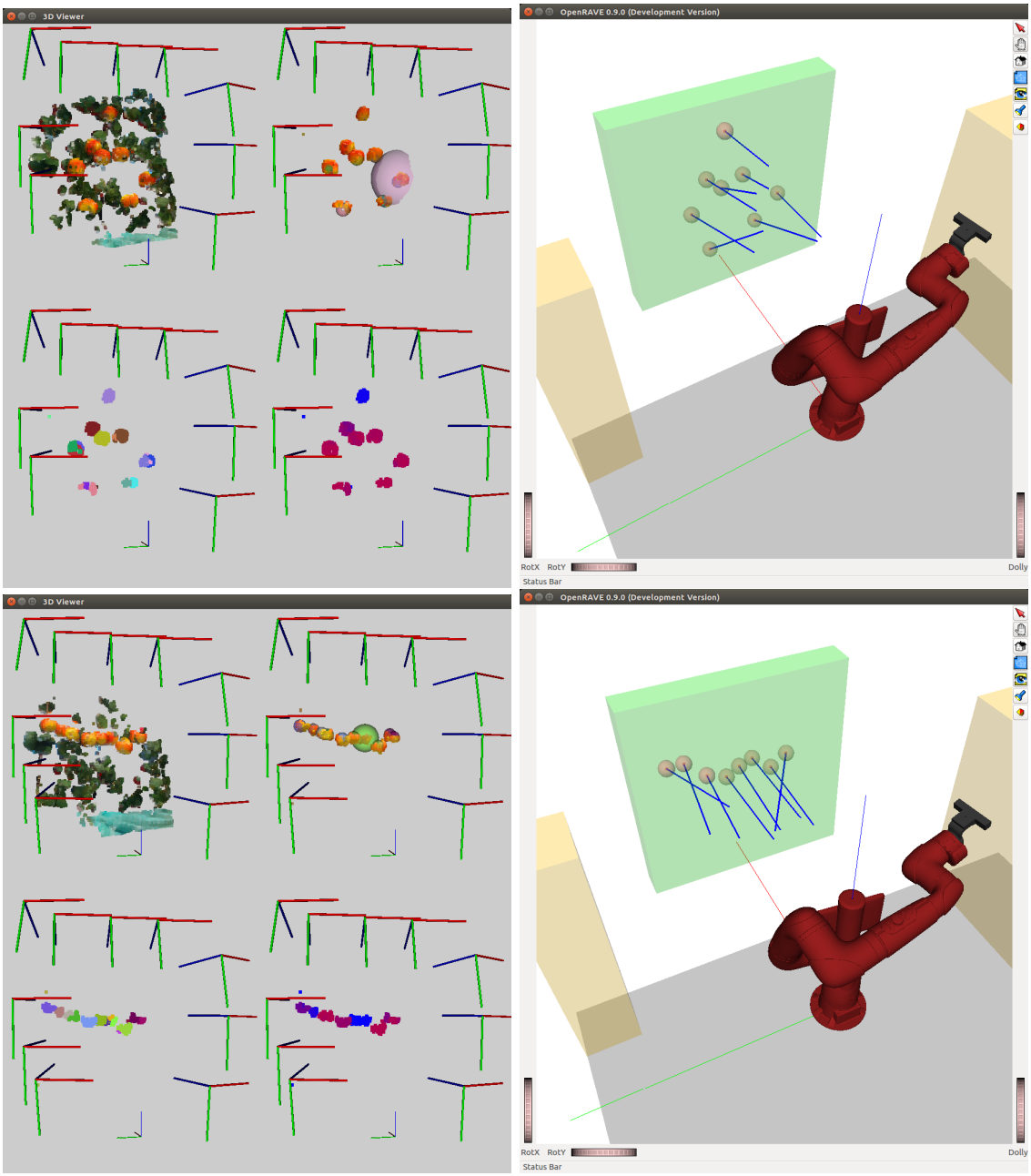
trellis with varying obstacle configurations.

Finally, one of the assumptions in this thesis was that the manipulator's base was manually placed in a "good" position relative from the trellis. It would be interesting to investigate what affect base placement has on the overall performance on the planner and to see if it can be optimised. Moreover, it would be interesting to see what affect a prismatic joint at the base would have on the overall performance of FREDS-MP.

Appendices

Appendix A

- Perception and Planner Outputs



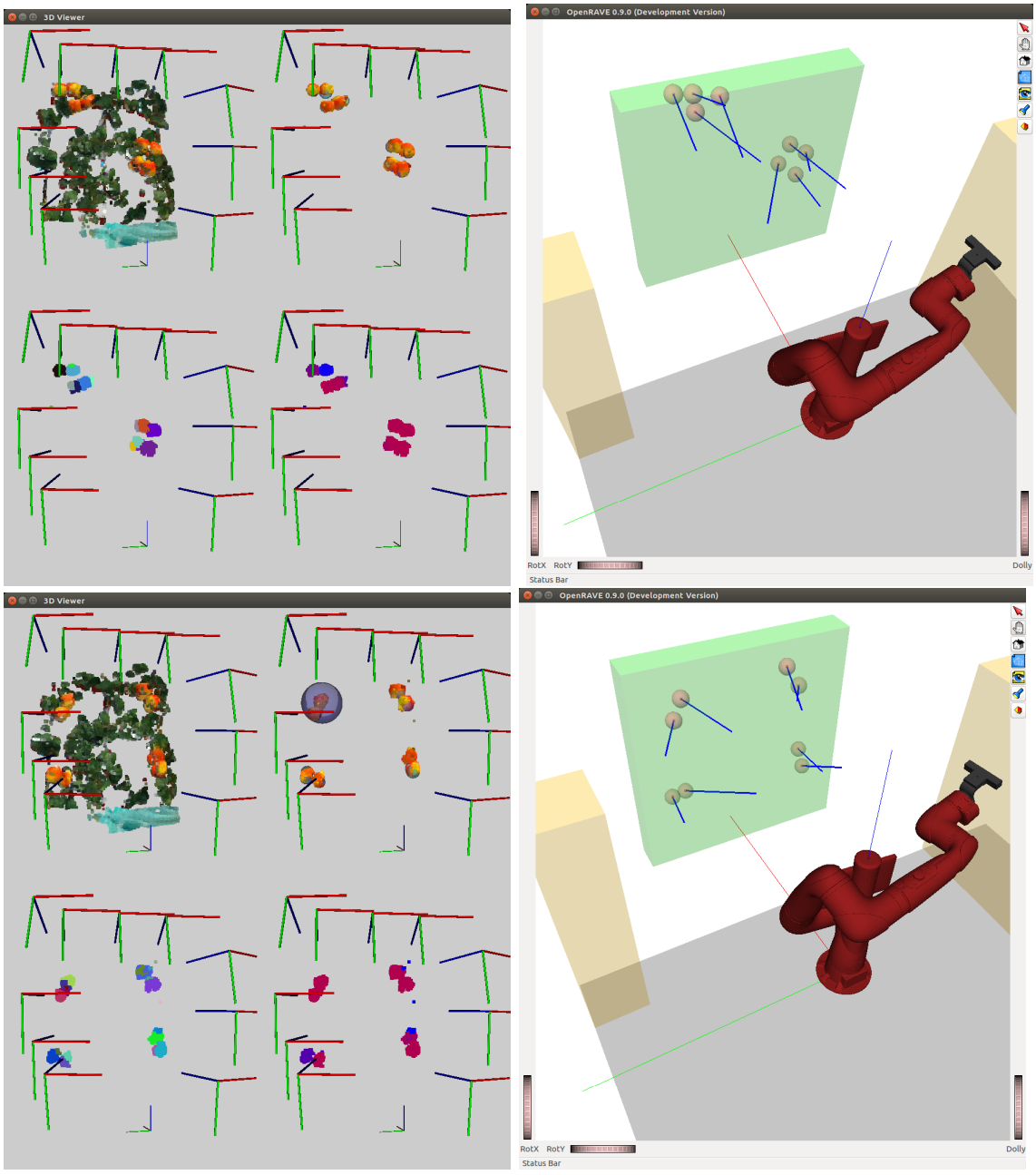
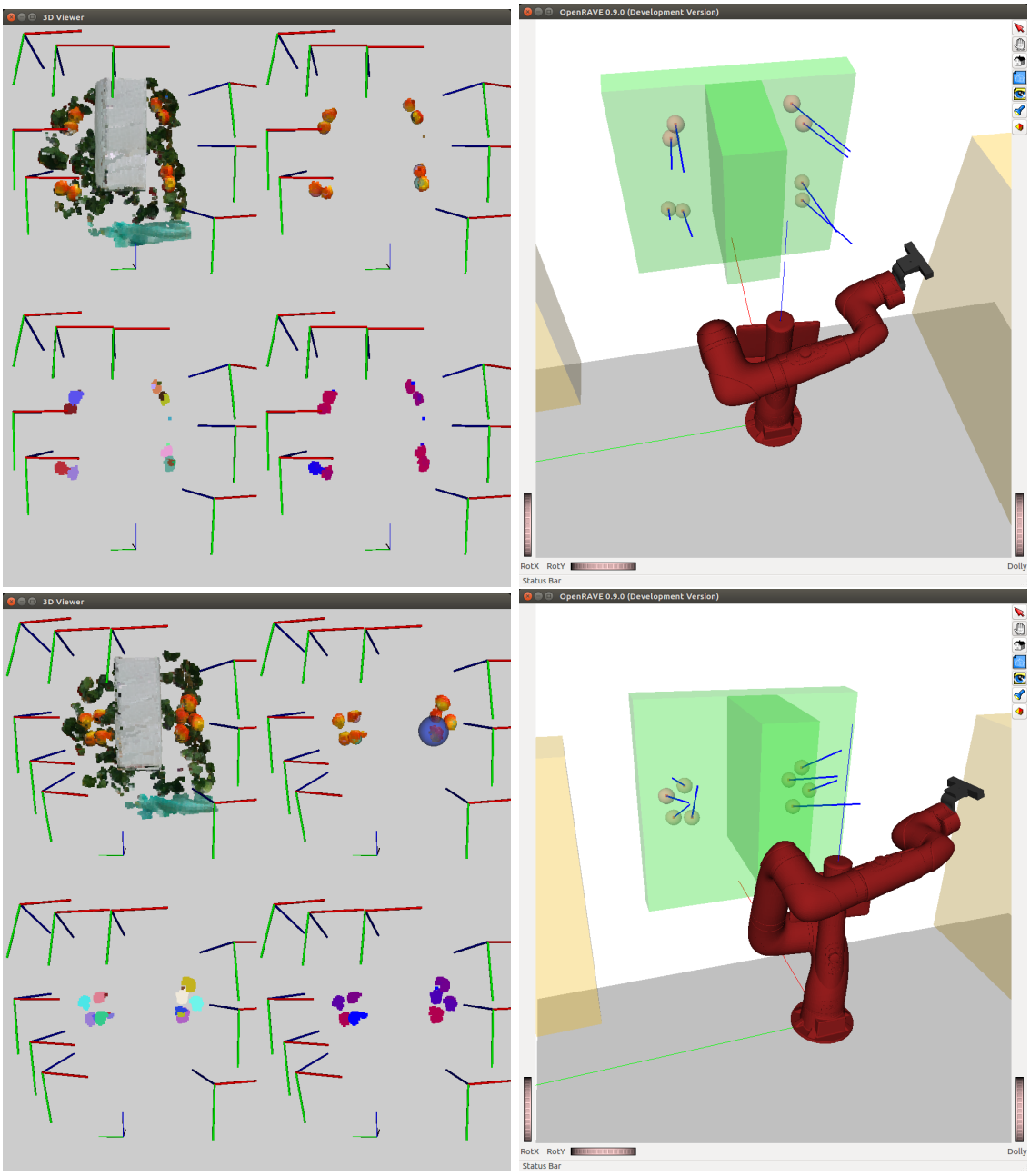


Figure A.1 : Hardware Experiment Visualisation for “Flat 2-5” Trellis Configurations


3D Viewer
OpenRAVE 0.9.0 (Development Version)
RotX RotY Dolly
Status Bar
3D Viewer
OpenRAVE 0.9.0 (Development Version)
RotX RotY Dolly
Status Bar

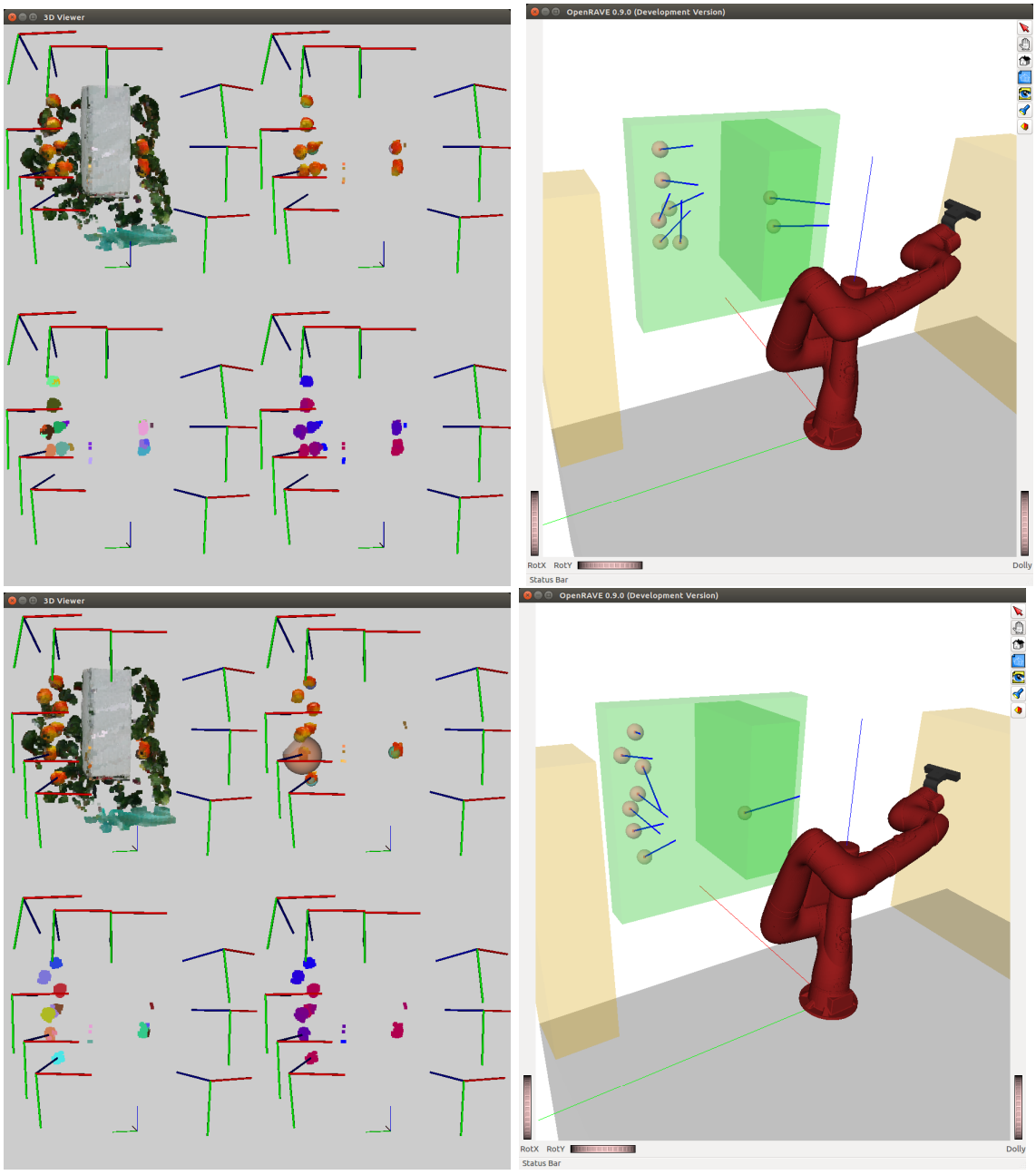


Figure A.2 : Hardware Experiment Visualisation for “Branch 2-5” Trellis Configurations

Bibliography

[1] GRDC, *Insights into future agricultural robotic systems*, Jul. 2014. [Online]. Available: https://grdc.com.au/resources-and-publications/grdc-update-papers/tab-content/grdc-update-papers/2014/07/insights-into-future-agricultural-robotic-systems.

[2] C. W. Bac, E. J. V. Henten, J. Hemming, and Y. Edan, "Harvesting robots for high-value crops: State-of-the-art review and challenges ahead," *Journal of Field Robotics*, vol. 31, no. 6, pp. 888–911, Dec. 2014. DOI: 10.1002/rob.21525.

[3] R. Aedy, *The fourth industrial revolution: Are robots stealing our jobs?* Dec. 2016. [Online]. Available: http://www.abc.net.au/radionational/programs/bestpractice/robots/8107998.

[4] D. of Foreign Affairs and Trade, *Issues related to agriculture*, Jun. 2014. [Online]. Available: http://dfat.gov.au/international-relations/themes/climate-change/submissions/Pages/issues-related-to-agriculture.aspx.

[5] D. Peinecke, C. Forland, and J. H. Wines, *Agricultural workforce report 2013*, May 2015. [Online]. Available: https://fortress.wa.gov/esd/employmentdata/docs/industry-reports/agricultural-workforce-report-2013.pdf.

[6] S. Neales, *Andrew bate's swarmfarm robotics finds a more efficeint way to spray weeds*, Mar. 2015. [Online]. Available: http://www.theaustralian.com.au/business/the-deal-magazine/andrew-bates-swarmfarm-robotics-finds-a-more-efficeint-way-to-spray-weeds/news-story/04c1464922bab1060c89ad38f98ccc92.

[7] R. Oberti, "Advances in robotic agriculture for crops," *Advances in Robotic Agriculture for Crops - ScienceDirect*, Jun. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1537511016302628.

[8] W. Martens, Y. Poffet, P. R. Soria, R. Fitch, and S. Sukkarieh, "Geometric priors for Gaussian process implicit surfaces," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 373–380, Nov. 2016. DOI: 10.1109/lra.2016.2631260.

[9] L.-P. Ellekilde and H. G. Petersen, "Motion planning efficient trajectories for industrial bin-picking," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 991–1004, 2013.

[10] J. J. H. Lee, K. Frey, R. Fitch, and S. Sukkarieh, "Fast path planning for precision weeding," in *In Proc. of ACRA*, 2014.

[11] T. T. Nguyen, E. Kayacan, J. De Baedemaeker, and W. Saeys, "Task and motion planning for apple harvesting robot," *IFAC Proceedings Volumes*, vol. 46, no. 18, pp. 247–252, 2013.

[12] H. M. Choset, *Principles of robot motion: Theory, algorithms, and implementation*. MIT press, 2005.

[13] J. H. Reif, "Complexity of the movers problem and generalizations," *20th Annual Symposium on Foundations of Computer Science*, 1979. DOI: 10.1109/sfcs.1979.10.

[14] J. Canny, "Some algebraic and geometric computations in pspace," *In Proc. of ACM STOC*, 1988. DOI: 10.1145/62212.62257.

[15] A. Hayashi, "Geometrical motion planning for highly redundant manipulators using a continuous model," 1994.

[16] F.-T. Cheng, J.-S. Chen, and F.-C. Kung, "Study and resolution of singularities for a 7-DOF redundant manipulator," *IEEE Transactions on Industrial Electronics*, vol. 45, no. 3, pp. 469–480, 1998. DOI: 10.1109/41.679005.

[17] C. Bac, J. Hemming, and E. V. Henten, "Robust pixel-based classification of obstacles for robotic harvesting of sweet-pepper," *Computers and Electronics in Agriculture*, vol. 96, pp. 148–162, 2013. DOI: 10.1016/j.compag.2013.05.004.

[18] M. Elbanhawi and M. Simic, "On the performance of sampling-based optimal motion planners," *2013 European Modelling Symposium*, 2013. DOI: 10.1109/ems.2013.13.

[19] E. J. Van Henten, J. Hemming, B. Van Tuijl, J. Kornet, J. Meuleman, J. Bontsema, and E. Van Os, "An autonomous robot for harvesting cucumbers in greenhouses," *Autonomous Robots*, vol. 13, no. 3, pp. 241–258, 2002.

[20] S. Hayashi, K. Shigematsu, S. Yamamoto, K. Kobayashi, Y. Kohno, J. Kamata, and M. Kurita, "Evaluation of a strawberry-harvesting robot in a field test," *Biosystems Engineering*, vol. 105, no. 2, pp. 160–171, 2010.

[21] A. J. Scarfe, R. C. Flemmer, H. Bakker, and C. L. Flemmer, "Development of an autonomous kiwifruit picking robot," in *In Proc. of IEEE ICRA*, IEEE, 2009, pp. 380–384.

[22] R. Barth, J. Hemming, and E. J. V. Henten, "Design of an eye-in-hand sensing and servo control framework for harvesting robotics in dense vegetation," *Biosystems Engineering*, vol. 146, pp. 71–84, 2016. DOI: 10.1016/j.biosystemseng.2015.12.001.

[23] C. Lehnert, A. English, C. McCool, A. W. Tow, and T. Perez, "Autonomous sweet pepper harvesting for protected cropping systems," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 872–879, 2017.

[24] J. Baeten, K. Donné, S. Boedrij, W. Beckers, and E. Claesen, "Autonomous fruit picking machine: A robotic apple harvester," in *Field and Service Robotics*, Springer, 2008, pp. 531–539.

[25] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, "Design, integration, and field evaluation of a robotic apple harvester," *Journal of Field Robotics*, 2017. DOI: 10.1002/rob.21715.

[26] *Crops*, 2014. [Online]. Available: http://www.crops-robots.eu/.

[27] *Sweeper*, 2017. [Online]. Available: http://www.sweeper-robot.eu/.

[28] K. Gallardo, M. Taylor, and H. Hinman, *Washington state university extension*. [Online]. Available: https://pubs.wsu.edu/ItemDetail.aspx?ProductID=15371.

[29] D. C. Cloutier and M. L. Leblanc, "Mechanical weed control in agriculture," in *Physical Control Methods in Plant Protection*, Springer, 2001, pp. 191–204.

[30] B. Whelan and J. Taylor, *Precision agriculture for grain production systems*. CSIRO Publishing, 2013.

[31] M. K. Upadhyaya and R. E. Blackshaw, *Non-chemical weed management: Principles, concepts and technology*. Cabi, 2007.

[32] O. Cohen and B. Rubin, "11 soil solarization and weed management," *Non-chemical Weed Management*, p. 177, 2007.

[33] J. Blasco, N. Aleixos, J. Roger, G. Rabatel, and E. Molto, "AE-automation and emerging technologies: Robotic weed control using machine vision," *Biosystems Engineering*, vol. 83, no. 2, pp. 149–157, 2002.

[34] F. K. van Evert, J. Samsom, G. Polder, M. Vijn, H.-J. v. Dooren, A. Lamaker, G. W. van der Heijden, C. Kempenaar, T. van der Zalm, and L. A. Lotz, "A robot to detect and control broad-leaved dock in grassland," *Journal of Field Robotics*, vol. 28, no. 2, pp. 264–277, 2011.

[35] H. Y. Jeon and L. F. Tian, "Direct application end effector for a precise weed control robot," *Biosystems Engineering*, vol. 104, no. 4, pp. 458–464, 2009.

[36] T. Botterill, S. Paulin, R. Green, S. Williams, J. Lin, V. Saxton, S. Mills, X. Chen, and S. Corbett-Davies, "A robot system for pruning grape vines," *Journal of Field Robotics*, 2016.

[37] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *In Proc. of IEEE ICRA*, IEEE, vol. 2, 2000, pp. 995–1001.

[38] C. W. Bac, T. Roorda, R. Reshef, S. Berman, J. Hemming, and E. J. V. Henten, "Analysis of a motion planning problem for sweet-pepper harvesting in a dense obstacle environment," *Biosystems Engineering*, vol. 146, pp. 85–97, 2016. DOI: 10.1016/j.biosystemseng.2015.07.004.

[39] C. W. Bac, J. Hemming, B. Tuijl, R. Barth, E. Wais, and E. J. Henten, "Performance evaluation of a harvesting robot for sweet pepper," *Journal of Field Robotics*, 2017.

[40] Y. Edan, D. Rogozin, T. Flash, and G. Miles, "Robotic melon harvesting," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 831–835, 2000. DOI: 10.1109/70.897793.

[41] F. Hauer and P. Tsiotras, "Deformable rapidly-exploring random trees,"

[42] E. Van Henten, D. Van't Slot, C. Hol, and L. Van Willigenburg, "Optimal manipulator design for a cucumber harvesting robot," *Computers and Electronics in Agriculture*, vol. 65, no. 2, pp. 247–257, 2009.

[43] C. Lehnert, T. Perez, and C. McCool, "Optimisation-based design of a manipulator for harvesting capsicum," in *In Proc. of IEEE ICRA*, IEEE, 2015.

[44] J. Baur, J. Pfaff, H. Ulbrich, and T. Villgrattner, "Design and development of a redundant modular multipurpose agricultural manipulator," in *In Proc. of IEEE/ASME AIM*, IEEE, 2012, pp. 823–830.

[45] S. Han, S. Xueyan, Z. Tiezhong, Z. Bin, and X. Liming, "Design optimisation and simulation of structure parameters of an eggplant picking robot," *New Zealand Journal of Agricultural Research*, vol. 50, no. 5, pp. 959–964, 2007.

[46] B. Sivaraman and T. Burks, "Geometric performance indices for analysis and synthesis of manipulators for robotic harvesting," *Transactions of the ASABE*, vol. 49, no. 5, pp. 1589–1597, 2006.

[47] B. Sivaraman and T. F. Burks, "Robot manipulator for citrus harvesting: Configuration selection," in *ASAE Annual Meeting*, American Society of Agricultural and Biological Engineers, 2007, p. 1.

[48] N. Kondo, M. Monta, Y. Shibano, and K. Mohri, "Session 7 automation & robotics in agriculture: Basic mechanism of robot adapted to physical properties of tomato plan," *ICAMPE*, vol. 3, pp. 840–849, 1993.

[49] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[50] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[51] A. Kanarat, "Motion planning and robust control for nonholonomic mobile robots under uncertainties," PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[52] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *In Proc. of IEEE ICRA*, IEEE, vol. 1, 2002, pp. 968–975.

[53] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *In Proc. of ICAPS*, 2005, pp. 9–18.

[54] D. Ferguson and A. Stentz, "Field d*: An interpolation-based path planner and replanner," *Robotics Research*, pp. 239–253, 2007.

[55] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," in *AAAI*, 2007, pp. 1177–1183.

[56] A. Nash, S. Koenig, and C. Tovey, "Lazy theta*: Any-angle path planning and path length analysis in 3d," in *Third Annual Symposium on Combinatorial Search*, 2010.

[57] P. Yap, N. Burch, R. C. Holte, and J. Schaeffer, "Block A*: Database-driven search with applications in any-angle path-planning," in *AAAI*, 2011.

[58] D. Harabor and A. Grastien, "An optimal any-angle pathfinding algorithm," *In Proc. of ICAPS*, 2013.

[59] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, and et al., "Junior: The stanford entry in the urban challenge," *Springer Tracts in Advanced Robotics The DARPA Urban Challenge*, pp. 91–123, 2009. DOI: 10.1007/978-3-642-03991-1_3.

[60] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *In Proc. of RSS*, 2010. DOI: 10.15607/rss.2010.vi.034.

[61] 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, no. 4, pp. 566–580, 1996.

[62] T. Sim'eon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[63] V. Boor, M. H. Overmars, and A. F. Van Der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *In Proc. of IEEE ICRA*, IEEE, vol. 2, 1999, pp. 1018–1023.

[64] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *In Proc. of IEEE ICRA*, IEEE, vol. 1, 2000, pp. 521–528.

[65] D. Hsu, "Randomized single-query motion planning in expansive spaces," PhD thesis, Stanford University USA, 2000.

[66] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *In Proc. of IEEE ICRA*, IEEE, vol. 3, 1997, pp. 2719–2726.

[67] J. M. Phillips, N. Bedrossian, and L. E. Kavraki, "Guided expansive spaces trees: A search strategy for motion-and cost-constrained state spaces," in *In Proc. of IEEE ICRA*, IEEE, vol. 4, 2004, pp. 3968–3973.

[68] I. A. Sucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[69] G. Sanchez and J.-C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *The International Journal of Robotics Research*, vol. 21, no. 1, pp. 5–26, 2002.

[70] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *In Proc. of IEEE ICRA*, IEEE, 2009, pp. 625–632.

[71] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[72] M. Stollenga, L. Pape, M. Frank, J. Leitner, A. Forster, and J. Schmidhuber, "Task-relevant roadmaps: A framework for humanoid motion planning," in *In Proc. of IEEE/RSJ IROS*, IEEE, 2013, pp. 5772–5778.

[73] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *In Proc. of IEEE ICRA*, IEEE, 2010, pp. 2493–2498.

[74] J. Pan, L. Zhang, and D. Manocha, "Fast smoothing of motion planning trajectories using b-splines," in *Robotics: Science and Systems*, 2011.

[75] A. Yershova and S. M. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 151–157, 2007.

[76] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.

[77] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *In Proc. of IEEE ICRA*, IEEE, vol. 1, 1999, pp. 688–694.

[78] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT," in *In Proc. of IEEE/RSJ IROS*, IEEE, 2011, pp. 3513–3518.

[79] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.

[80] A. Nash, S. Koenig, and M. Likhachev, "Incremental Phi*: Incremental any-angle path planning on grids.," in *In Proc. of IJCAI*, 2009.

[81] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[82] M. Likhachev and A. Stentz, "R* search," *Lab Papers (GRASP)*, p. 23, 2008.

[83] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2004, pp. 767–774.

[84] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *In Proc. of IEEE/RSJ IROS*, IEEE, 2014, pp. 2997–3004.

[85] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[86] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *In Proc. of IEEE ICRA*, IEEE, 2013, pp. 2421–2428.

[87] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *In Proc. of IEEE ICRA*, IEEE, 2015, pp. 3067–3074.

[88] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *In Proc. of ICML*, ACM, 2005, pp. 89–96.

[89] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *In Proc. of IEEE ICRA*, IEEE, 2009, pp. 489–494.

[90] Z. Shiller and S. Dubowsky, "On computing the global time-optimal motions of robotic manipulators in the presence of obstacles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 785–797, 1991.

[91] S. Sundar and Z. Shiller, "Optimal obstacle avoidance based on the hamilton-jacobi-bellman equation," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 2, pp. 305–310, 1997.

[92] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *In Proc. of IEEE ICRA*, IEEE, 1993, pp. 802–807.

[93] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *In Proc. of IEEE ICRA*, IEEE, 2011, pp. 4569–4574.

[94] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *In Proc. of RSS*, vol. 9, 2013, pp. 1–10.

[95] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs," in *In Proc. of RSS*, 2016.

[96] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, no. 2, pp. 108–120, 1983.

[97] M. Koval and P. Velagapudi, *Or_trajopt*, Aug. 2016. [Online]. Available: `https://github.com/personalrobotics/or_trajopt`.

[98] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, *Orgpmp2*, Jun. 2017. [Online]. Available: `https://github.com/gtrll/orgpmp2`.

[99] C. Dellin, *Or_cdchomp*, Jul. 2016. [Online]. Available: `https://github.com/personalrobotics/or_cdchomp`.

[100] R. Diankov and J. Kuffner, *Introduction to the openrave architecture 0.9.0*. [Online]. Available: `http://openrave.org/docs/latest%5C_stable/coreapihtml/architecture%5C_concepts.html`.

[101] R. Diankov, *Openrave*, 2017. [Online]. Available: `https://github.com/rdiankov/openrave`.

[102] M. Koval, *Constrained manipulation planning suite*, Aug. 2016. [Online]. Available: `https://github.com/personalrobotics/comps`.

[103] M. Koval, C. Dellin, and M. Klingensmith, *Or_ompl*, Sep. 2016. [Online]. Available: `https://github.com/personalrobotics/or_ompl`.

[104] M. Koval and C. Dellin, *Prpy*, Apr. 2017. [Online]. Available: `https://github.com/personalrobotics/prpy`.

[105] R. Robotics, *Sawyer robot*, Mar. 2017. [Online]. Available: `https://github.com/RethinkRobotics/sawyer_robot`.

[106] R. Industrial, *Universal_robot*, Sep. 2017. [Online]. Available: `https://github.com/ros-industrial/universal_robot`.

[107] M. Koval and P. Velagapudi, *Or_urdf*, Apr. 2017. [Online]. Available: `https://github.com/personalrobotics/or_urdf`.

[108] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *In Proc. of IEEE ICRA*, IEEE, 2012, pp. 3859–3866.

[109] Google, *Solving TSPs with OR-Tools*, Sep. 2017. [Online]. Available: `https://developers.google.com/optimization/routing/tsp/tsp`.

[110] *Swarmfarm robotics*, 2017. [Online]. Available: `http://www.swarmfarm.com/`.