Real-Time Motion Planning and Execution of a 6DoF Manipulator

R.B. Burger

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Real-Time Motion Planning and Execution of a 6DoF Manipulator

Master of Science Thesis

For the degree of Master of Science in Electrical Engineering at Delft University of Technology

R.B. Burger

June 3, 2016

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) · Delft University of Technology

The work in this thesis was supported by Technolution BV. Their cooperation is hereby gratefully acknowledged.

Abstract

In recent years, the desire to use manipulators in close collaboration with human workers has been a driving force behind a lot of robotics research. As part of a research project, Technolution BV has built a robotic pick and place system using the Universal Robots UR5, a 6 DoF manipulator. It was recognized that to realize the full potential of a robotic manipulator, it would have to be able to respond to unforeseen changes in the environment like dynamic obstacles or a moving target. Following these observations, the project was continued and aimed at researching methods that can be used to formulate a controller that is capable of adapting to a changing environment.

A thorough analysis of the technical issues and theoretical background related to dynamic motion execution is presented in Part 1. Based on an examination of state of the art pick and place methods, a pick (or a place) motion is chosen to be dissected in two distinct parts: a *global* motion, which is used to move the end-effector to a pose close to the target object, and a *Cartesian* approach, to move towards the target object in a straight line and perform the actual grasp.

For the global motion, a fast geometric path planner is a vital component. Several state of the art path planning algorithms are discussed and benchmarked with the specified scenario in mind. In order to ensure that a well performing configuration of each path planner is used, a novel tuning method based on automatic algorithm configuration tools was developed. An analysis of the performance of optimal planners was found to offer too few benefits for use in a system that requires fast reactions.

To allow the Cartesian approach to be used when the target object is in motion, two different trajectory execution methods are tested. The first attempts to conform to existing infrastructure and uses cubic interpolation for determining a setpoint. This was found to offer accurate path execution and decent tracking performance. The 2$^{\text{nd}}$ method uses an on-line trajectory generation algorithm and was found to improve the tracking performance but did so using jerkier motions.

For collision avoidance during the global motions, a time-scaling method has been combined with a replanning approach. Combining the global execution method with pose tracking, a system is proposed that is capable of responding to any changes in the environment.

Table of Contents

Chapter 1

Introduction

Since beginning of the Palaeolithic age around 2.5 million years ago, the use of tools has been deeply embedded in human nature. In the industrial revolution, the drive to automation is what lead to a large number of developments that brought us where we are today. While the use of robotic manipulators is already widespread, it is obvious that their potential to even further automation is much greater. A popular topic in science fiction, many believe artificially intelligent robots will someday prove superior to humans in many areas.

This seemingly inherently human desire to automation has lead to research towards robots that can safely and effectively work in cooperation with people being one of key areas in robotics research today. A humanoid robot that could serve as a drop in replacement to a human operator performing any arbitrary task would a tremendous step towards levels of automation currently unheard of.

One of the main differences between a robot and a human lies in the flexibility with which people are able to perform tasks. Robots are already magnitudes better when it comes to very accurately performing a well-defined task over and over again. However, when something unexpected happens, a person will be able to quickly devise an alternative strategy and successfully finish the task at hand. Achieving such flexibility in a robotic manipulator is an active area of research. With a humanoid robot the final goal, a robotic arm that is able to perform manipulation tasks with such flexibility can be considered as a very significant step in the right direction.

1-1 Background

As a technology integrator, Technolution B.V. realizes innovations for its clients and is able to handle each stage of the process, from concept to end product or service. With a large multi-disciplinary team, Technolution is proficient in developing electronic hardware, programmable logic systems or (embedded) software solutions to a wide variety of problems. Besides working with clients, the company continuously tries to explore new technological area's that are expected to play a big role in the technological landscape of the future. As part of a research project that explored the possibilities of robotic manipulators, a pick and place system was built as a demonstrator. A photograph of the setup can be seen in Figure 1-1.

Figure 1-1: Pick and place system at Technolution

The central part of this setup is a Universal Robots UR5, a 6DoF serial chain manipulator that is known to be a "collaborative robot". This means that the UR5 has certain safety features which allow the manipulator to be used without a protective cage, something that is commonly required when using industrial manipulators. These safety features and a comparatively low price make the UR5 a popular choice for many robotics labs and commercial system developers.

The UR5 is controlled using Robotics Operating System, or ROS[1]. ROS is a framework for writing robotics software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Although a large part of the users of ROS consist of academics, the ROS-Industrial consortium attempts to bridge the gap between academia and industry.

For the UR5, the goal was specified to be a pick and place task where Jenga blocks would need to be picked up from one side of the table, and stacked into a tower on the other end. A perception pipeline based around the Microsoft Kinect 2 RGBD camera was installed to provide the input to the robot controller. After carefully assessing the performance of this

system, a decision was made as to where to proceed. Consider Figure 1-2 for an overview of the system.

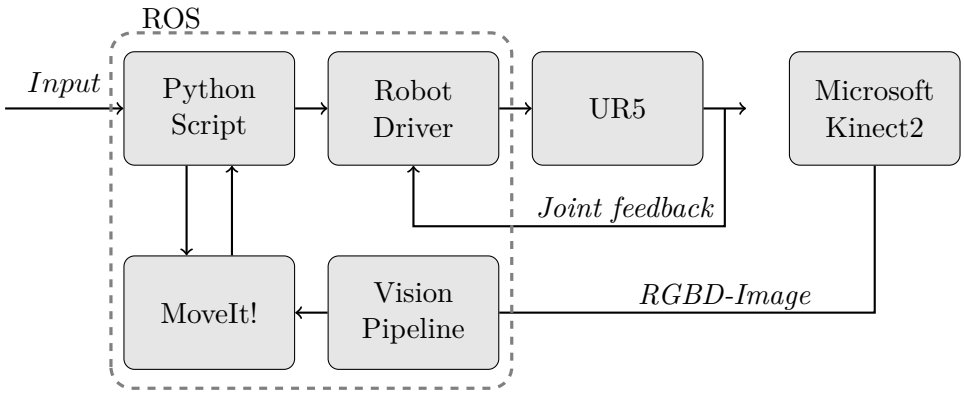


Figure 1-2: System topology for the pick and place pipeline developed at Technolution prior to the start of this project.

1-2 Research goals

One thing that was quickly noticeable about the system was the lack of reactivity to any changes in the environment. Whenever something would change during the execution, like an obstacle appearing, or the manipulator dropping an object that it was carrying, there would be no response. For a manipulator to work in cooperation with a person, it needs to be able to respond to a changing environment. In order to safely operate among a person, it is very important that the manipulator avoids collisions at all costs. In addition, to enable a hand-over between the manipulator and a person, dynamically tracking a target will also be required. These observations were then used to formulate the following requirements:

1. Grasp objects in all feasible locations of the workspace
2. Move in a predictable manner
3. Avoid both static and dynamic obstacles in real-time
4. Track a target object in real-time using smooth motions

Most motion control methods that are currently in use are only capable of dealing with the static constraints. The problem gets a lot more challenging whenever dynamic obstacles and tasks are involved. Specifying a hard requirement on the 'real-time' capabilities of a manipulator is not an easy task. As a starting point, one can take the performance of a person in such scenarios. Considering a person, there are a number of things that come into play. While the time that it takes to go from perception to action is about 200 to 500ms for most people, this is not the complete picture. People predict the trajectory of an obstacle or goal and adjust accordingly. An algorithm that plans a path assuming a static environment

every 100ms does not necessarily offer satisfactory performance. However, specifying an upper bound of **200ms** for responding to any changes seems like a reasonable target.

In order to perform these tasks on an actual robot, the robots perception of environment is one of the key parts. For this research, the perception part is mostly abstracted. An accurate world model is assumed in order to separate the perception from the control method. The environment will be modeled as it is present at Technolution. An important limitation on the control method is that there is no prior information on any of the changes of the environment. The main drawback from completely separating the perception and control is that there will always be errors and uncertainties in the generated world-model that will have influence on the control method. Specialized control methods have been researched that are more suitable to deal with these uncertainties, but these will be disregarded for this research.

1-3 Thesis outline

This thesis is structured in the following 3 parts.

Part I, explores the required theoretical background that is needed to fully analyze the problem. In Chapter 2, the path planning problem is more thoroughly defined and extended with the challenges related to planning in a dynamic environment as well as a review of state of the art pick and place systems. Afterwards, Chapter 3 provides an overview of path planning algorithms that are commonly used in practice, and several newer approaches that might be applicable to dynamic motion planning. Known methods that are used for dynamic motion execution are discussed in Chapter 4.

Part II describes the method that has been used the benchmark several planners with a specified pick and place scenario in mind. The methodology of the benchmarking is detailed in Chapter 5 and the results of the benchmark are presented in Chapter 6.

Finally, Part III will present simulation results of several different dynamic control methods. Chapter 7 will detail the experimental setup and system architecture. Methods that can be used to track a moving target are discussed in Chapter 8, and methods concerning avoiding obstacles are discussed in Chapter 9. Finally, a conclusion and recommendation for further research can be found in Chapter 10.

Part I

Theoretical background

Part 1 will focus on providing the theoretical background that is necessary to fully understand the problem. It is obvious that formulating a motion controller that is capable of reaching the specified performance is not an easy task and requires an in depth analysis of each aspect. In order to get a better understanding of the problem, Chapter 2 will explore and define the problem. As a path planner is an important part of a pick and place system, Chapter 3 will focus on detailing several widely used and state of the art path planning algorithms and see if they can be leveraged for this research. Dynamical control methods that are often used for obstacle avoidance are discussed in Chapter 4.

Chapter 2

Problem definition

In this chapter, the problem that is outlined in the research requirements will be explored and defined. Initially, the path planning problem will be defined in Section 2-1, after which Section 2-2 will consider the problems related to planning in the presence of dynamic obstacles and events. Then, Section 2-3 will discuss state of the art pick and place methods in order to further analyze the problem.

2-1 Path planning

Planning collision-free motions is one of the most fundamental tasks in robotics. A motion can be described as the process of moving from some initial state to a goal state over time. For a rigid body in 3D space, its state can be expressed as a pose, which consists of a translation and a rotation. Together, these coordinates can be denoted as the Special Euclidean 3 group, $SE(3)$. A motion can be described as an ordered set of time-parameterized waypoints, in $SE(3)$ this means a set of poses denoted $q_i \in SE(3)$, starting at $q_{init} \in SE(3)$, moving through $q \in SE(3)$ to finally arrive at the final pose $q_{goal} \in SE(3)$.

For a robot arm, the goal of a motion planning query is often to reach a certain pose of the end-effector, like getting the gripper to a specified pose to grasp an object. As the pose of a rigid body can be expressed in 3 translation and 3 rotation coordinates for a total of 6 coordinates, the *workspace*, the space in which all end effector poses can be expressed, of a manipulator is said to be 6-dimensional. The space that contains all possible states of the robot, called the *configuration*, describes each states as a value for each joint position of the manipulator. This means that for a robot with n joints, this is an n-dimensional space. A more important specification of a robotic manipulator is its *degrees of freedom*, or DoF. For a serial chain manipulator, like most industrial robot arms, each joint represents an added degree of freedom. As the UR5 has 6 revolute joints and a serial chain kinematic structure, it is a 6DoF robot. This also means that for these types of robot, the number of degrees of freedom is equal to the amount of joints. The number of DoF in relation to the dimension of the workspace can provide valuable information on the reachability of the robot. Consider the following 3 different scenario's.

- **DoF == workspace dimension** The manipulator is holonomic. This means that in theory, each pose of the end-effector can be reached, but only in a limited amount of ways.
- **DoF < workspace dimension** The manipulator is be non-holonomic, which means that a large set of end-effector poses cannot be reached by this manipulator
- **DoF > workspace dimension** The manipulator is redundant, as it has more joints that it strictly *needs*. This means that there are multiple ways to reach a certain end-effector pose. As en example, A human arm has 7 DoF and is redundant, allowing us to have more flexibility in constraint spaces

As the UR5 has 6 DoF, it should be able to move the end-effector to almost all poses in the workspace.

One way of finding a path would be to consider the end-effector pose and try to find a path between the initial and goal pose. This would then provide a path that the gripper could follow in order to arrive at the goal. For something like a quadcopter, this could be a valid path, but for a manipulator things are a little more complicated. Unfortunately, such a paths does not take any of the manipulator kinematics into account. For instance, some end-effector poses might not be reachable because they are invalidated by self-collisions or collisions with obstacles. In order to actually execute such a path, corresponding joint states will need to be calculated for each waypoint.

In contrast to forward kinematics, a general solution to the inverse kinematics problem does not exist. A common library used for IK is OROCOS KDL[2]. KDL uses a pseudo inverse of the Jacobian to iteratively seek out an approximate solution. Analytical solvers do exist for specific kinematic structures. An example are those generated by IKFast[3], which outperform approximate approaches but need to be especially generated for each new robot. A recently introduced IK solver, called TRAC-IK[4], uses KDL and several different Sequential Quadratic Programming approaches to improve the solve rates of KDL without relying on a custom generated solver. As an additional benefit, TRAC-IK can be used to optimize towards configurations with certain characteristics, for instance minimizing the configuration space distance norm with respect to a given seed.

When IK is successful in finding the joint configurations, they can serve as waypoints for the robot arm to travel through towards the goal. See Figure 2-1 for a diagram that shows a planning query.

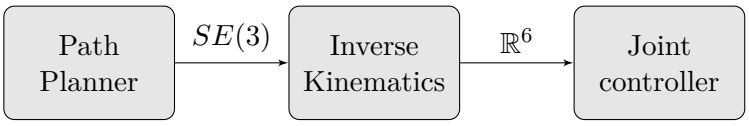


Figure 2-1: Flow diagram of a path planning query in $SE(3)$

Another option is to perform the planning in the configuration space. Figure 2-2 shows such a planning query. A challenging aspect of path planning is state validity checking, or collision checking. In order to plan a path it is vital to know that all configurations the manipulator passes through are valid, which means that no part of the robot is ever in collision with

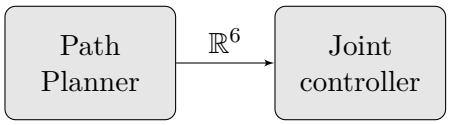


Figure 2-2: Flow diagram of a path planning query in a 6 dimensional configuration space, $\mathbb{R}^6$.

either obstacles or parts of the robot itself. When planning a the end-effector path in the workspace, obstacles can easily be avoided and the problem of self collisions is left to the IK solver. As for planning in the configuration space, it turns out that it is very difficult to map the workspace obstacles onto the configuration space. In most cases, forward kinematics are used to transform each link of manipulator to the workspace, and collision checking is done there.

Consider Figure 2-3 as a demonstration on the non-intuitive representations of obstacles in the configuration space. Both spaces are divided into a free space and a (complementary) obstacle space. As the name implies, the free space consists of all valid configurations of the manipulator, whereas the obstacle space consists of the set of configurations that are invalidated by a collision. Especially in high dimensional robots, this can be a significant portion of the complete configuration space. Despite the problems related to the obstacle

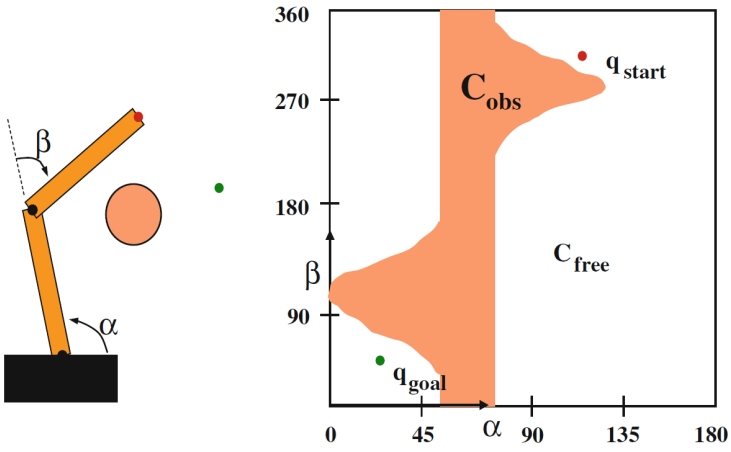


Figure 2-3: The work and configuration space of a 2D manipulator including the free and obstacle space[5].

space, most planning is done in the configuration space. Random sampling based planners, as will be discussed in following chapters, are very efficient in sampling new possible states and quickly checking if they are valid. Finding a path in the workspace, also called *Cartesian* planning, has several application area's where it is much better suited, as will also be discussed in later sections.

As the general overview of the planning problem is now clear, the next step is to examine the research goals to find out how they affect the requirements on the path planning. The following concepts are important when discussing requirements on the planning methods:

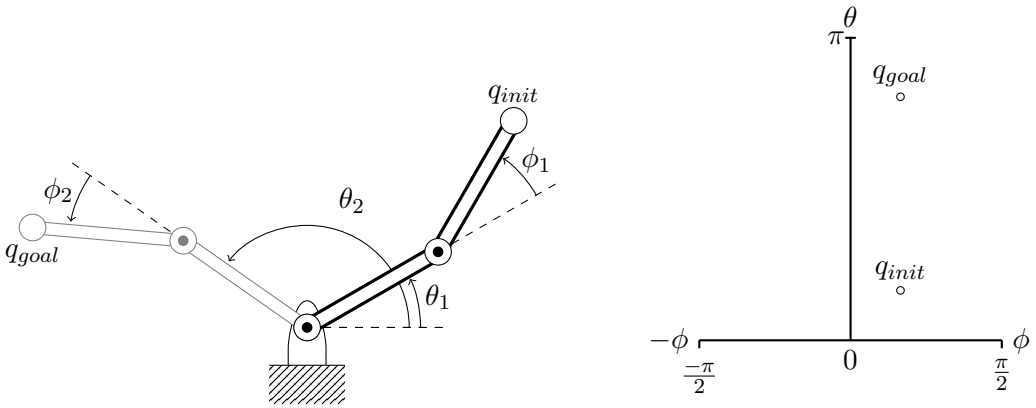
- **Completeness** A planning algorithm is said to be complete if it is always able to find

a feasible path if one exists.

- **Optimal Planning** A path is optimal when it is found by minimizing a specified cost function.

In order to reach all feasible states of the workspace, like specified in requirement 1, completeness is a desirable property. Optimal planners can theoretically be leveraged in order to steer the planner towards a certain type of path.

When it comes to optimality however, it is important to keep in mind the meaning of the cost function. In the context of path planners, the optimal path is often the *shortest* path from the initial state to the goal state. As planning will take place in the configuration space, one logical cost function is to penalize large distances in the configuration space. However, a path that is short in the configuration space does not have to be a short path in the workspace. As an example consider query as shown in Figure 2-4. The initial state is denoted with $q_{initial}$ and the goal state with q_{goal}.



(a) Workspace **(b)** Configuration space

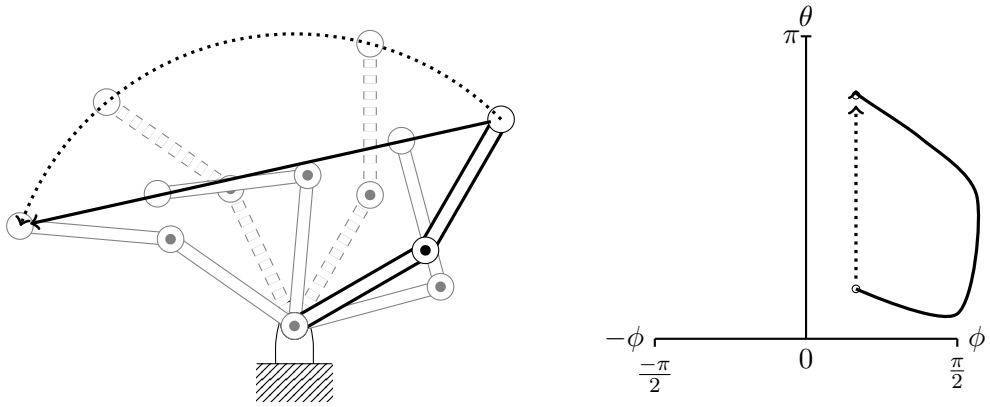
Figure 2-4: A simple planning problem for a 2D manipulator.

To demonstrate the difference between the shortest path in the workspace and the shortest path in the configuration space, consider Figure 2-5.

For a natural looking movement, it is important that the path is short according to *some* metric. When planning with a suboptimal planner, the planner will occasionally find a path that erratically moves back and forth for a short while before moving towards the goal. While this is obviously unwanted behavior, it is unclear what an adequate cost function would be to force a planner towards 'natural' looking movement. Further research will be needed in order to make such an assessment.

2-2 Dynamic motion execution

In the previous section the basic concepts of motion planning were introduced. This section will describe problems that arise when attempting to execute these paths.



(a) Workspace **(b)** Configuration space

Figure 2-5: the shortest path in workspace (solid line) and the configuration space (dashed line) for a robotic manipulator with 2 rotating joints.

The first distinction that needs to be made is between the *path* and the *trajectory*. A *path* consists of a set of ordered waypoints that, when connected, form a geometric path between the start and goal configurations. A *trajectory* also consists of such waypoints, but they are now parameterized by a variable that represents time. In order to execute a path on a manipulator, it will first need to be turned into a valid trajectory. Aside from self collisions, other constraints like joint speeds and accelerations will need to be considered when applying time-parameterization to a path.

Most planners plan a path instead of a trajectory. One of the reasons that planning a trajectory is a much harder problem is because the planner needs to take all differential and kinematic constraints of the manipulator into account. Planners that are able to do so are called *kinodynamic* planners. Instead of planning a path, they account for constraints such as joint limits, accelerations, velocities and plan a trajectory. In order to do so, these planners cannot simply sample a new configuration that is close by and assume it is possible to get there, instead, physical simulation is needed to ensure all kinodynamic constraints will be satisfied. Although theoretically a more complete solution to the planning problem, kinodynamic planners have so far been limited to simulation scenarios as planning times can reach several minutes[6].

A more common approach is to decouple the path planning from the time parameterization as demonstrated in Figure 2-6.

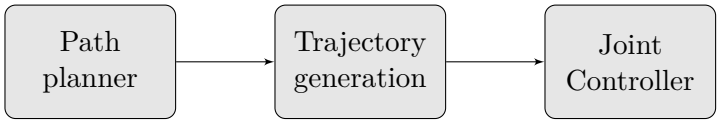


Figure 2-6: Decoupling the path and trajectory generation

The drawbacks of decoupling the path planning and trajectory generation are dependent on the application. Especially when required to perform fast motions, a certain geometric paths

might not be able to reach the goal state in time without violating certain joint constraints. In cases where fast motions are not needed, this aspect of decoupling will not be encountered. Despite some limitations, decoupling the geometric planning and time parameterization is often used in practice.

To get a better overview of what kind of approaches are possible for performing dynamic motion planning, a more extensive analysis of the dynamics that are present in the system is in order. A useful way of decomposing the dynamic motion planning problem was presented in [7]. The idea is to consider the problem as a set of constraints that need to be satisfied. These motion constraints can be treated as a perturbation to the manipulator that are compensated for using feedback. The feedback frequency that is required differs for each type of constraint and can vary significantly. As an example, exerting a constant force on an object in the environment can require feedback in the range of 1000 Hz, where changes in the global environment typically happen much slower and can be compensated for with feedback in the range of 1 Hz. Figure 2-7 demonstrates the relationship between several motion constraints and the required feedback frequency.

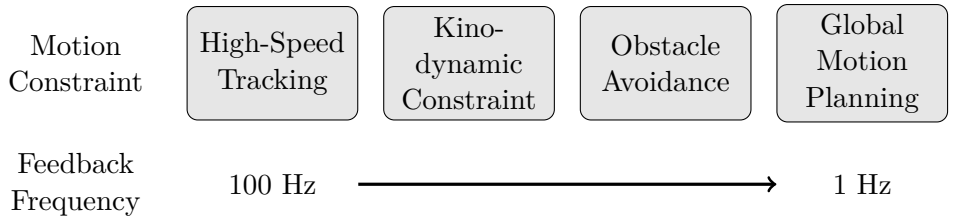


Figure 2-7: Motion constraints and required feedback frequencies.

The fastest two constraints are not influenced by the environment and only need feedback from the manipulator itself to be controlled. In the case of the UR5 manipulator, the high-speed tracking is done by the embedded joint controllers. The exact frequency of the control loop on the joints is unspecified, but as the highest rate that they are able to receive new setpoints is set to 125 Hz, it is expected to be well over 300 Hz. By decoupling the path and trajectory it can be ensured that the target trajectories satisfy the kinematic constraints.

The two rightmost constraints both require feedback from the environment in order to be controlled. These are the relevant motion constraints that are to be solved by the motion controller, as seen in Figure 2-8.

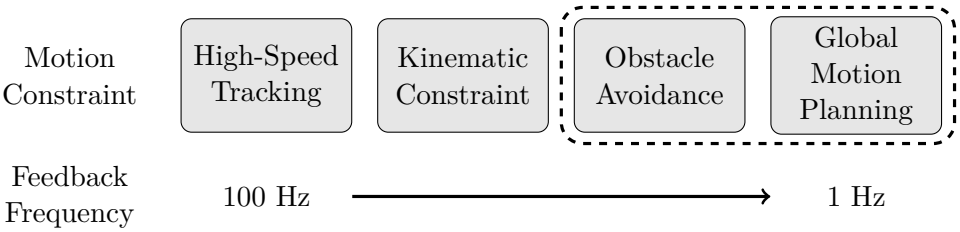


Figure 2-8: Motion constraints and feedback frequencies that require environment feedback and are to be solved in the motion controller.

Two main methods of executing motions on a manipulator can be identified from literature.

- Dynamic System Control
- Global Path Planning

Dynamic system control involves designing a dynamic system that directly uses the feedback to formulate an input to the system. This can be a PID controller, a Linear Quadratic Regulator, Model Predictive Controller or any other dynamic control law. The joint controllers in the UR5 use such a dynamic system to control the joint angles. An advantage of control with a dynamic system is speed. The feedback is handled immediately and very little computational time is needed to calculate the input signal. These systems are often used for learning based controller that are capable of performing very specific but highly dynamical tasks, like playing table tennis[8]. The main drawback is that they are prone to getting into local minima. They are also generally not able to find a path for a difficult problem.

A global path planner, although a lot slower, does not get stuck in local minima and optimal planners exist that optimize to a certain cost function. The main drawback is the time that is needed to find a path. Practical planners are currently limited to finding a path in the order of 1 Hz. An overview of the available control methods and their possible application and problems are shown in Figure 2-9.

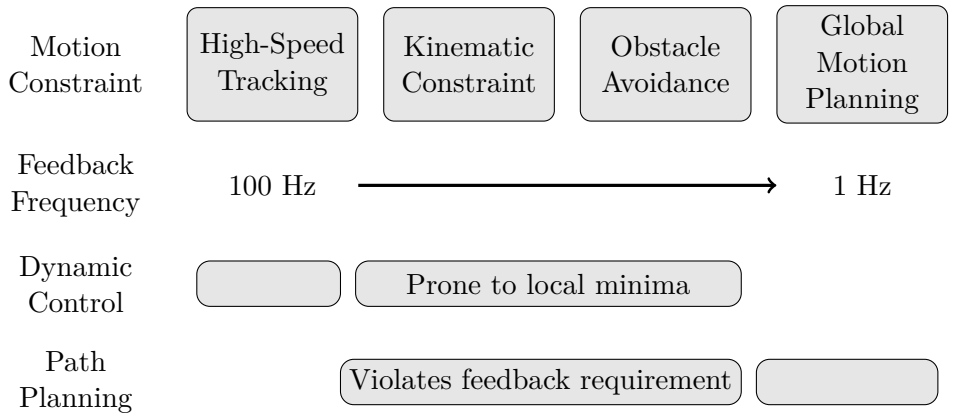


Figure 2-9: Motion constraints and required feedback frequencies.

Judging from these constraints, there are 3 main directions that can be explored to formulate a dynamic motion controller that will satisfy the specified requirements:

- Formulate a dynamic controller that does not get stuck in local minima and is able to find paths in difficult environments
- Increase the frequency of a path planner so that it can handle dynamic obstacle avoidance
- Combine a path planner with a dynamic control system in order to handle both constraints

A quick glance on the current state of research indicates that a globally stable dynamic controller that is not prone to getting stuck in local minima is currently not available.

Increasing a motion planner to work in real-time produces a system architecture as demonstrated in Figure 2-10.

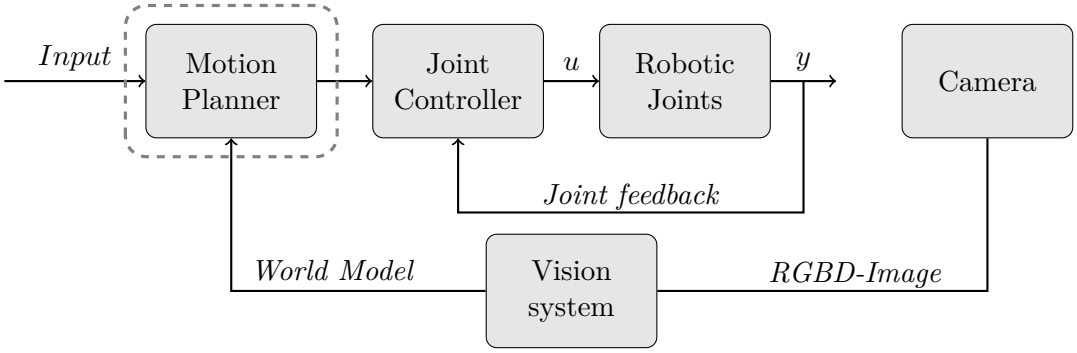


Figure 2-10: System topology for a real-time motion planning system. The dashed rectangle indicates the systems that are relevant to this research. In order to satisfy the motion constraints, the inner loop should run at several 100 Hz, and the outer loop at about 5 to 10 Hz.

Even though this approach would satisfy the feedback requirement, it would not mean it would be a 'perfect' solution. As planning the path still happens in a static scene, increasing the speed will not be able to mitigate all problems. For example, an obstacle crossing the intended path of a manipulator would cause the system to find a new path, even though the obstacle might be long gone before the manipulator reaches that part of the path. Several methods and heuristics have been developed in search of solving this problem, but it remains a challenge of which a single solution has not been found.

A second option is to combine a path planner with a dynamic control law. There are several ways to combine these approaches, one of which is demonstrated in Figure 2-11.

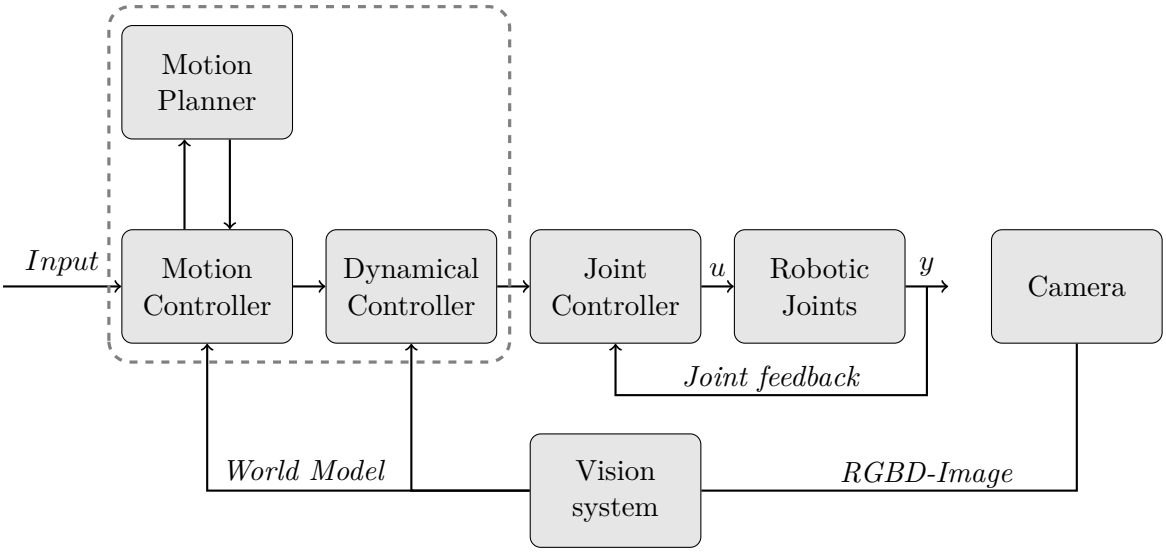


Figure 2-11: System topology for a real-time motion planning system that uses a motion planner and a dynamical system. The dashed rectangle indicates the systems that are relevant to this research. In order to satisfy the motion constraints, the inner loop should run at several 100 Hz, the Dynamical control loop at 10-100 Hz, and the Motion controller loop at about 1 Hz.

In such a system, a motion planner first plans a global motion that is then fed to a dynamical system. The dynamical system follows the motion while avoiding obstacles. Whenever the goal is reached or the system is stuck in a local minimum, the motion controller plans a new motion and resets the dynamical controller.

By now it is clear that a motion controller that satisfies the research goals will need to either use a very fast path planner, or a path planner combined with a dynamic controller. In the next part, current research will be examined to find which methods are appropriate in what conditions and in what capacity they can be employed to reach our requirements.

2-3 Pick and place systems

Given the practical objective of this research, it is important to consider a more practical viewpoint than found in a lot of research. With the bulk of research focusing on a small subproblem, it is sometimes difficult to determine the practical capabilities of a proposed approach. As the goal is to perform a pick and place task, a higher level overview of these motions will be described in this section.

In most systems, a picking motion is split into two parts, the *global* motion and the *grasp approach*. When attempting to perform a grasp, most systems first move to a pre-grasp pose using a global path planner planner. The pre-grasp pose is chosen such that it is easy to move towards the final grasp pose using a straight line approach in the workspace. This is called the grasp approach. Consider Figure 2-12 for a 2 dimensional example of a pick motion.

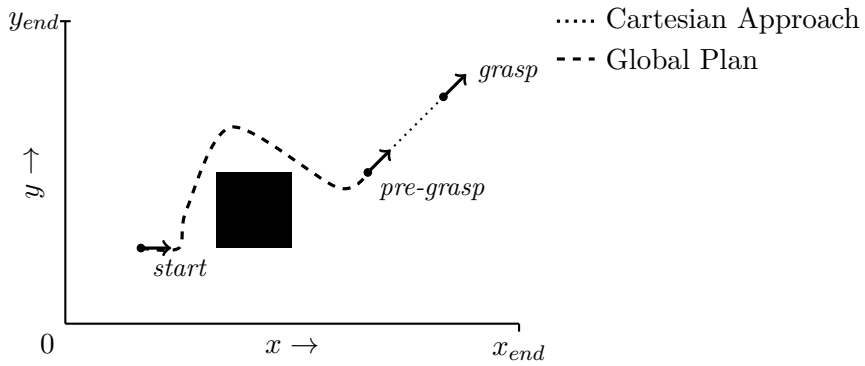


Figure 2-12: 2D pick and place example

The current picking pipeline at Technolution works in a similar fashion. For testing purposes, the target objects of the system are Jenga blocks measuring 15x25x75mm. For completeness, the perception system is also included:

- **Perception**
    1. **Observation** This is done using a Kinect 2 RGBD camera, which outputs a point-cloud: a large array of points that each have color and depth values.
    2. **Pointcloud Processing** By only allowing points that are inside the relevant workspace of the manipulator to remain in the pointcloud, processing is a lot

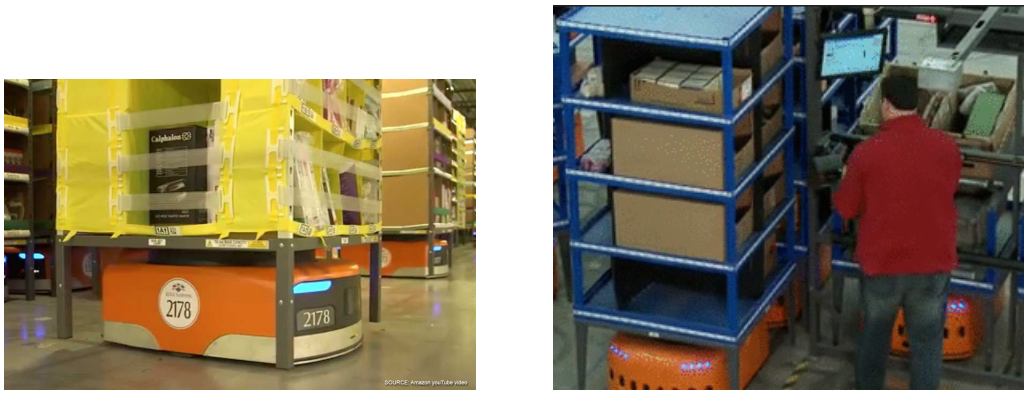
faster for later stages. In this stage, color filtering is also done as the system is only interested in objects with a certain color.

3. **Object Detection & Pose Estimation** By running a clustering algorithm, the perception system now figures out what points belong to a single block. By calculating a bounding box, a pose is estimated.
4. **Determine grasp pose** From the estimated location and pose of the target block, the pose that the manipulator needs to attain is calculated.

- **Manipulation**

1. **Plan & execute path to pre-grasp pose** Before anything else, the grasp pose, which is still in reference to the camera, is transformed to the robot frame. Then, the pre-grasp pose is calculated and a path is planned and executed.
2. **Perform grasp** A Cartesian approach is used to move to the actual grasp pose, and the gripper is closed.
3. **Retreat to post-grasp pose** Again, a Cartesian path is used to retreat back to the grasp pose.

To examine a number of current practical approaches, teams that participated in the Amazon Picking Challenge[15], or APC, in 2015 will be considered. In the APC, a manipulator is tasked with recognizing and grasping a number of specified objects from a shelf and placing them in the correct stowing unit in an unstructured environment. Amazon started the competition as a means to strengthen the ties between academics and industry and attempt to spur the advancement of warehouse automation. To this effect, Amazon started using Kiva robots to take on the task of locating and moving shelving units towards locations where human workers pick the right items. Two photographs depicting the current situation can be seen in Figure 2-13.



(a) Kiva Robot **(b)** Human worker

Figure 2-13: Current state at an Amazon warehouse

The goal of the APC is to also replace the human worker with a robotic manipulator. In 2015, the first APC was organized. Team RBO from TU Berlin won the competition by scoring 148 out of 190 points. Second place was awarded to the team from MIT with 88 points and Team Grizzly from Oakland University came in third with 35 points.

In [16], the winning team describes their system. Their picking primitives are specified using so called Hybrid automatons. A simplified depiction of their picking automaton is shown in Figure 2-14. Such a segmented approach allows them to use different planning and control

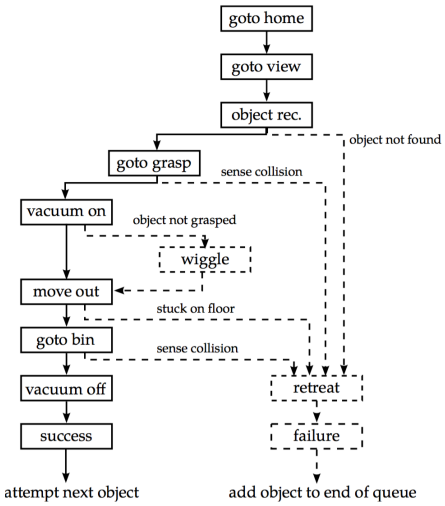


Figure 2-14: Simplified hybrid automaton for a picking primitive of team RBO entry to the APC 2015 [16]

methodologies for each step, instead of being tied to a single paradigm. For the motions to and from known positions, they used pre-programmed trajectories. Doing this negates the need for planning global paths, and can help in avoiding the sporadic paths that fast randomized planners can sometimes produce. For the grasping approach and other fine motions, they are able to use a combination of the mobile base and the manipulator, just as is more appropriate in each scenario. By doing so, they effectively decrease the degrees of freedom of the manipulator, but impose a set of constraints that are advantageous when manipulating in tight areas such as the inside of a bin.

Another approach to pick and place motions that was used in the APC is described in [17]. Their system, called *Dorapicker*, is described as *"An autonomous picking system or general objects"*. Consider Figure 2-15 for an overview of the architecture. This system is very similar to what is currently in place at Technolution. One difference is ICP, or *iterative closest point*, an algorithm that can be used to match two different pointclouds. If the pointcloud of a target object is known and available, you can use ICP to match it to the pointcloud from the camera, thus estimating the pose of the object. This does require having accurate object models beforehand, a common requirement for these systems.

To summarize the takeaways form the APC 2015, Amazon published a document entitled *"Lessons from the Amazon Picking Challenge"*[18]. With regards to the motion planning and execution, it is noted that a lot of teams felt increasing the reactivity of the motion execution in order to adjust for uncertainties in the perception would significantly improve their approach. Both the winner and runner up used feedback in their motion execution. One difference between the APC and the system that is being investigated in this research, is that the APC setup was well defined and designed to be easily accessible for human workers. This lead to a setup where the robot was not required to perform very *difficult* motions, thus

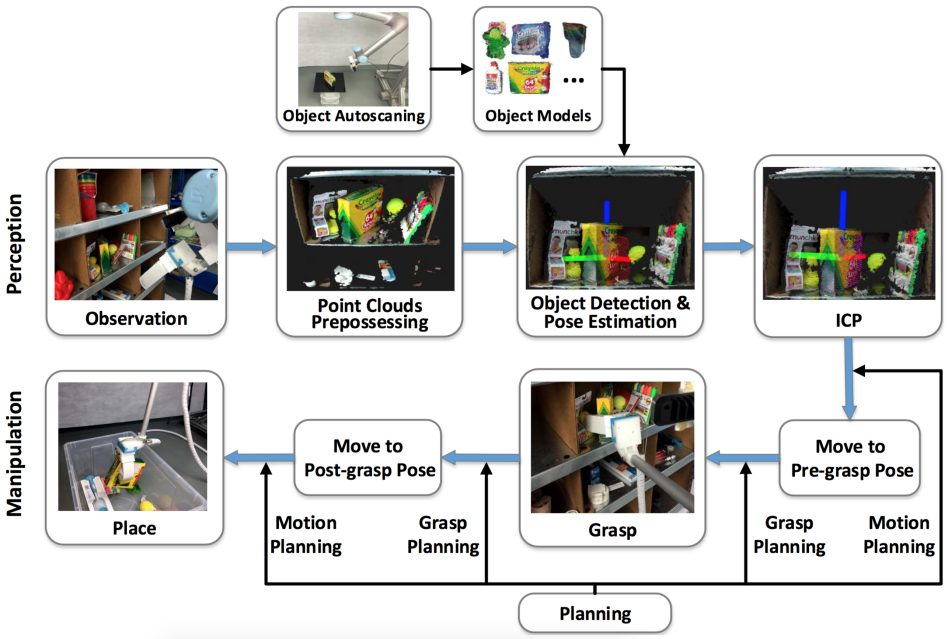


Figure 2-15: Architecture of the Dorapicker system [17]

enabling the user to move towards more reactive methods, as the chances of ending up in a local minimum are much smaller. This is the reason a large part of the teams refrained from using motion planning altogether. For a more general approach that is capable of dealing with a multitude of different environments, there is a good chance that configuration space planning algorithms cannot be omitted.

Chapter 3

Path planning

In Section 3-1, a number of basic path planning concepts will be introduced. Afterwards, the focus will be on practical path planning methods that are currently in use. Of these, many belong the the Probabilistic Roadmap class of methods, and will be discussed in Section 3-2. Subsequently, methods related to the Randomly exploring Random Tree, will be discussed in Chapter 3-3. Several other methods that have shown promising practical uses are discussed in Section 3-4. A qualitative comparison between these methods will be made in Section 3-5 and a few concluding remarks will be shared in Section 3-6.

3-1 Path planning basics

This section will introduced some of the basic concepts related to path planning. First the main methodologies will be introduced, after which a short introduction on practical collision checking will be given.

3-1-1 Planning methodologies

There are a number of different widely used methodologies in motion planning. Following [9], these can be divided in the two main philosophies:

- Sampling-based motion planning
- Combinatorial motion planning

As the name implies, sampling-based motion planning works by sampling the solution space, whereas combinatorial planning algorithms act on the solution space directly. In practice, most combinatorial methods are difficult to use for manipulators as they require an explicit representation of the state space. Therefore, combinatorial methods will only be briefly discussed.

An extensive review of sampling-based motion planners can be found in [6]. Sampling based methods can be further subdivided in the following two directions:

- Roadmap methods
- Incremental search methods

The next few paragraphs will offer a further explanation of some of the classes of methods and terms that have been introduced up until this point.

Roadmap methods Roadmap methods transform the motion planning problem into a undirected graph search. One common way to construct the roadmap is to utilize a preprocessing stage in which the configuration space (and the corresponding workspace states) are randomly sampled to construct a set of states throughout the workspace, acting as nodes in a graph. These states are the connected to their nearest neighbors and a collision detection algorithm is used to find out whether a free path exists. If it does, the length of the path is stored as an edge in the graph. Early efficient methods to solve undirected graph problems date back as far as 1959 and several heuristics have been developed that perform well under different circumstances. Planners based on roadmap methods will be discussed further in Section 3-2.

Incremental search methods Incremental search methods also sample the work space but do so in a different way. They start out with a graph that only has the initial state and successively adding a new vertex to expand the graph until the goal state is reached. For every new vertex a local planner finds a path that connects the new vertex to the existing graph and performs a collision detection. One of the more practical and widely used incremental search methods is the Rapidly-Exploring Random Tree. This method and its derivatives will be further discussed in Section 3-3.

Combinatorial methods Combinatorial approaches find paths through the continuous configuration space without discretizing the space. One example of such methods are the class of cell decompositions methods.

Cell decomposition methods divide the configuration space into convex cells within which the path from one point to another is trivial to generate. A connectivity graph is then calculated that maps the connection between the cells in order to reduce the problem to the well known graph search problem. Consider figure 3-1 for two examples of cell decomposition methods. In approximate cell decomposition a recursive method is used to continue subdividing cells that contain both C_{free} and C_{obs} until either *a)* all cells consist completely of either C_{free} or C_{obs}, or *b)* a specified minimum cell size is reached.

3-1-2 Collision detection

Collision detection is an important part of any planner. Most random sampling based planners are completely agnostic to the specific details of the type of problem they are asked to solve. In the sampling phase, a random state is sampled and checked for validity. In this

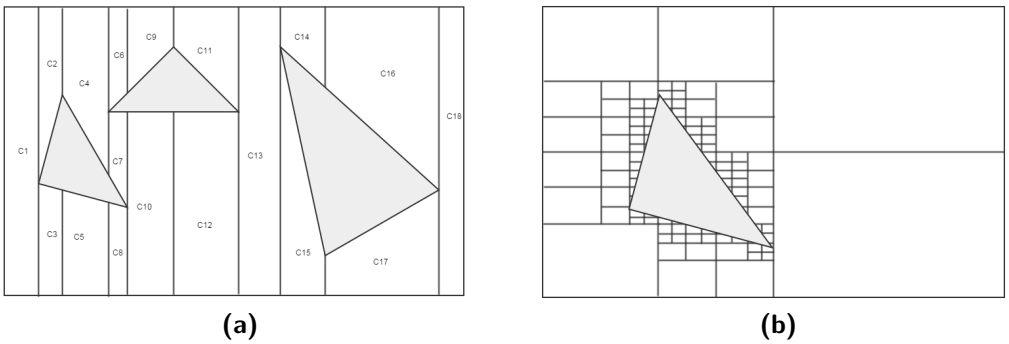


Figure 3-1: Two examples of cell decomposition methods. a) Vertical cell decomposition and b) Approximate cell decomposition.

particular scenario, that check consists of checking for both self-collisions and collisions with any obstacles. Moveit uses the Flexible Collision Library, or FCL[10] for collision checking.

Different methods invoke the collision detector at different times during the execution. As a lot of the collision detection is done on-line, it significantly impacts the performance of the planning method. Figures regarding computational cost of collision checking found in literature range from 94% [11] to 10-15% [12], depending on what kind of collision detection and planner is used. It is very difficult to make an assessment about the impact that the collision detection will have on real-time performance in advance.

There are a few different approaches to speeding up the collision checking, one of which is the so called *lazy* approach. Lazy planners postpone all the collision checking until it is absolutely necessary. Lazy collision checking has been researched in combination with a variety of different planners[13][14]. Some planners also use the information from the collision checking to guide the search. For example, using the distance to the closest obstacle to guide the search into the open space. As the information returned by the collision checker varies between methods this is not always possible.

3-2 Probabilistic roadmap method

The Probabilistic Roadmap Method was introduced in 1996 in [19]. PRM divides the planning into two distinct phases; a *preprocessing* phase and an *on-line* phase. In the preprocessing stage a roadmap is constructed by sampling configurations from C_{free} and connecting them to other configurations that are close by some arbitrarily chosen distance norm. Doing so effectively transforms the problem into an undirected graph which is much easier to solve. In the on-line phase the nodes corresponding to the initial and goal configurations are added to the constructed roadmap after which a shortest path algorithm determines the optimal path. Figure 3-2 shows the basic concepts of PRM.

In its most basic form PRM uses a random sampling to find the configurations in C_{free}. Random sampling has been shown to have efficient and fast solutions for a large variety of problems but other sampling methods such as sampling around obstacle boundaries, sampling in a grid and pseudo random sampling can be found in literature. PRM is said to be

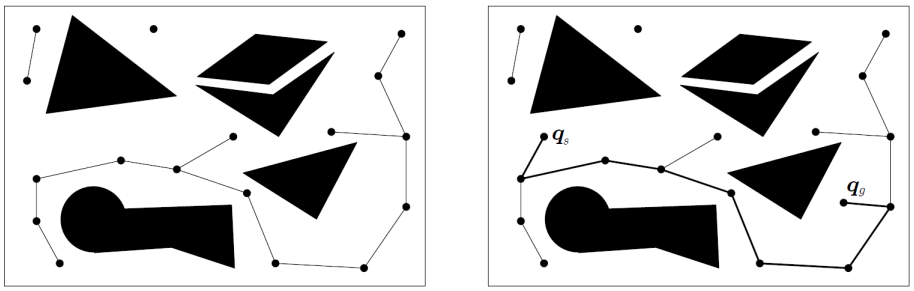


Figure 3-2: Left: A roadmap in a 2D configuration space. Right: A solved query in that same configuration space. [20].

probabilistically complete [21]. This means that as the sampling density approaches infinity, the probability that the planner finds a feasible path (given that it exists) goes to 1.

The on-line phase of the PRM planner consists of connecting the initial and goal configurations to the roadmap, and finding the shortest path. Finding the shortest path in an undirected graph can be done in several ways, of which Dijkstra's algorithm[22] and A*[23] are the most well known. Several different graph search method exist that are capable of finding a path with different constraints like a moving target state, or a changing roadmap.

A note should be made about path search algorithms that use time information, like planning towards a moving target or in the presence of dynamic obstacles. As the path is defined in the configuration space, the notion of *when* a certain state will be reached becomes somewhat of an abstract concept. If the speed between the waypoints is assumed to be constant, joint constraints might be violated and you lose the possibility to do a more suitable time parameterization.

An important subclass of sampling-based motion planners were introduced in [24]. These motion planners are a family of *optimal* planners, which are guaranteed to asymptotically converge to an optimal solution based on a specified cost-function.. The PRM variant of these planners is denoted PRM*. Due to their favorable paths and high theoretical merit, optimal planners play a large part in the current research towards new path planning approaches. Whether optimal planners are useful in a practical scenario remains unclear from literature.

3-2-1 Dynamic roadmap planning

In [25], a framework is described that adapts PRM to real-time control in dynamically varying environments. The preprocessing stage consists of calculating a roadmap to the obstacle free environment and creating a mapping from workspace cells to the nodes of the roadmap. In the query stage, nodes corresponding to cells that are being occupied by obstacles are removed from the roadmap after which a standard tree search finds the shortest path. In [26], this framework, called DRM, is improved and applied to a robot arm to solve the dynamic planning problem in under 100ms. note that the approach does not make any prediction on the movement of obstacles, but works sufficiently fast to deal with simple dynamic obstacles. In [27], this work was again extended into Parallel DRM (PDRM). The planner was optimized for parallel calculations on a GPU and managed to get the planning time down to 25 to 65ms.

3-2-2 Anytime PRM

Another interesting PRM based dynamic planner was introduced in [28]. Again, the PRM roadmap is constructed to take into account the static constraints on the robot's motion. In addition, it includes cost information associated with desirability of all the configurations. The trajectory is then calculated according to the cost function shown in Figure 3-3.

$$C(path) = \omega_t * t_{path} + \omega_c * c_{path}$$

Figure 3-3: Cost function for the Anytime Planning and Replanning method, ω_t and ω_c are weights on the time and path that satisfy $\omega_t + \omega_c = 1$. t_{path} is the time in seconds and c_{path} denotes the cost of the path.

In order to deal with dynamic obstacles, the position of moving obstacles are extrapolated and are avoided throughout the path. In the on-line phase the path planner works in an anytime fashion which is to say it first generates a suboptimal path as fast as possible and then uses the remaining time to improve on that. This is done using the Anytime D* heuristic [29]. A graphic depiction of this path is shown in Figure 3-4.

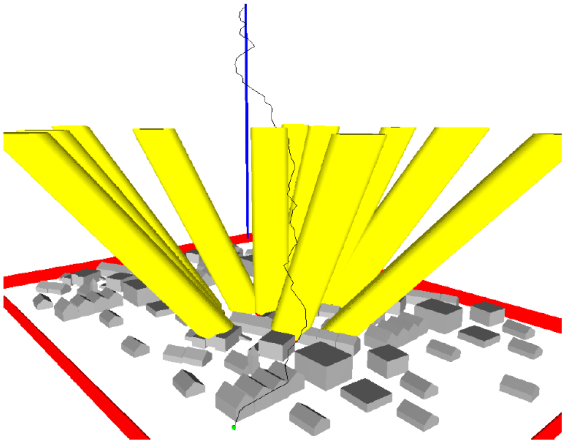


Figure 3-4: The path generated by the Anytime PRM algorithm introduced in [28]. The initial state is depicted in green and the target state in blue, moving up as time goes on. The path is black and the extrapolated obstacles are shown in yellow.

3-3 Rapidly exploring random trees

Rapidly exploring random trees (RRT) are another very commonly used algorithm in path planning. The concept was introduced in 1998 in [30]. A basic description of the RRT algorithm reads:

- The search is initialized in q_{init}
- A sample procedure is used to select a new random node q_{rand}

- If the node is in C_{obs} it is discarded
- A search finds the tree node that is nearest to q_{rand}, q_{near}
- A local planner is used to connect q_{rand} to q_{near}, if this is not possible, a new point q_{new} may be added and q_{rand} is discarded
- The new path is checked for a collision, if none is found q_{rand} and the path are added to the tree
- The search terminates whenever the goal is reached, or some other termination limit like time or iterations has been exceeded

The procedure is shown in Figure 3-5

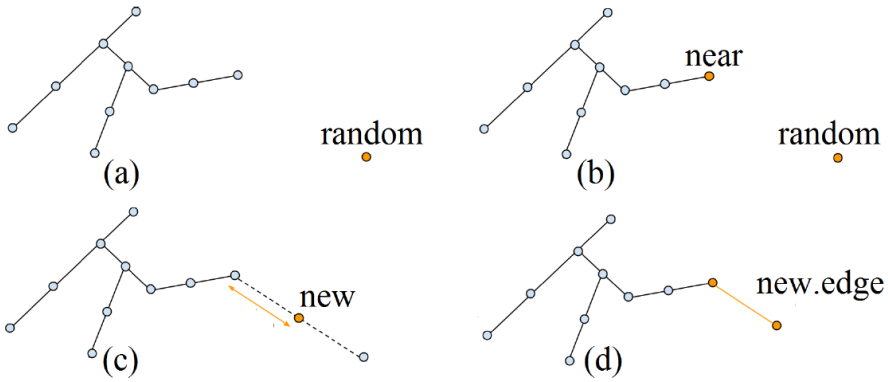


Figure 3-5: RRT procedure. (a) adding a new node. (b) finding the nearest tree node (c) a new point q_{new} is added as q_{rand} is unreachable (d) q_{new} and the new edge are added to the tree [6].

An example of an expanding RRT in free space can be seen in Figure 3-6

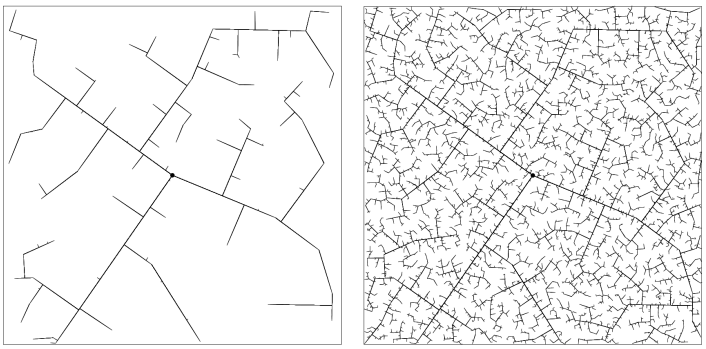


Figure 3-6: Example of an expanding RRT in free space[9].

Because of the random sampling nature of RRT, it is biased towards large open regions in the available space[6], thanks to this property RRTs are very capable of dealing with obstacles.

One important note to make is that RRT is said to be a *single-query* sampling-based planner as it only finds a single path from one point to another. In contrast, PRM is a *multi-query* planner as it can simultaneously find the shortest paths between many nodes in the roadmap. As with PRM, RRT is also shown to be probabilistically complete [31].

Just as with PRM, an asymptotically optimal RRT was presented as RRT* in [24]. An interesting variant was described in [32], where RRT* was adapted to work in an anytime manner.

3-3-1 RRT-Connect

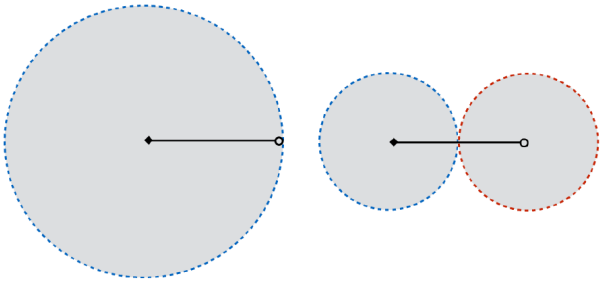


Figure 3-7: On the left the coverage area of an unidirectional search is shown, and on the right the coverage area of a bidirectional search [6].

One other heuristic that is used to speed up RRT is to perform a bidirectional search[33]. In addition to searching from start to goal, a second search is initiated from finish to start. When both trees connect the two paths are combined to construct the full path. Figure 3-7 shows why this is an effective approach to reduce your search space.

Unlike RRT*, RRT-Connect is a non-optimal planner. There are several mentions in literature of RRT-Connect being a particularly fast planner. However, as RRT-Connect is a non-optimal planner, the cost of the path is often found to be higher than other planners [34][35][36].

3-3-2 Transition-based RRT

A variation on RRT is called Transition-based RRT, or TRRT [37]. TRRT extends RRT by testing new potential states using stochastic optimization in order to guide the search towards solutions that are favorable according to a predefined costmap. As with the optimal planners, the formulation of the cost function takes a central role in the performance of the planning algorithm. Being very close to standard RRT, TRRT is expected to offer similar performance, with the added benefit of the costmap potentially being used to guide the search towards favorable solutions.

As with RRT, a bidirectional variant, called BiTRRT was proposed that builds an expanding tree from both the start as the goal state.

3-3-3 Informed methods

Informed-RRT*

A recently introduced method is called informed-RRT*[38]. In informed-RRT, the search is guided towards the solution by a bounded search area. A first solution is found, after which a hypersphere surrounding the found solution bounds the search space. A new solution is sought within that search space and the search continues. Figure 3-8 shows how the search area is bounded.

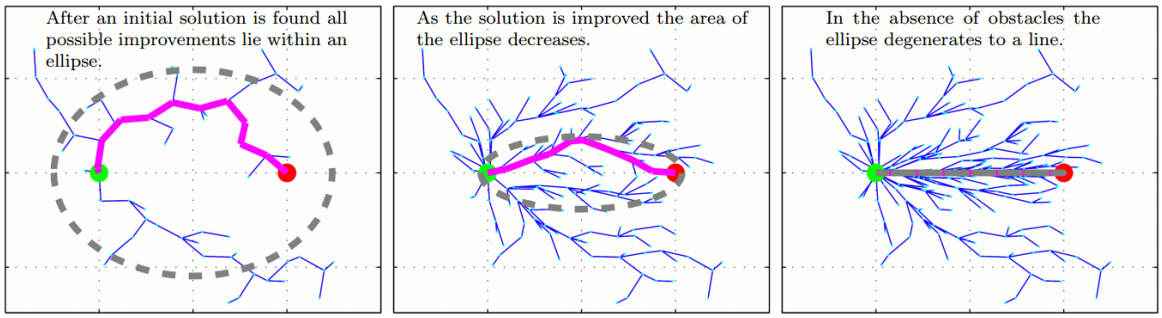


Figure 3-8: The procedure of bounding the search area in informed-RRT*. [38].

For planning in the workspace, it is easy to see how a hypersphere provides a intuitive heuristic for guiding the search, but for planning in the configuration space it is less obvious how performance will be affected. Further testing needs to be conducted in order to draw any meaningful conclusions.

Fast Marching Trees*

An algorithm that was specifically aimed at solving complex motion queries in a high dimensional space are Fast Marching Trees, FMT* [34]. FMT* is specifically aimed at solving complex motion planning problems in high dimensional spaces. The algorithm builds an outward expanding tree on a preset number of random samples in the configuration space, combining some elements of single-query planners like RRT and multi-query planners like PRM.

Figure 3-9 shows how FMT* expands it's tree throughout the search space. FMT* has been tested on high dimensional spaces in random environments and its performance was superior to other methods, especially in cases were collision checking was expensive. One of the ways FMT* improves performance is by employing the previously mentioned lazy collision checking.

Batch Informed Trees*

Another very recent planning method is BIT* [36]. BIT* has some similarities with Informed-RRT* and FMT*. However, BIT* uses an incremental search and increases the sample density

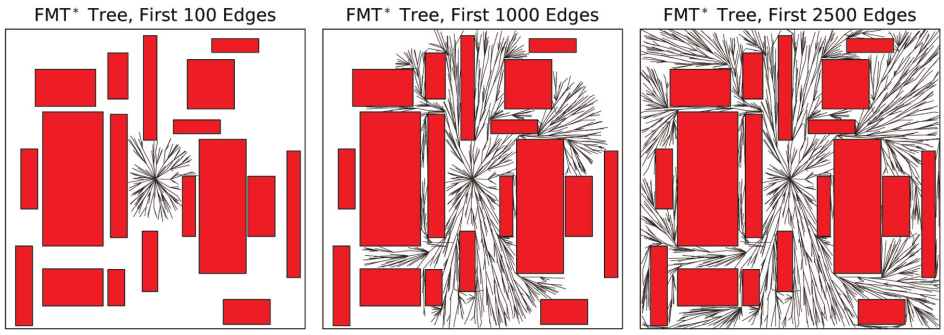


Figure 3-9: FMT* expanding over a space with 2500 samples[34].

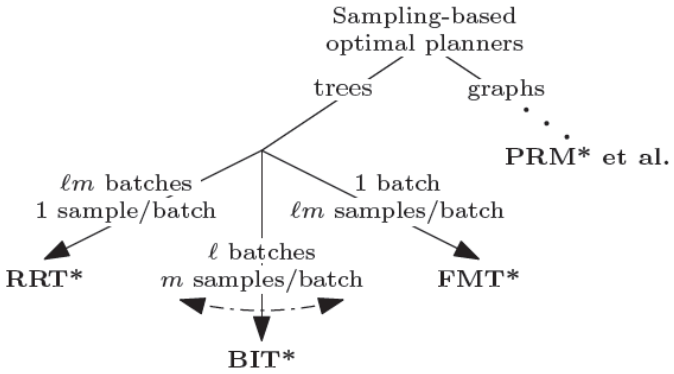


Figure 3-10: The relationship between RRT*, BIT*, PRM* and BIT* in terms of sampling methods [36]. RRT* samples a single point for each batch, FMT* samples all points in advance for a single batch, and BIT* adds new samples for each batch.

as soon as a path has been found. In addition, BIT* uses the sampling heuristic introduced in Informed-RRT* to guide the search towards an area where better solutions are likely. Figure 3-10 shows a simplified taxonomy of different sampling based path planners.

In Figure 3-11, An example is shown of the expansion of the tree expansion of a RRT*, FMT*, informed-RRT* and BIT*

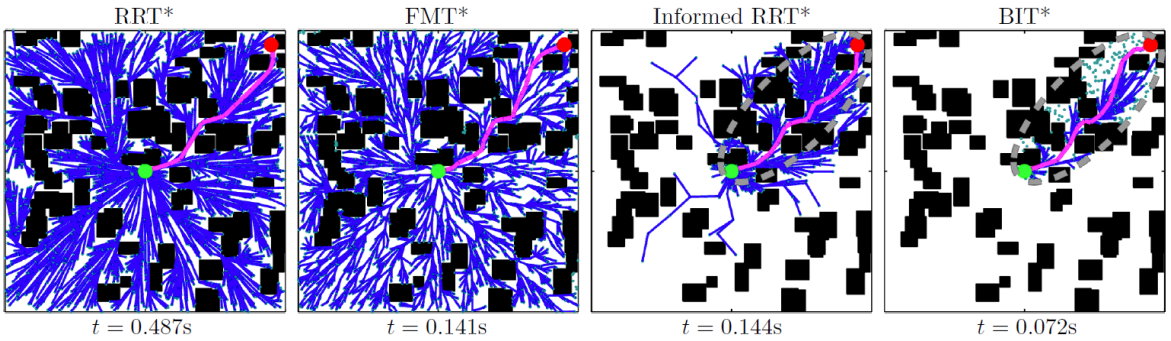


Figure 3-11: Comparison of the tree expansion of RRT*, FMT*, Informed-RRT* and BIT*[36].

3-3-4 KPIECE methods

Kinodynamic motion planning by interior-exterior cell exploration, or KPIECE methods were introduced in [39] and use a grid-based discretization to estimate the coverage of the search space. While they are specially designed for systems with complex dynamics, they are commonly applied and are known to be a find a solution within limited runtime.

The core components of KPIECE are:

1. Using projections from the state space to a lower dimensional space that can be discretized to guide the search
2. Keeping track of the boundary of the explored space efficiently so exploration can be biased towards unexplored regions
3. Combining collected exploration information to better select new regions to explore
4. On-line evaluation of the exploration progress

Similar to RRT, KPIECE grows a tree of motions. However, as the basis of KPIECE lies in forward simulation of a physical system, the states are added in a somewhat different manner. For each new motion, KPIECE selects a motion primitive, which is added in small steps to reach a new state.

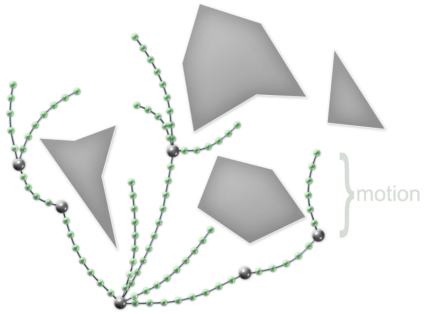


Figure 3-12: Tree of motions as grown by KPIECE, the larger vertices are states at the start of a motion. The smaller vertices depict the intermediate steps, which are not stored by KPIECE [40]

Furthermore, KPIECE maps the space by using a projection and a discretization to evaluate the interior and exterior cells, in order to guide the search towards open regions. An example of such a discretization is seen in Figure 3-13

Aside from the standard KPIECE method, a bidirectional variant (BKPIECE) and a lazy bidirectional variant (LBKPIECE) were shown to have promising performance.

3-4 Other methods

This section will discuss methods that cannot be classified in either of the previous two classes.

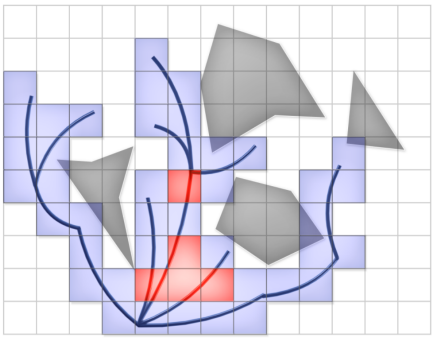


Figure 3-13: Representation of a tree of motions. Interior cells are depicted in red and exterior cells in blue [40]

3-4-1 Experience-based planning

Experience-based planning is an area of research that displays promising performance for high dimensional manipulators. The main idea of these planners is to use previously planned paths to guide the search towards faster solutions. It is especially suited to robots with a large number of invariant constraints such as joint limits and self collisions.

After several earlier approaches at a planning framework that uses experience, The 'Lightning' framework was introduced in [41]. The core concept is to allow the planner to learn from experience by storing successful paths in a database. A diagram of the framework is shown in Figure 3-14.

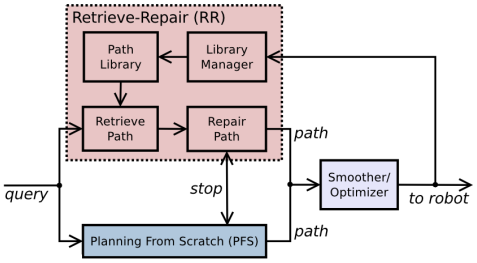


Figure 3-14: The Lightning framework[41].

As can be seen in the diagram, the planner actually consists of 2 main modules:

- The Retrieve-Repair module (RR)
- The Planning From Scratch module (PFS)

Whenever the planner is posed with a query, both modules simultaneously start computing a path. The first path to be found is executed and the other module is stopped. After a path is generated the library manager checks if the path is worth to be added to the library. This allows the framework to 'learn' certain facts about it's environment. Several scenarios were tested with a variety of different library sizes and it was demonstrated that with a sufficiently large library, over 95% of the queries could be solved by the RR module. As the RR module is faster with a factor of 3-10 in comparison to the PFS module, this is a large improvement.

The Lightning framework was extended into the Thunder framework in [42]. The key difference lies in in the way the library is constructed and stored. Experiences are stored in a sparse roadmap spanner. Sparse roadmap spanners attempt to create a sparse representation of the search space in order to decrease the search time on dense graphs. In order for dense graph methods, like PRM, to achieve optimality, the size of the graphs can get very large for higher dimensional problems. This is improved in the sparse roadmap spanners introduced in [43].

The compact representation does come at a cost, even though probabilistic completeness is preserved, the SPARS spanner only has asymptotic *near*-optimality. However, for the experiments ran in the paper, path-lengths were very close to paths generated by PRM* and were much better than the theoretical bounds. The difference between storing 2000 experiences in the Lightning and in the SPARS library is shown in Figure 3-15.

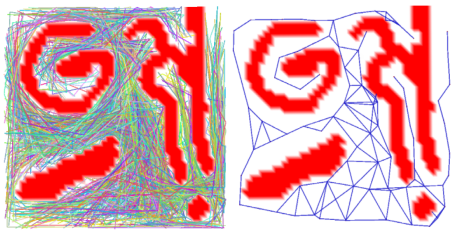


Figure 3-15: The difference between 2000 stored experiences in (a) Lightning (PRM) and (b) Thunder (SPARS)[42].

In addition to the compact representation, lazy collision checking is used. Once a collision is found the invalidated paths are removed from the graph and the search restarts. Thunder showed impressive performance when applied to a humanoid robot with 30 DoF. See Figure 3-16 for results of the planning query.

For a problem in $\mathbb{R}^{30}$, a mean planning time of 0.86 seconds for the Thunder framework is a very promising result. However, the training stage takes a significant amount of time to achieve this performance and it is unclear how the roadmap generalizes to a changing environment. For Figure 3-16, it takes the system about 5000 queries to converge close to the final planning time.

The reason these frameworks are said to work especially well for manipulators with a large number of invariant constraints is that by inserting paths into the library, you are effectively creating a explicit representation of the free space of the manipulator. For a humanoid, a large number of the states that a random sampling motion planner samples will be invalidated by these invariant constraints, effectively increasing planning time. For a robot arm like the UR5, it might very well be that the performance gain is not as great, as the invariant constraints are not as big of a problem. In addition, in the presence of variable constraints like dynamic obstacles, the advantage of Thunder and Lightning is expected to be much smaller.

3-4-2 Optimization based planners

A very different method was presented as CHOMP in [44]. CHOMP, Covariant Hamiltonian Optimization for Motion Planning, is a motion planner that takes a naive initial guess and

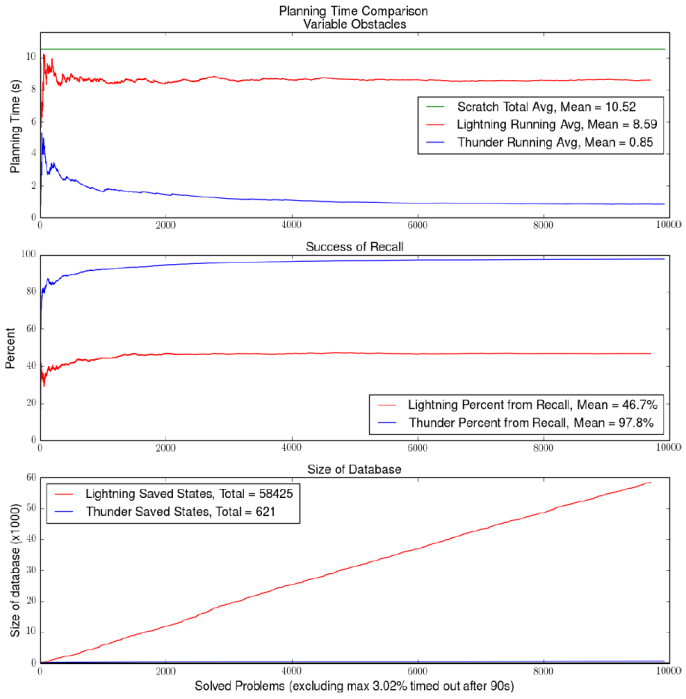


Figure 3-16: Results of the planning query comparing Lightning and Thunder. Success of recall refers to the cases where the solution was produced by the RR module[42].

transforms it into a suitable trajectory for execution on a manipulator without invoking a separate motion planner. In contrast to other optimization based techniques, the initial guess does not need to be collision free. CHOMP assigns a cost function based on the smoothness of the path and the avoidance of obstacles. Using gradient descent methods, CHOMP then attempts to optimize the cost function in order to find an optimal trajectory. CHOMP uses a potential function, much like those discussed in Section 4-1-1 to avoid collision with obstacles.

Workspace gradient information is used to inform the search and 'push' the trajectory away from obstacles. CHOMP is compared to several planners and can be seen to offer improved performance over RRT. A related algorithm is ITOMP, Incremental Trajectory Optimization for real-time replanning in dynamic environments[45]. As the title implies, this approach has support for dynamic obstacles, but the details about performance and implementation are unclear.

A third approach from the same class can be found in [46] and is called STOMP, *Stochastic Trajectory Optimization for Motion Planning*. STOMP creates a number of noisy trajectories around the current trajectory and then produces produces a lower cost trajectory based on the cost information obtained from the noisy trajectories. This process is repeated until the trajectory cost is sufficiently low. Due to the stochastic nature of STOMP, the authors claim that STOMP is less likely to get stuck in local minima. Consider Table 3-1 for the experimental results of STOMP and CHOMP on a simulated shelf environment.

In [47], an optimization method based on Sequential Convex Optimization named TrajOpt was introduced. The planning problem is formulated as a quadratic programming problem with convex constraints. By considering the convex hull of all rigid bodies in the environment,

	STOMP	CHOMP
Successful plans	210 / 210	149 / 210
Avg. Time (s)	0.88 ± 0.40	0.71 ± 0.25
Avg. iterations	52.1 ± 26.6	167.1 ± 113.8

Table 3-1: STOMP & CHOMP results on 210 arm planning problems for a 7 DoF PR2 in a simulated shelf environment [46]

	TrajOpt	TrajOpt-Multi	RRT-Connect	CHOMP	CHOMP-Multi
Successful	0.818	0.955	0.854	0.652	0.833
Time (s)	0.191	0.3	0.615	4.91	9.27
Length	1.16	1.15	1.56	2.04	1.97

Table 3-2: TrajOpt & CHOMP results on 198 arm planning problems for a 7 DoF PR2 [47]

TrajOpt uses efficient algorithms to calculate the convex-convex signed distance between the robot links and any obstacles. The signed distance is the defined as the vector that either object has to travel in order to get in or out of collision as fast as possible. Consider Figure 3-17 for an illustration of the signed distance. When a robot link is found to be in or close

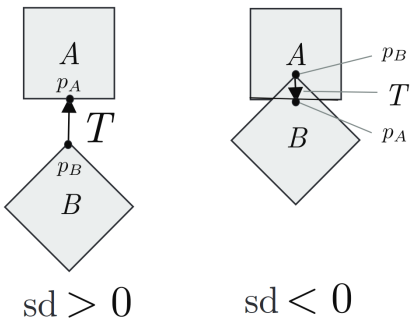


Figure 3-17: Signed distance illustration. The signed distance is positive when object A and B are not in collision. T denotes the sd vector and p_A and p_B denote the points of contact[47]

to collision, the robotic manipulator Jacobian is used to calculate a linear approximation of the signed distance function and is converted into a cost for the convex optimization. This cost is set so that it decreases as the robot link moves away from the collision. In doing so, TrajOpt effectively creates a local approximation of the collision scene, allowing a quadratic programming solver to make improvements until the original problem until the approximation breaks down, after which a new iteration is started. By repeatedly solving these subproblems, the approach iteratively makes progress on the original problem until an optimum has been found. In a benchmark using the 7 DoF PR2 Robot manipulator, TrajOpt performed better than CHOMP and RRT-Connect. The results of this benchmark can be seen in Table 3-2.

Optimization based planners have a number of advantages over random sampling based planning. As they do not rely on random sampling, they are not as prone to producing very strange paths. Also, the cost function allows the the approaches to converge to smoother

paths than random sampling planners. Unfortunately, neither of these optimization based planners have available and stable open source implementations that are compatible with ROS, the preferred platform for this research, which makes it very difficult to get an good estimate of their performance. Work is being done on integrating STOMP, but initial testing showed that the performance was not yet good enough to be a serious contender.

3-5 Comparison of path planners

In order to make a meaningful comparison, the criteria that the planners are judged on will be discussed.

- **Planning time** One of the main limitations of the planners is planning time. In order to execute a task within reasonable time a time constant is needed that not all methods can attain. If a planner is fast enough, a continuously queried path planner can be used to satisfy all requirements.

  As for most methods, a reliable benchmark is not available, the planning time performance has to be estimated. Several papers do have benchmarks in different configuration spaces that hold some insight but nothing conclusive can be said.

- **Optimality** As per the requirements, the ability to find a short path is coupled to the efficiency and predictability of the control method.

- **Completeness** The planners need to be able to find a path in any given situation. Depending on whether the planner is later coupled with a reactive controller, this is a very important property.

- **Practical Feasibility** For a number of the methods discussed, not implementation was made available. Implementing a method from scratch requires a significant engineering effort and is considered to be outside the scope of this research. For other methods, implementations are available, but they are not directly compatible with the current configuration.

- **Anytime planning** An Anytime planner facilitates a number of different system topologies that can offer benefits over others.

- **Learning** Whether the planner needs a learning phase. This can be either be in a preprocessing stage, or on-line

Table 3-3 lists the qualitative differences between the available planning approaches.

Judging the performance difference between the methods, a number of approaches stand out to be subjected to further testing, but a lot of the methods require too much effort to get working.

Method	Time	Optimal	Practical	Anytime	Learning
DRM	++	-	-	-	Off-line
Anytime-PRM	+	-	-	+	No
PRM*	-	+	++	-	No
RRT	+	-	++	-	No
RRT*	-	+	++	-	No
RRT-Connect	++	-	++	-	No
TRRT	+	-	++	-	No
BiTRRT	++	-	++	-	No
Informed-RRT*	+	+	+	+	No
FMT*	+	+	+	+	No
BIT*	++	+	+	+	No
AMP	-	+	-	-	Off-line
KPIECE	+	-	++	-	No
BKPIECE	++	-	++	-	No
Lightning	+	-+	+	-	On-line
Thunder	++	-+	+	-	On-line
E-Graphs	+	+	-	-	On-line
CHOMP	-	-+	-+	-	No
ITOMP	-	-+	-	-	No
STOMP	-	-+	+	-	No
TrajOpt	++	+	-+	-	No

Table 3-3: Comparison of different planners

3-6 Conclusion

Judging the performance of the planners described in the previous section, a selection of planners can be made that will be investigated further. A global path planner will likely be an integral part of any reactive motion system, so a thorough benchmark of these potential planners is warranted. The following planners are deemed to have to most potential:

- **RRT-Connect** Despite the occasional very long path, RRT-Connect seems to be able to find relatively decent paths very fast judging from several benchmarks found in literature.
- **BiTRRT** Closely related to RRT-Connect, BiTRRT is expected to be able to generate paths very quickly with potentially being able to generate shorter paths.
- **BIT*** shows the most promise from the group of informed planners and is optimal so can be used as a path-length benchmark. Fast path planning and seemingly good generalization properties make BIT* an interesting method.
- **KPIECE** The information on KPIECE and related methods is scarce, but it's performance is supposedly in the range of RRT-Connect. Further testing is required to determine which of the KPIECE methods carries the most potential.

- **RRT** Although slower than RRT-Connect, RRT is known to be a reliable planner that manages to find paths within reasonable planning times.
- **Thunder** is the most promising candidate from the experience based planners.
- **TrajOpt** appears to be the fastest option from the optimization based planners. An open source implementations is available on the web, but it require a substantial amount of work to get it running.

Out of these, RRT, RRT-Connect, BiTRRT and the KPIECE methods are readily available to be tested. BIT* and Thunder is available in a new release of the OMPL library, but will need to be integrated with MoveIt! in order to be tested. As for TrajOpt, an open-source implementation based on OpenRave exists, but it is unclear how much effort will be required in order to change that dependency to MoveIt!. Communication with author of TrajOpt indicated that this was indeed possible, but would require a significant effort.

As far as the optimal planners go, depending on the results of the first benchmark, it will be interesting to see how the following planners are able to utilize an extend planning time:

- **RRT*** One of the first optimal planners and known to offer good results
- **BIT*** Hopefully, the informed sampling of BIT* will enable the planner to quickly converge to an improved solution

Part 2 of this thesis will continue with the testing and benchmarking of these planning methods to find out which is most suitable for this application.

Chapter 4

Dynamic motion execution

This chapter will discuss the relevant theory related to dynamic motion execution. Section 4 will describe several dynamic control methods and analyze their respective performance. Next, Section 4-2 will further discuss the need for time-parameterization of a geometric path, and methods that can be used to do so. Methods that rely on quickly planning a new path in the presence of obstacles are discussed in Section 4-3. Finally, Section 4-4 will discuss how these methods can be applied to the research goals.

4-1 Dynamic control methods

In this chapter, methods that use a global control law will be discussed. As explained in Section 2, these methods are suitable for local control, but are prone to getting stuck in minima and are generally not capable of dealing with a changing global environment. In Section 4-1-1, Artificial potential fields will be discussed. Section 4-1-2 will present Visual Servoing methods. Finally, Elastic Band methods will be discussed in Section 4-1-3.

4-1-1 Artificial potential fields

Artificial potential fields are another class of methods that have interesting properties. The idea is to consider the state as a point in the planning space that moves towards the goal state by ways of a potential field. By assigning a low potential to the goal state and a high potential to the obstacles, the agent will be pushed away from the obstacles towards the goal configuration. The potential fields thus results in an artificial force that moves the agent through the configuration space towards the goal. A graphical explanation can be found in Figure 4-1.

One of the distinct advantages of the artificial potential field is that it can be applied on-line As the obstacles can be modeled to by dynamically varying, artificial potential method are easily adapted to being able to avoid obstacles and act in a real-time manner. However,

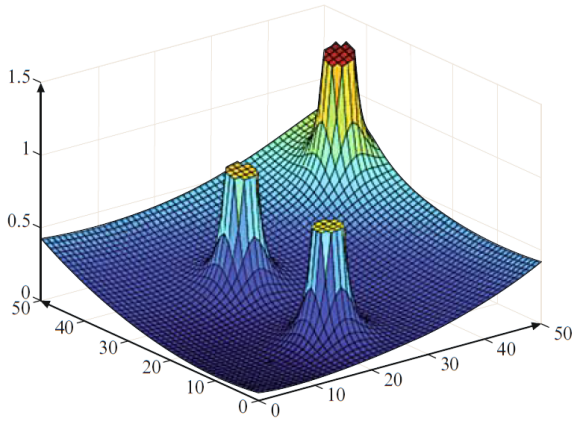


Figure 4-1: Graphical depiction of an artificial potential field with a low potential at the goal state and high potential at obstacles.

because of the nature of the potential fields they are prone to getting stuck in local minima. In addition, applying potential fields to a robotic manipulator, whose obstacles are defined in the workspace but that is controlled the joint space, is not straight forward.

4-1-2 Visual servoing

The term visual servoing is used to describe a technique that uses visual information as direct feedback for the control of a robot. There are two main directions within visual servoing:

- Position Based Visual Servoing (PBVS)
- Image Based Visual Servoing (IBVS)

The difference is that in IBVS, the 2D output of the camera is used directly as an input to the control law, whereas in PBVS a world representation is estimated from visual data before being used by the controller. The system as it is currently implemented at Technolution takes a PBVS approach, where the image processing pipeline is largely abstracted from the controller. Essentially, every system that uses sensors to maintain a virtual representation of the world around it can be classified as a PBVS system.

A characteristic that is often found in IBVS research is that there is a very direct feedback path from the vision system to the control law. This allows for very fast responses and has been implemented for a variety of manipulation tasks that easily outperform a human agent. In contrast, most PBVS systems maintain a much less direct connection. As an example, in the system that Technolution was using, the path planner sits in between the vision information and the control law, effectively inserting a significant delay into the feedback loop. In some works, only IBVS is seen as visual servoing.

An example of PBVS using a Kinect camera and a 7 DoF manipulator is presented in [48]. The objective of the controller is to follow a moving object. A Kalman filter is used to get a better estimation of the position of the target. See Figure 4-2 for the architecture. This is an example of PBVS where the feedback loop is quite fast.

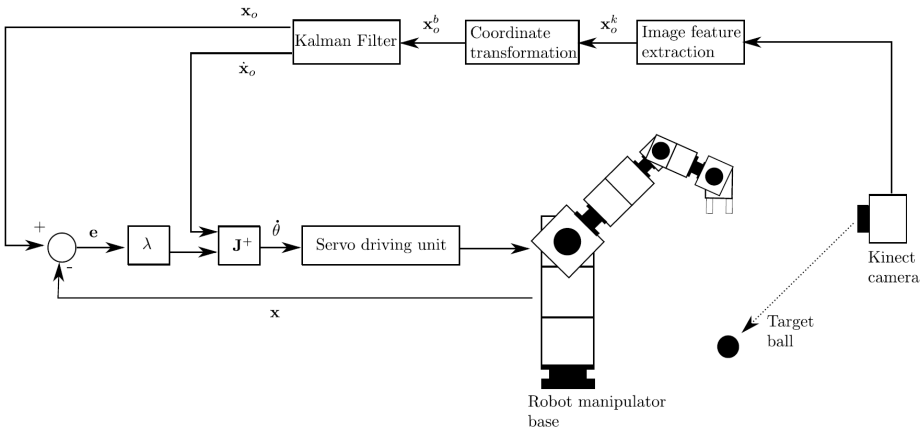


Figure 4-2: The architecture of the PBVS system from [48].

Because of the simple control law, the manipulator is not able to deal with unstructured environments or dynamic obstacles. There have been several attempts around these constraints. In [49], a Model Predictive Control based system was proposed that included constraints in the objective function to achieve obstacle avoidance. The MPC architecture is shown in Figure 4-3 Experiments were conducted on some simple problems which the manipulator could

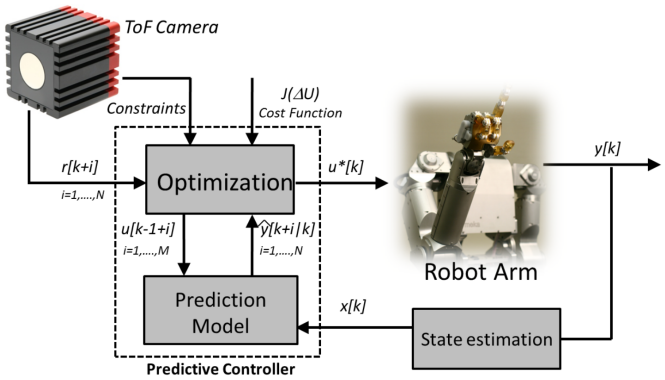


Figure 4-3: Visual Servoing MPC architecture [49].

safely navigate. The controller would most likely run into problems when posed with more complex problems.

4-1-3 Elastic band methods

Elastic Bands were first introduced in [50]. The method attempts to locally approximate the free space around a planned trajectory by using so called 'bubbles'. These bubbles are placed on the trajectory and increase in size until colliding with an obstacle. The bubbles are then used as part of a local optimization in order to find a feasible trajectory. Figure 4-4 shows the bubbles along a path.

By using proximity information, the elastic band method avoids calculating an explicit representation of the free space in the configuration space, which is very difficult. Instead, by

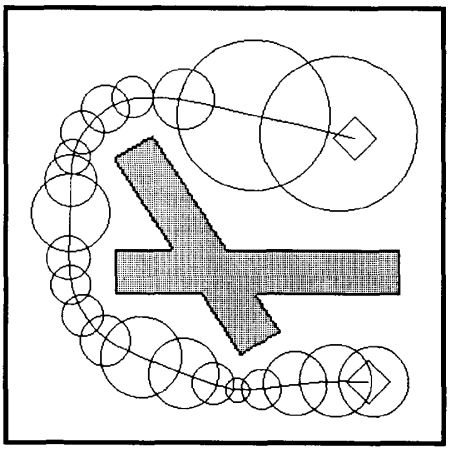


Figure 4-4: Bubbles along a path using the Elastic Band approach[50].

having the assumption on the free space close to the original path, a new valid path can be planned *very* quickly, effectively realizing near instantaneous replanning.

However, as the dimensionality of the problem increases, the estimate becomes excessively conservative which leads to higher computational cost and loss of real-time performance.

The method was extended into the Elastic Strip method in [51]. In the Elastic Strip method, a single collision-free path is used as a *candidate*. The path is translated to a swept volume in the workspace. By slightly changing the path in the configuration space, the volume of the free space around the candidate path is explored to form a so called *elastic tunnel* of free space. The elastic strip is the combination of the candidate path and the elastic tunnel. See Figure 4-5 for a depiction of an elastic tunnel. When encountering an obstacle, a potential

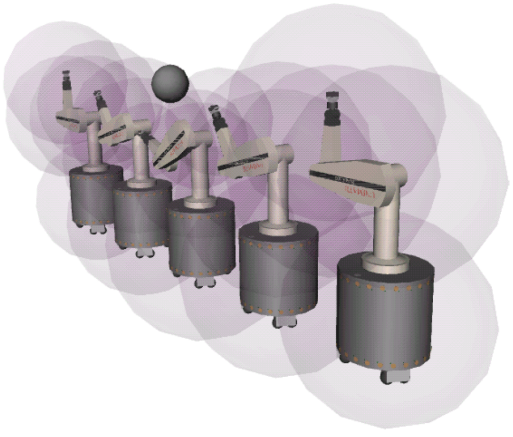


Figure 4-5: An Elastic Tunnel. The trajectory is modified in the presence of the dark colored spherical obstacle[51].

based approach efficiently selects a new candidate path that is enclosed in the elastic tunnel.

The elastic strip method is also capable of dealing with task constraints like, for example, maintaining a certain end-effector pose throughout the trajectory.

A related approach, called Timed Elastic Bands, was introduced in [52]. Timed Elastic Bands is an extension of the elastic band framework that also accounts for temporal aspects of the motion such as joint velocities and accelerations. Figure 4-6 shows how a system using the timed elastic band method works.

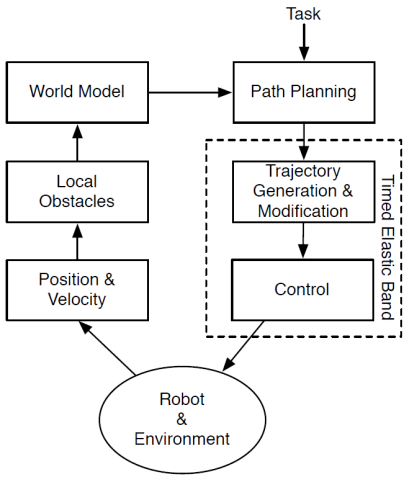


Figure 4-6: Robot system using the timed elastic band[52].

The timed elastic band performs a weighted multi-objective optimization in real time to find the time-optimal trajectory. Waypoints from the original path attract the elastic band and obstacles repel it. Timed elastic bands are still prone to getting stuck in local minima. If an obstacle moves through the path, it deforms the path in such a way that the the manipulator will always try to go around the obstacle on one side, while perhaps going along another side is much easier. Figure 4-7 shows how Timed Elastic Bands deal with obstacles.

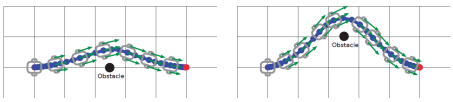


Figure 4-7: Timed Elastic Bands in the presence of dynamic obstacles[52].

One option that can be used is to solve this keep track of multiple trajectories with regards to obstacles and selecting the best one. This idea was introduced and extended in [53].

In [54], the method was benchmarked against CHOMP and showed improved performance in finding a smooth path around obstacles. Whether Timed Elastic Bands are capable of updating fast enough to be used as a dynamic control method in $\mathbb{R}^6$ remains to be seen.

Elastic Band methods have a number of similarities with the optimization methods described in section 3-4-2. A key difference is that Elastic Band methods only work whenever the initial path is already in free space whereas the other methods also work when the initial path is not collision free.

4-1-4 Comparison of dynamic control methods

This section will consider the performance of several different dynamic control methods. Due to a lack of clear experimental data and available implementations, it is somewhat more difficult to make an assessment when it comes to the dynamic control methods. However, an attempt will be made to compare the methods on the following criteria:

- **Time** The method needs to be able to handle an update frequency of about 10 Hz in order to effectively deal with dynamic obstacles. This column will indicate how likely it is that the method will achieve such performance.
- **Flexibility** Flexibility indicates whether the method is capable of dealing with more intricate (dynamic) constraints.
- **Practical** Indicates the complexity of implementing the method.
- **Whole-body** Indicates whether the method is capable of doing whole-body collision checking or just end-effector collision checking.
- **Smooth** Some methods smooth the path.
- **Optimality** Optimization methods are capable of optimizing for different cost functions like time, obstacle clearance, and others.

Making a judgment on the dynamic controllers is very difficult. Information on practical implementations is sparse, and it seems that a structured method to combine a global path planner with a local dynamic controller is not freely available. Table 4-1 lists the differences between the discussed approaches.

Method	Time	Flexibility	Practical	Whole-body	Smooth	Optimal
Potential Fields	++	-+	-	-	-	-
Elastic Bands	+	-	-	+	+	+
Elastic Strips	+	-	-	+	+	+
Timed Elastic Bands	+	-	-	+	+	+

Table 4-1: Comparison of different planners

Potential fields offer a very intuitive and simple approach to on-line local trajectory modification. However, as transforming obstacles to the configuration space is computationally hard, available implementations are limited to operating in end-effector space. One interesting direction would be to explore the option of treating each subsequent joint location as an end-effector and using multiple potential fields to avoid obstacles. This would still not offer full whole-body collision avoidance, but perhaps it will suffice for a large fraction of the realistic scenarios.

One disadvantage for these methods is that a currently available implementation that can be applied to the UR5 and integrated with the existing infrastructure does not seem to exist.

4-2 Time parameterization

As explained in section 1, time parameterization of a planned path is a central part of any motion planning system. Without time parameterization, the controller has no information on how to actually perform the planned path and will not be able to correctly follow the planned path. Simply streaming the waypoints of the path to the robot controller will result in the robot following an entirely different path from the one that was found by the path planning algorithm. Consider figure 4-8 for an schematic illustration of the problem that will be encountered.

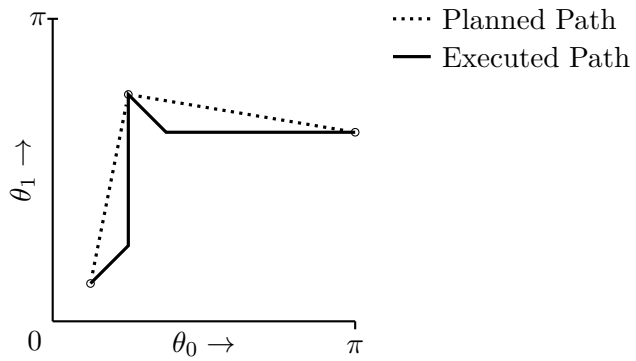


Figure 4-8: Resulting path of directly streaming waypoints to a joint controller. Joints are denoted θ_0 and θ_0 and have identical acceleration and velocity limits

Most time parameterization algorithms optimize for one of the following three objectives:

- Minimum Execution Time
- Minimum Energy
- Minimum Jerk

In industry, minimizing the execution time is usually preferred. In most cases, using more energy in order to be faster is more profitable. For minimum jerk trajectories, the execution time is usually fixed, as minimizing the jerk without constraining the execution time would lead to have an infinite duration.

An important distinction should be made between off- and on-line time parameterization:

In the **off-line** case, the complete trajectory is parameterized at once. All way points between the start and end point are known, and there is no hard limit on the calculation time, as the parameterization is technically a part of the planning.

In **on-line** parameterization, a new joint setpoint has to be calculated in each interpolation period of the robot controller. In order to guarantee safe execution, the runtime of the parameterization algorithm must then satisfy an upper bound. In contrast to off-line optimization, it is not strictly required to parameterize the complete trajectory, but that might be at cost of finding a non-optimal solution.

In Moveit, off-line time parameterization is used. Initially, waypoint time stamps are calculated using velocity constraints, after which an iterative procedure fits splines to the trajectory while satisfying acceleration limits. This results in a trajectory that does not *exactly* follow the planned path, but is somewhat smoothed, as illustrated in Figure 4-9.

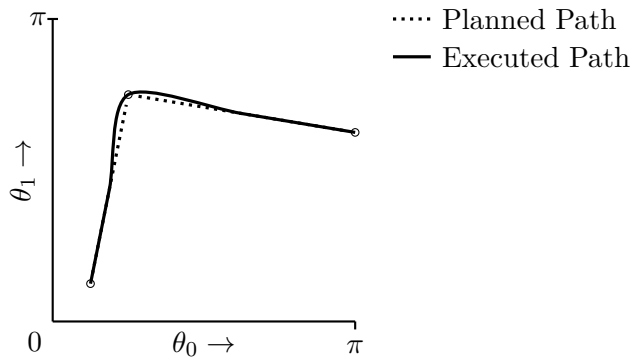


Figure 4-9: Exaggerated path of a time-parameterization using splines. Joints are denoted θ_0 and θ_0 and have identical acceleration and velocity limits

Since the geometric path is constrained, the only thing you can optimize is the scalar function $t \rightarrow s(t)$, which represents the "position" on the path at each time instant. If the system constraints are of the second order, the solution can be found in the 2-dimensional space $(s, \dot{s})$. As far as the before mentioned objectives of the parameterization go, a commonly used heuristic is to optimize for optimal-time execution, while satisfying a number of higher order constraints. Usually, manipulators are limited in their velocity and acceleration, leading to 2nd order constraints. However, as jerks can lead to oscillations and shorter component lifetimes, a jerk constraint is also desirable.

Following the classification of methods used in [55], three promising families of methods that are used for solving the time parameterization problem are found to be dynamic programming, convex optimization and numerical integration. A general overview of many different trajectory generation methods can be found in [56].

Dynamic Programming

Dynamic programming solutions divide the $(s, \dot{s})$ space into a grid and use dynamic programming to find the optimal trajectory. Several similar methods have been proposed [57][58], but most of these seem to stem from a fairly long time ago.

Thanks to computational efficiency, dynamic programming methods have also been proposed for on-line applications[59], but again, there doesn't seem to be any recent work in this direction.

Convex Optimization

In convex optimization approaches, the s-axis is discretized and the problem is turned into a convex optimization problem.

In some cases, extensive physical limits of the manipulator can be accounted for in the time parameterization. In [60], a Sequential Convex Programming approach is proposed to follow any given path in optimal-time. Different from earlier approaches, the method supports a wider range of constraints by using so called Difference Convex (DC) functions. This allows

for a much wider set of constraints including velocity-dependent torque constraints, viscous friction effects or the inclusion of cutting forces at the end effector. In addition, the approach includes Jerk limits on both the joint states as well as the end effector in the workspace. Several mentions of other convex optimization approaches can be found in [55] and [56].

Numerical Integration

Numerical integration relies on the fact that the time-optimal trajectory to a specified state is a *bang-bang* trajectory. this means that the trajectory is always either at the minimum or maximum acceleration. Consider Figure 4-10 for the a second-order bang-bang profile. Notice that for each section, the acceleration is either at the maximum, 0 or the minimum. It

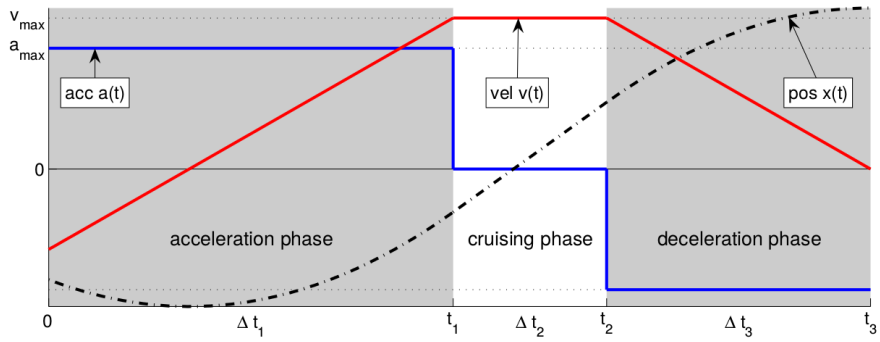


Figure 4-10: Canonical second-order profile comprising three phases[61]

is easy to see that for the given acceleration and velocity limits, this does indeed lead to the time-optimal trajectory. Increasing the order of the constraint, the "free variable" is not the acceleration, but the jerk. For a third order profile, have a look at Figure 4-11. Considering

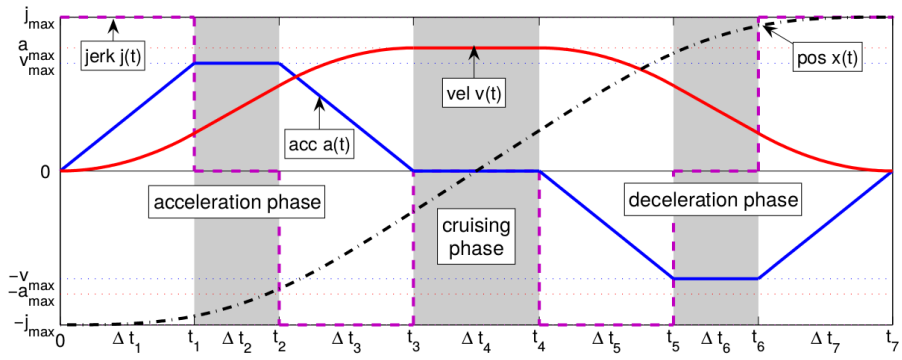


Figure 4-11: Canonical third-order trajectory profile consisting of seven phases: three acceleration, one cruising, and three deceleration phases. Due to equal initial and final conditions the profile is highly symmetric[61]

the jerk limit leads to a much smoother trajectory, but the numerical integration is a lot more complicated. In [61], a real-time safe on-line trajectory generation algorithm is proposed that manages to satisfy an upper limit on the computation time. In contrast to what is typically needed to parameterize a complete path, this algorithm only generates a trajectory to a single

target state, specified as a position and a velocity.

This algorithm was refined, cleaned up and released as the Reflexxes Motion Library[62]. Reflexxes offers two different libraries: Type II which can only handle acceleration (2nd order) constraints, and a Type IV that can also handle jerk (3rd order) constraints. Like before, it is important that the different joints of the robot are synchronized in order to execute the planned path. Reflexxes is able to take care of this synchronization in two ways, either *time* or *phase* synchronized, as demonstrated in Figure 4-12.

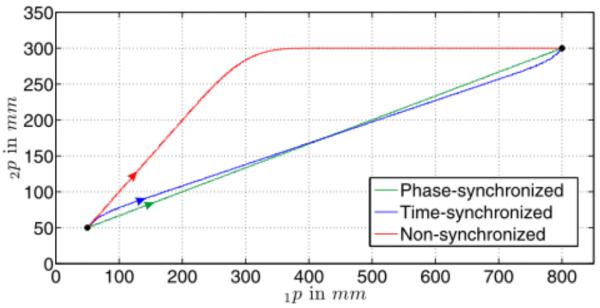


Figure 4-12: Synchronization behavior of the Reflexxes Library[63]

Accurate path execution is a topic that has relevance for a wide variety of applications. For example, robotic welding is an application where to execution of the path needs to conform to a much stricter bound than for some other applications. In [64], a case is made for path-accurate execution of planned paths. Several methods, including Reflexxes, are compared with respect to their ability to accurately track a certain path. Consider Figure 4-13 for an experiment of their proposed method. Clearly, their method manages to achieve a much

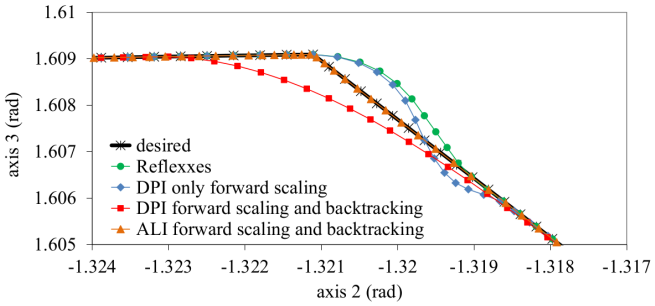


Figure 4-13: Path accurate execute for several different on-line time parameterization algorithms. DPI stands for Direct Position Interpolation and ALI stands Arc Length Interpolation[64]

better tracking of the given trajectory and it does so by slowing down when approaching sharp turn.

An approach that handles the complete path can be found in *Time-Optimal Parameterization of a given Path*, or TOPP[55]. As the name suggests, TOPP attempts to find the time optimal solution while satisfying a set of constraints. Unfortunately, TOPP does currently not support jerk limits. Unlike other approaches, TOPP is readily available in the form of a C++ library.

4-3 Replanning methods

Replanning is a method that is often encountered when researching path planning in dynamic environments. Whenever an obstacle invalidates part of the currently executing trajectory, a new path is planned and the obstacle will be avoided. Simply restarting the planning algorithm is a naive option, but most replanning approaches only remove the invalid parts and attempt to stay close to the original solution. This has been done both for RRT[65] as well as PRM[25]. Other methods, like [66], attempt to find several related paths between which a controller can switch during execution if need be. This is similar to the elastic bands methods from Section 4-1-3. These are commonly classified as reshaping or deforming the path instead of actual replanning. While this works as long as the obstacles are not reshaping the path *too* much, it will at some point deform a path into an invalid one.

A replanning method that uses RRT and discusses the practical aspects in more detail is found in [67]. In this work, the path execution process operates in a separate thread from the replanning process. Consider Figure 4-14 for an overview of the operation. In order to

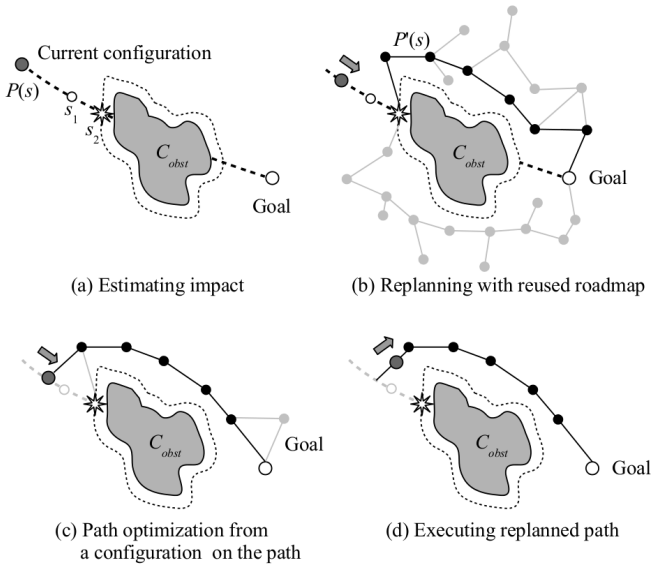


Figure 4-14: Replanning a path using the method in [64]

get a smooth transition from the current path to the replanned one, a method like adaptive shortcut optimization[68] can be used. If the path planner does not find a plan before reaching the point of collision, the execution of the robot is slowed down, before coming to a full stop. For obstacles that are further in the trajectory, this is a viable strategy as it is possible to wait before the obstacle removes itself from the trajectory before taking evasive action. In many cases however, the manipulator will have to come to a full stop.

In addition to slowing down and replanning when faced with a nearby obstacle, the method was later extended to perform path deformation up until the point that a new plan was absolutely necessary [69]. The path deformation method used is the elastic band method, which is also capable of smoothing the path when no obstacles are present. The method showed improved performance over the more straightforward replanning, which can be observed in Figure 4-15.

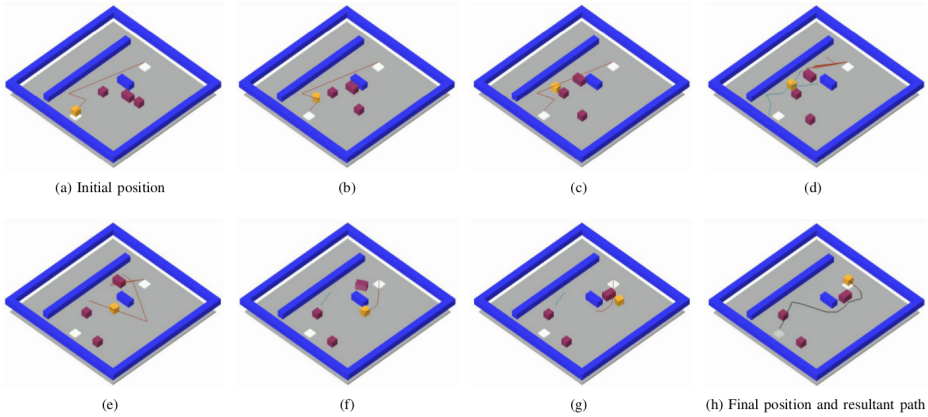


Figure 4-15: Results of reactive motion with deformation and replanning. The initial path is deformed at (b-d). Alternative path is planned at (d) and is executed (e-g) which is also deformed at (g). The resultant path is shown in (h)[69]

This is a very interesting result in that it combines several different methods that each have their respective ways of dealing with a changing environment. Unfortunately, it is not very clear how these methods would translate to a UR5, and an implementation of these methods does not seem to be available.

4-4 Conclusion

Throughout the preceding chapter, many different methods that are concerned with dynamically executing a path have been discussed. An unfortunate fact is that very few of these methods are readily available to be tested. Judging from literature, there is not a single best method. Unfortunately, due to the poor availability, performing a benchmark to determine which work best is not feasible.

In terms of practical application, an interesting combination can be found when considering path deformation methods with an on-line trajectory generator. By removing the need for time parameterization, it is much easier to dynamically alter the path, even while it is being executed. The elastic band methods are also of particular interest, as they seem to scale well towards the type of problems that a pick and place system might be faced with.

With path planning algorithms being a key ingredient, the next few chapters will focus on finding out which planner is most suited to be used in a dynamic motion execution system.

Part II

Planner benchmarking

Judging from the current state of literature, it is unclear what the performance of the selected planning methods will be for the environment at Technolution. In order to get meaningful data that can aid in making a well justified decision, a benchmark has been devised in that aims to simulate a realistic scenario in order to finally arrive at a well justified performance analysis of the different planning algorithms. Part 2 will focus on this benchmark. First, Chapter 5 will detail the benchmarking methodology, after which the results of the benchmark will be presented in Chapter 6.

Chapter 5

Benchmarking method

Recall the following three ways to formulate a controller that achieves the specified performance:

1. Find a method that can guide dynamic controllers to globally complete solutions and mitigate local minima problems
2. Increase the speed of a global path planner in order to effectively deal with dynamic obstacles
3. Integrate several approaches that each tackle a specified subproblem

The first method was rejected as it was found that such systems do not exist. A common requirement for both is the use of a global path planner. Depending on the actual method, the requirements on the path planner will differ, but for both methods, a path planner that can solve a large percentage of problems within a relatively short time is needed.

In order to select which planner is most suitable for this scenario, the planners need to be benchmarked in a realistic way.

5-1 Methodology

In order to find the planner that is most suitable for this application, a benchmarking method was devised that attempts to closely model the planning problems and scenarios that the system is expected to face. In light of finding a planner that works well in a changing environment, it is important to acquire performance figures on a set of different environments. By only testing the planners on a single scenario, it is possible that a certain planner seems very well suited for the task, but is not capable of solving queries for more difficult scenes.

In order to get an idea about the performance for changing environments, 3 different environments of increasing complexity were constructed. Moveit was used to model the scene, and the planners that are to be tested are part of the OMPL planning library.

Algorithm 1 Benchmarking procedure

Input : Planners P, Scenes (with N defined problems) Ψ, Maximum iterations n
Output : Planning Results Data *PlanData*

```
for planner = P(0), ..., P.back() do
    for scene = Ψ(0), ..., Ψ.back() do
        for problem = scene.problems(0), ..., scene.problems.back() do
            for index = 0, ..., n do
                Timer.start()
                PlanResult = plan(planner, scene, problem)
                PlanResult.time = Timer.stop()
            PlanData(planner, scene, problem) = Average(PlanResult)
return PlanData
```

Consider Algorithm 1 for an overview of the fairly benchmarking procedure.

As this procedure repeatedly requires the planner to plan from the exact same set of states, it is not possible to get a realistic representation of the performance experience based planner Thunder. The specific experiments that were used to compare Thunder are described in Section 6-3. The metrics regarding the performance of the planner and path quality are stored in **PlanResult**. Consider Table 5-1 to see what data is stored for problem query.

Property	Unit
Algorithm Runtime	ms
Solution found	boolean
Path Length in configuration space	rad
Path Length in workspace	m

Table 5-1: Contents of the *PlanResult* of the benchmarking procedure.

In light of aiming for a realistic scenario, these planners will be judged on their *total* runtime. this means that an external script is used to invoke the planners, and timing and other metrics will be calculated outside of Moveit. Consider Table 5-2 for an overview of the system that was used for the benchmark.

Component	Specification
CPU	i5-3470 @ 3.20 GHz
Memory	8 GB
OS	Ubuntu 14.04.4 LTS
ROS	Indigo Igloo

Table 5-2: Specifications of the system that has been used for the benchmark

5-2 UR5 kinematics

As the kinematics of the UR5 have a big influence on the *difficulty* of a specified scenario, a shallow analysis of the kinematics of the UR5 will be discussed in this section. Consider Figure 5-1 for a look at the dimensions and kinematic structure of the UR5.

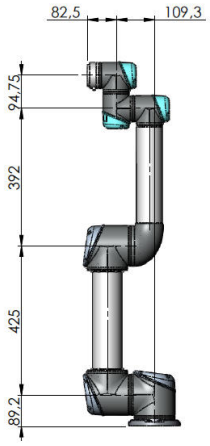


Figure 5-1: UR5 Dimensions

Table 5-3 displays more detailed information on the joints of the UR5. Physically, the UR5 supports a full rotation in each direction, however, a known problem in Moveit is that planning times for manipulators with joint limits greater than a single full rotation are very long, effectively rendering the increased joint limits useless. Naturally, this is unfortunate as the increased joint limits would potentially offer a greater range of motion for the robot.

Joint	Name	Physical Joint Limits	Moveit Joint Limits
1	Base	$-2\pi : 2\pi$	$-\pi : \pi$
2	Shoulder	$-2\pi : 2\pi$	$-\pi : \pi$
3	Elbow	$-2\pi : 2\pi$	$-\pi : \pi$
4	Wrist1	$-2\pi : 2\pi$	$-\pi : \pi$
5	Wrist2	$-2\pi : 2\pi$	$-\pi : \pi$
6	Wrist3	$-2\pi : 2\pi$	$-\pi : \pi$

Table 5-3: Specifications of the Joints of the UR5

Looking at the robot kinematics, a number of things are important to observe.

- **Wrist Configuration** The wrist configuration of the UR5 is a little more "bulky" than many other wrist configurations. With the three joints being close together and not able to be aligned, there is always a considerable risk of having self collisions.
- **Limited IK solutions** As the UR5 is a 6 DoF holonomic robot, it is expected to be able to reach every pose in its workspace. However, being *only* 6 DoF and being limited

by self collisions and joint limits, means that in some cases, the UR5 will suffer from jumps in the IK solution.

- **Major joints: Base, Shoulder, Elbow** When considering collisions with external obstacles, the Base, Shoulder and Elbow joint have the most influence, as changes in those joints have a bigger influence on the workspace occupancy of the manipulator.

To expand on the self collisions, consider table 5-4 for an indication on how much of the search space is invalidated by self collisions. The combination of Wrist_1 and Wrist_2 is included

Joints	Discretization Levels	Total States	Self-collisions
0:4	25	9 765 625	29%
0:4	35	52 521 875	27%
0:5	35	18 382 656 255	0.8%
3:4	35	1 225	11%
3:4	500	250 000	11%

Table 5-4: Percentage of the UR5 workspace that is invalidated by self collisions for different joint combinations.

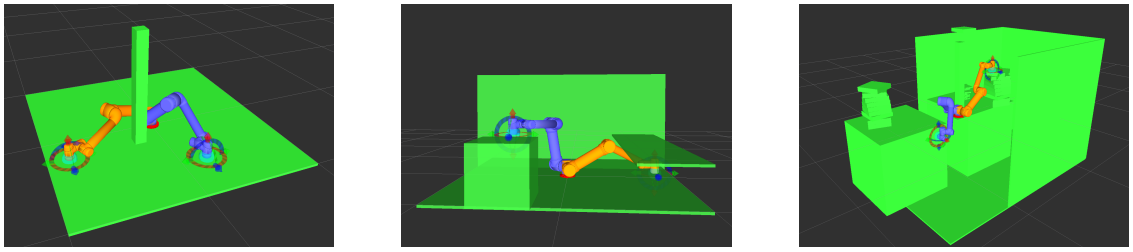
as an illustration of the unfortunate wrist kinematics, as well as way to get an idea about the convergence of the percentage of self-collisions as a function of the discretization levels. The final wrist joint was omitted as it does not contribute to self-collisions when no end-effector is attached. When considering the last joint for this scenario as well, the percentage of self collisions drops below 1%. While this is by no means a full analysis of the kinematics of the UR5, it does offer a glimpse of the difficulty that a random sampling based planner faces when planning for a robot arm with such a structure.

5-3 Scenes

- Scene 1 is a fairly simple scene with only two obstacles; a floor and a pillar. As the pillar is close to the base of the manipulator, this scene is not as easy as it might appear on first glance. In order to pass the pillar, the manipulator needs to be angled all the way upward. This means that there are tight constraints on the base, shoulder and elbow joints. This is can be considered a *narrow* passage, which is typically a problem for random sampling based planners as odds of sampling a valid state that successfully traverses the narrow passage is relatively small.

- Scene 2 is more difficult. The number of objects is still low, but geometry allows for more difficult problems, with both the start as the goal state in difficult to reach positions. Especially configurations where the end effector of the manipulator is below the shelf were found to increase the planning complexity.

- Scene 3 has the most obstacles. Other than the number of obstacles, the scene geometry is similar in complexity to the other scenes. As with the first scene, the pillar is close to the base, and this there are more obstacles to be tested for collision.



(a) Scene 1 **(b)** Scene 2 **(c)** Scene 3

Figure 5-2: Planning Scenes.

For each of these scenes, a set of 5 different collision-free poses were stored as start and goal positions to the planning problems. The poses were chosen to be representative of a realistic pick and place problem. With 5 different states, the total number of unique planning problems was 10 per scene. Each of these problems were iterated for a large number of times in order to find an average planning time. Other metrics like joint and workspace path length are also saved.

5-4 Automated planner configuration

One problem that has to be addressed when doing a meaningful benchmark is planner configuration. In the past several decades, a very large number of different planning algorithms have been proposed. Many of these methods are designed to solve a certain set of problems that have some specific characteristic. Consider T-RRT, briefly discussed in section 3-3-2, which efficiently manages to solve the problem it was posed for, but whether the added heuristics can also be used for other purposes is unclear.

With the number of options available, selecting a planner for a given problem is not a trivial task, especially for one not familiar with the field. Configuring a planner is even more of a daunting task. With the number of configurable parameters ranging from just 1 for RRT-Connect to 12 for RRT*, even an expert will have a hard time predicting the ways in which these parameters will influence the performance of the planner. Especially when several heuristics are combined, the problem becomes completely intractable.

Thankfully, specially designed optimizers exist that attempt to find an the best configuration to an algorithm. These optimizers continuously pose new configurations with an identical benchmark in order to find the best performing configuration. The way that different methods differ from each other is often related to the way they select a new contender. Until recently, most of these methods were model free, and used some sort of local gradient method to converge towards a better solution. Recently however, several model-based methods have been developed that showed very promising performance on a wide set of problems. Sequential Model-based Algorithm Configuration (SMAC), introduced in [70], is such a model-based approach.

In order to use SMAC to find the best configuration for the planners that are to be tested, a problem instance similar to the benchmarking method described in Section 5-1 was generated. Care has to be taken to ensure that the resulting configuration is not only effective at solving

a specific problem, but also generalizes well towards a greater class of problems. Consider Algorithm 2 for an overview of the methodology used to configure the planners.

Algorithm 2 SMAC Configuration Tuning

Input : Algorithm A, Problemset Ψ with N problems,
Environment X, Iterations n, default configuration Φ_0
Output : Incumbent configuration Φ_{inc}

1: $\Phi = \Phi_0$
2: $best = \infty$
3: **while** *SMAC.Runtime* $<$ *AllowedRuntime* **do**
4: $result = 0$
5: **for** $i = 0, \ldots, N-1$ **do**
6: $\psi = \Psi(i)$
7: **for** $0, \ldots, n$ **do**
8: Timer.start()
9: $A(X, \Phi, \psi)$
10: *result* += Timer.stop()
11: **if** *result* $<$ *best* **then**
12: $\Phi_{inc} = \Phi$
13: *best* = *result*
14: SMAC.UpdateModel(*result*, Φ)
15: Φ = SMAC.GetNewConfig()
16: return ***Incumbent***

The actual problem is solved at step 9: $A(X, \Phi, \psi)$. Here, an Algorithm A with configuration Φ is posed with problem ψ in environment X. By supplying the algorithm with varied environments and problems, the resulting configuration is aimed to be as general as possible. For a more detailed description of the configuration method, the reader is referred to Appendix A, the paper written on the method that has been submitted to IROS 2016.

The tuning results of the first problem discussed in the paper, which was modeled after the Technolution environment, is reproduced in Table 5-5. Clearly, several of the methods display a substantial performance increase thanks to tuning.

5-5 Selected planners

During testing, tuning and integration, several observations regarding the selected planners were made. A final selection was made and the planners that will be benchmarked are:

- **RRT-Connect**
- **BiTRRT**
- **BIT***

Table 5-5: Tuning results for the selected planners.

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	36.3	100	7.3
RRTConnect - Tuned	36.0	100	7.2
BiTRRT	46.8	100	7.1
BiTRRT - Tuned	45.8	99	7.2
BKPIECE	254	99	7.8
BKPIECE - Tuned	108	99	7.4
KPIECE	95.7	100	7.8
KPIECE - Tuned	40.3	100	7.5
BIT*	102	92	9.0
BIT - Tuned	52.2	99	8.9

- **KPIECE**
- **BKPIECE**
- **RRT**

It was also observed that Thunder requires a different benchmarking method, as repeatedly posing an identical problem to a planner that stores previously found paths is not a very representative benchmark. This will be discussed further in Section 6-3.

Because of the promising results from the paper, a serious effort to integrate the optimization based method TrajOpt into MoveIt!. Unfortunately, due to time constraints, it was not possible to finish the integration up to the point where it was useful. The intention is to finish this work after the research is concluded, so as to release it as an open-source plugin to MoveIt!.

For the optimal planning benchmark the planners that will be used are:

- **RRT***
- **BIT***
- **RRT-Connect**

While RRT-Connect is not an optimal planner, it was modified to utilize the longer planning times and choose the best found path according to the same cost function as the optimal planners. The planner will be ran for the allocated planning time while only storing the shortest path it finds. While the mathematical derivation on whether this would qualify as an *optimal* planner has not been performed, it is expected that the planner is able to improve on the initially found path at the very least.

Chapter 6

Results

In this chapter, the results of the benchmark will be discussed. The first section will discuss the results of the benchmark that was focused on the solving percentages and planning time. In Section 6-2, the performance of optimal planners will be investigated. Experience based planners will be discussed in Section 6-3. Finally the a concluding remark on the performance will be given in section 6-4.

6-1 Planner speed

With the solution speed of the global path planner being a key part of the performance of the system, the first benchmark was designed to find out which of the proposed planners would be able to quickly solve the problem with high solving rates. The following settings were used:

- **Planning timeout** was set to 1s. Although this is relatively short, a planning time for more than 1s is deemed too long and can be considered a failure.
- **Scene problems** 10 problems were defined for each scene.
- **Planning iterations** was set to 100 iterations for each problem. Given 3 scenes and 10 problems, this means each planner is queried a total of 3000 times.

6-1-1 Results

Table 6-1 shows the solving rates for each scene of the benchmark.

It is immediately obvious that the bidirectional planners manage to solve a very large fraction of the problem. BIT* has the lowest solving rate, followed by RRT and KPIECE. It is interesting to see KPIECE gets achieves worse solving rates for more difficult scenes, and RRT actually gets *better* solving rates.

Planner	Scene 1 [%]	Scene 2 [%]	Scene 3 [%]	Total [%]
RRTConnect	98	99	98	98
BiTRRT	98	99	99	99
BIT*	78	78	85	80
BKPIECE	98	99	98	98
KPIECE	98	89	85	91
RRT	81	84	93	86

Table 6-1: Solving rates for the selected planners

What is also interesting to see is that the *way* that each planner fails is different across problems. Take Scene 3. Both BIT* and KPIECE achieve a similar solving rate, but as we have already seen before, BIT* can get stuck on specific problems. Consider the Figure 6-1 for the results of BIT* and KPIECE on Scene 3.

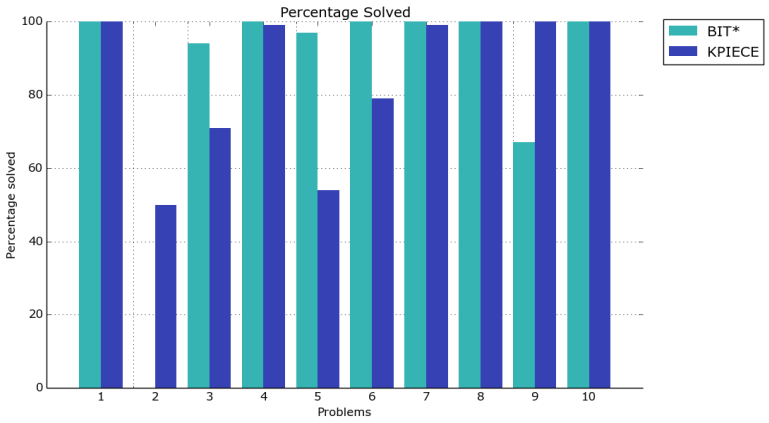


Figure 6-1: Solving percentage bar chart of the 10 problems of scene 3 for BIT* and KPIECE.

The same problem is again observed in this case. With these observations, BIT* is not a suitable candidate for this system, even though it does a better job on the remaining problems. These problems that seem "unsolvable" to BIT* appear in other scene's as well. Consider Figure 6-2 for such a problem in scene 1. The reason why BIT* cannot manage to find a solution is unclear, and further research would be required to determine the cause.

Going further, Table 6-2 presents the relative computation time performance of the planners. For each separate problem, the planners are evaluated in terms of computation time relative to the average across all planners. Then, the fractions for all problems in a single scene are average to provide a single performance figure per scene. Although this seems like an overly complicated performance metric, it was deemed to give the most insight into the relative performance between the planners.

Drawing conclusions from the averages for each scene, it can indeed be said that the planners have more difficulty quickly finding plans for the more complex scenes. As before, RRTConnect and BiTRRT are among the top-performing planners. It was to be expected that these would be the two top contenders with regards to computation time. A somewhat surprising

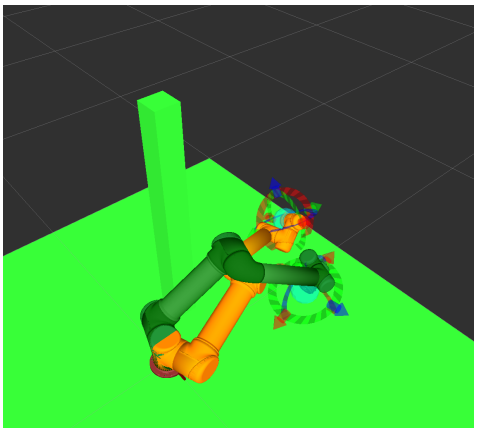


Figure 6-2: 9th problem of the first benchmark scene. The start state is depicted in green and the goal state in orange. In order to reach the goal, the manipulator base needs to rotate about 150 degrees clockwise, and flip over backwards towards the goal.

Planner *Average time*	Scene 1 *149ms*	Scene 2 *219ms*	Scene 3 *322ms*	Total *230ms*
RRTConnect	0.49	0.78	0.57	0.61
BiTRRT	0.63	0.83	1.09	0.85
BIT*	1.17	1.18	0.81	1.05
BKPIECE	1.06	1.14	1.59	1.26
KPIECE	1.14	0.98	1.06	1.06
RRT	1.51	1.09	0.89	1.16

Table 6-2: Computation time results

result is the big difference in the third scene. In contrast to the solving percentage, BKPIECE does not manage to score very well when it comes to computation time. As a matter of fact, *when* KPIECE manages to find a solution, it does so quicker than BKPIECE, especially for the more challenging scenes.

Planner *Average length*	Scene 1 *11.3*	Scene 2 *8.6*	Scene 3 *9.6*	Total *9.8*
RRTConnect	0.93	0.93	0.99	0.95
BiTRRT	1.07	1.16	1.02	1.08
BIT*	1.15	1.11	1.18	1.15
BKPIECE	102	0.93	0.99	0.98
KPIECE	0.98	0.93	0.98	0.96
RRT	0.96	0.95	0.97	0.96

Table 6-3: Joint length results

To get some insight into the quality of the solutions, a similar comparison to that of the computation time is done for the path length in the configuration space as well as the workspace.

Considering these path lengths, it remains difficult to make a meaningful assessment of the actual path quality as neither the joint or work space graphs are an accurate measure for the path quality. As a consequence, these metrics will only be treated as an indicator for the ability of the planner to find reasonably short paths. Table 6-3 holds the results for path length in configuration space and Table 6-4 for the workspace path length.

Planner *Average length*	Scene 1 [%] *3*	Scene 2 [%] *2.9*	Scene 3 [%] *3.5*	Total [%] *3.1*
RRTConnect	0.95	0.93	1.00	0.97
BiTRRT	1.07	1.19	1.01	1.09
BIT*	1.06	0.92	1.05	1.01
BKPIECE	1.01	0.98	1.04	1.01
KPIECE	1.03	1.00	1.01	1.02
RRT	0.98	0.98	1.00	0.99

Table 6-4: Workspace length results

The differences are very small in both cases, but perhaps even smaller for the workspace lengths. Judging from these results and regarding the fact that the path length is only meant as an *indicator* for the path quality, it is not possible to draw any conclusions.

Summarizing the results of this benchmark, it is obvious that RRT-Connect easily manages to find a solution the quickest with BiTRRT being a close second. Considering the solving rates, the only feasible planners are:

- RRT-Connect, BiTRRT and BKPIECE for simple environments
- RRT-Connect and BiTRRT for more complex environments

The differences in path quality are difficult to classify but the difference does not seem to be significant for these set of planners. Judging from the initial solution, **RRT-Connect** seems to be the most suitable planner. The next chapter will investigate optimal planners to find out whether they can offer a significant improvement when allowed more planning time. If BIT* or RRT* is able to improve the path quality and solving rate by using more planning time, it might prove to be a better candidate than RRT-Connect.

6-2 Optimal planners

Optimal planners are capable of optimizing to a specified cost function in order to arrive at a 'better' solution. Selecting an optimal planner will mean striking a balance between the planning time and the path quality. If the planner is able to significantly improve the path quality within a reasonable amount of time, it can be worth it to increase the allowed planning time.

The planners and allowed planning times that will be tested are as follows:

- **BIT*** 1s, 2s, 5s, and 10s

- **RRT*** 1s, 2s, 5s, and 10s
- **RRT-Connect** 1s, 2s, 5s, and 10s

An optimal planner decides whether a solution is better using a specified cost function, also called the optimization objective. As briefly touched upon before, it is very difficult to specify a cost function that will *always* provide good paths. The cost functions that were ultimately considered are:

- 1-norm configuration space cost
- 2-norm work space cost

The reason for using the 1-norm is a practical one. Due to a strange convention in MoveIt!/OMPL, the default cost function is the 1-norm in the configuration space. Unfortunately, this came to light at a stage in the research where re-doing the experiments was not feasible.

6-2-1 Results

Table 6-5 shows the results of the first benchmark scene using a 1-norm joint space optimization objective.

Planner	initial solution	1s	2s	3s	5s	10s
		Percentage of initial				
BIT* - length	**13.7**	100%	98%	97%	95%	96%
BIT* - solve rate	78%	74%	80%	84%	85%	86%
RRT* - length	**10.8**	92%	88%	87%	85%	85%
RRT* - solve rate	65%	69%	76%	80%	81%	84%
RRT-Connect* - length	**10.57**	89%	84%	84%	83%	80%
RRT-Connect* - solve rate	98%	93%	91%	96%	92%	93%

Table 6-5: Configuration space path length optimization results for the first scene.

Going from the initial solution to a 1s planning time, there is a significant improvement for most planners, but after that it mostly stops. Given a lengthy planning time of 10 seconds, the improvement is still rather small. BIT* is not able to optimize the relatively lengthy initial paths into short paths within the given planning time. Consider Figure 6-3 for a plot of the improvement over time.

Looking at the solving rates of the planners, we see that neither RRT* nor BIT* manages to consistently solve the difficult problems given more time. Judging from these results, choosing an optimal planner does not offer significant improvements. One thing that does stand out is that Iterating-RRT-Connect manages to solve fewer queries than in the previous benchmark, even though it has more allowed planning time. This is likely due to RRT-Connect finding paths that are very close to obstacles, which are rejected in a later stage. Further research would be needed to rule out other problems. Next, consider Table 6-6, where the results of the benchmark with a *workspace* optimization objective are tabulated. The planners clearly

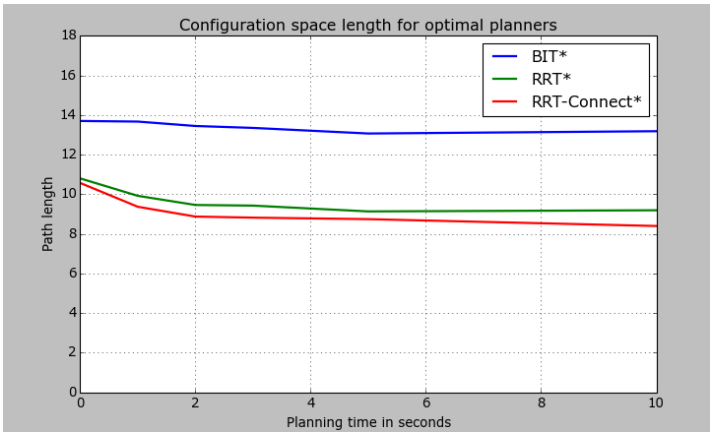


Figure 6-3: Plot of the configuration space length with increased planning times.

Planner	initial solution	1s	2s	3s	5s	10s
		Percentage of initial				
BIT* - length	**13.7**	107%	102%	100%	97%	99%
BIT* - solve rate	78%	74%	80%	84%	85%	86%
RRT* - length	**10.8**	102%	99%	97%	95%	92%
RRT* - solve rate	65%	69%	76%	80%	81%	84%
RRT-Connect* - length	**10.57**	105%	101%	98%	96%	93%
RRT-Connect* - solve rate	98%	93%	91%	96%	92%	93%

Table 6-6: Workspace length results

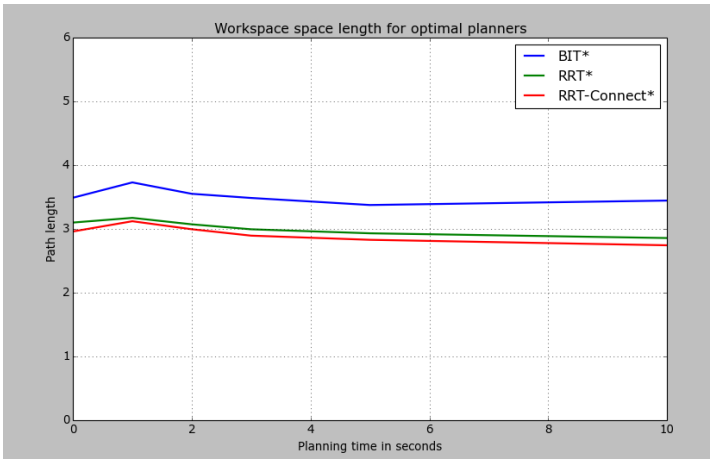


Figure 6-4: Plot of the workspace space length with increased planning times.

have a problem with optimizing the workspace path length, as the improvements are not very great. Again, consider Figure 6-4 for a more explanatory graph.

Note: the fact that the initial solution performs better at workspace optimization is likely related to a mistake made while benchmarking. Unfortunately, by the time this was discovered, it was not possible to redo the benchmark.

Regardless of the problems with the workspace planning benchmark, it is now clear that optimal planners are only able to improve the solution to a very small degree. Planning times of over 2 seconds are barely able to improve the solution. Further research into whether a 500ms planning time would perform better than the initial solution could be of interest.

6-3 Experience based planners

The experience based planners require a little deeper look in order to make a meaningful assessment. Of the different planners, the Thunder framework is most likely to be the best performing option. Thunder will be compared against RRT-Connect as this was found to be the best performing planner of the standard planning algorithms.

In order to get an initial idea of the performance, an identical benchmark as described in section 6-1 was performed. This would be an ideal case for Thunder, as an identical query is performed several times in a row. Thunder was configured to start each new scene with a fresh database, and store each result directly following the query. Consider Figure 6-5 for the results of the third benchmark scene.

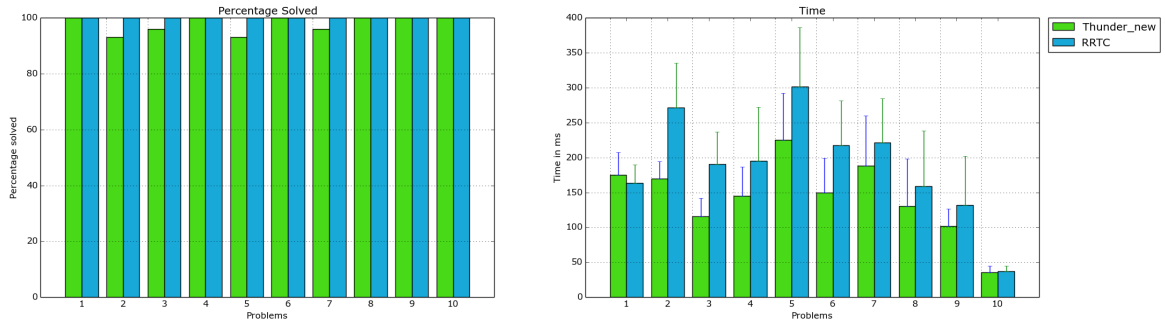


Figure 6-5: Plot of the length and solving percentages and planner runtime of the third scene of the benchmark. Each planner completed 30 iterations of each problem. The error bars denote the standard deviation.

The reason the third scene is shown is that the biggest differences can be observed for the more complicated scenario's, but even in this ideal case, they are not very great. An analysis of the differences between the planners revealed an interesting pattern. Thunder was consistently able to plan geometric paths very fast, but would subsequently lose some of that time on in the *path simplification* stage.

The path simplification stage is a Moveit class that attempts to quickly improve the quality of a given path by performing a number of heuristic functions within a short amount of time. These include attempting to find valid shortcuts, smoothing the path and finding a better goal state. Technically, this stage can be omitted at cost of less polished paths, but in the case of Thunder, disabling this stage caused a lot of paths to be dismissed during post processing. This suggests that these paths are very close to collision, and Thunder relies on the path simplifier to ensure that the final path is valid.

Further testing with Thunder has demonstrated that when re-using a database of stored paths, the planning time is indeed significantly reduced. However, Thunder has several other downsides:

- As a path planning system will most likely be tasked with a large number of very similar problems, it is likely that Thunder would present a single solution for each of these problems. Being "trained" by a random sampling planner, it is possible that this is a highly undesirable path. With the current implementation, it is not possible to prime Thunder towards better solutions.
- As most of the paths will be in the free space of the manipulator, the database of paths will also be a mapping largely consisting of free space paths. For a avoiding obstacles, a fast response time is needed when unexpected parts of the workspace are suddenly invalidated by obstacles. This is typically a problem where Thunder would offer no benefits.

These downsides, combined with the observed performance of the non-experience based planners, lead to the conclusion that further research into Thunder was outside the scope of this research.

6-4 Conclusion

Recall the research goals as specified in Section 1-2.

1. Grasp objects in all feasible locations of the workspace
2. Move in a predictable manner
3. Avoid both static and dynamic obstacles that would otherwise disrupt the task
4. Respond to events or changes that affect the current task (for example, dropping an object or a moving target location)

Recognizing that a large portion of the motion that the manipulator will perform during a pick and place task is dictated by the path planner, it is seen that the performance of the path planner is related to goals 1, 2 and the static part of 3.

Unfortunately, it was found that none of the more novel path planning methods were able to perform better than RRT-Connect. In terms of overall performance, KPIECE could be ranked 2$^{\text{nd}}$. But with a significant increase in planning time, without seemingly being better than RRT-Connect at avoiding weird paths, there does not seem to be a solid reason to prefer KPIECE.

If a higher time budget is available to the planner, it would most likely pay off to select a planning strategy where RRT-Connect is penalized for producing overly long paths. For instance, when the reactive part of planning is not reliant on a global path planner, planning might not negatively impact performance even if it takes up to 1 second. Even though not fully reflected in the benchmarking results, RRT-Connect will tend to produce very long paths from time to time. Running several instances of RRT-Connect, either in parallel or sequentially, is a proven and effective method to filter out these negative outliers.

When considering the research goals, it seems that it is indeed possible to respond to any changes in the environment within 200ms using RRT-Connect. However, as current implementation of the planner are not able to reuse parts of the tree from the initial planning query,

it is likely a replan will be completely different from the current path. This would then violate the requirement of smooth motions. In addition, it is very doubtful whether replanning can be used to accurately and quickly track a moving target. Therefore, a combination with some sort of dynamical method will be required. The next part of this thesis will focus on such an approach.

Part III

Dynamic path execution

It is now known that a fast path planner is able to respond to a changing environment within a very short time. However, it is expected that the performance can be greatly improved by combining it with a more dynamic control method. Unfortunately, it is difficult to do so using existing and available software. Part 3 will focus on experiments that have been conducted with regards to a more reactive system topology. First, the experimental setup will be detailed in Chapter 7. Then, a number of methods that were used to perform pose tracking are discussed in Chapter 8. Methods that can be used for dynamically executing a path in order to avoid collisions are discussed in Chapter 9 and finally, several concluding remarks are presented in Chapter 10.

Chapter 7

Experimental setup

In this chapter, the experimental setup will be discussed. Several practical issues have to be considered in order to conduct a valid experiment. The high level controller architecture will be discussed in Section 7-1, and the system architecture will be discussed in 7-2.

7-1 Controller architecture

In order to test different approaches to the problem, several practical choices have to be made. The first consideration is related to the difference between several types of dynamic constraints on the motion. Recall the two main objectives related to dynamic motion execution:

- **Obstacle avoidance**
- **Target tracking**

To relate this to a picking motion, these constraints are expected to be dominant in different parts of the complete motion. More precisely, it is likely that a obstacle avoidance will be dominant in the global path execution, and tracking in the Cartesian approach. As the target changes are defined to be over a comparatively small distance, the global path will be largely similar, even for a moving target. While obstacle avoidance is a legitimate concern when tracking a target pose, a limitation of the solution space means that an avoidance strategy is difficult to formulate. This will be elaborated on in Section 8-1. In order to switch between the global and Cartesian motions, the overall system is dictated by a flow diagram as shown in Figure 7-1.

7-2 System architecture

For practical reasons, most of the experiments have been performed in simulation. By using the Universal Robots URSim, which is effectively the same software that is running on the

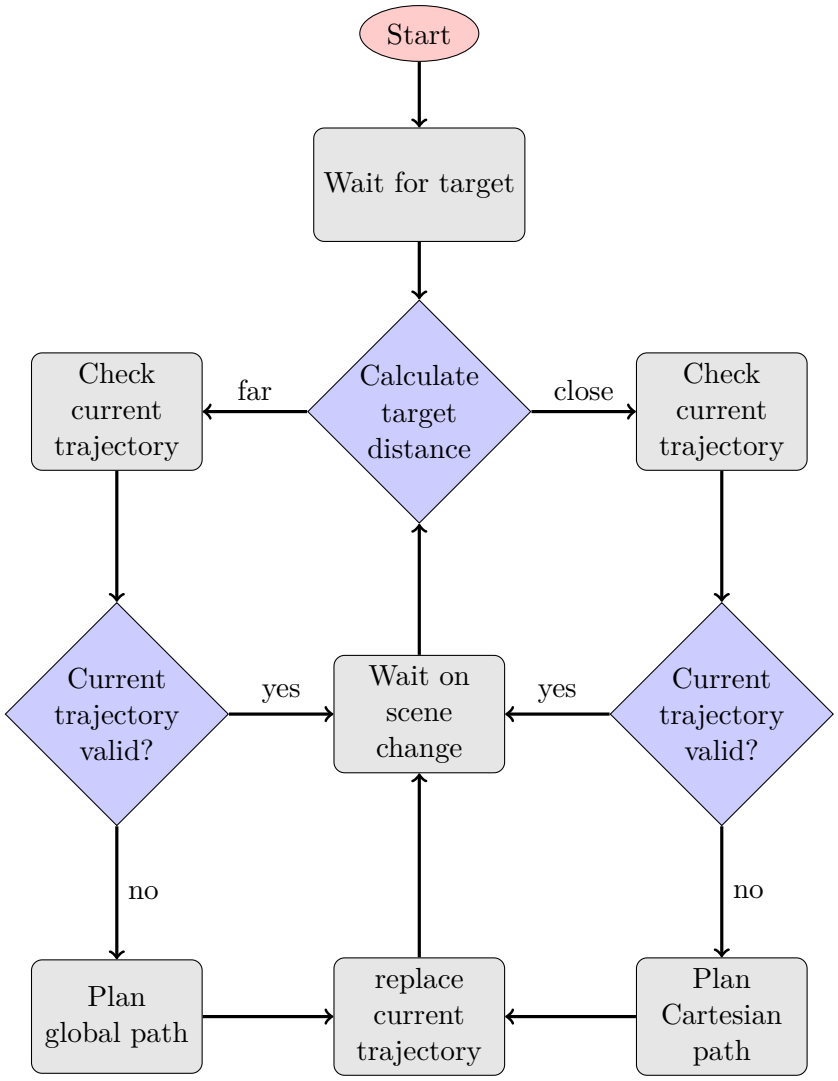


Figure 7-1: Simple Flow diagram for tracking behavior

UR5 itself, the practical situation is modeled as closely as possible. URSim uses the exact same interface as the actual UR5, so running an experiment on the actual robot is as simple as changing an IP address. While the perfect forward simulation that the UR5 employs to model the robot behavior is too optimistic, it has been confirmed that simulated behavior is close to what the actual robot does.

In order to exploit available software frameworks as much as possible, it is vital to first examine their performance and flexibility. By default, MoveIt! has very limited support for dynamic control. The standard method consists of sending a full trajectory to a robot controller and waiting for it to finish. For Moveit, a trajectory is defined as a set of waypoints that are to be reached at a specified timestamp. These waypoints are expressed by the joint positions, velocities and efforts. During execution, the robot controller attempts to follow the trajectory as defined by these timestamped waypoints. In between the waypoints, the controller performs a cubic interpolation to convert the waypoints to a smooth trajectory.

Coupled with the MoveIt! time parameterization, this leads to fairly smooth trajectories.

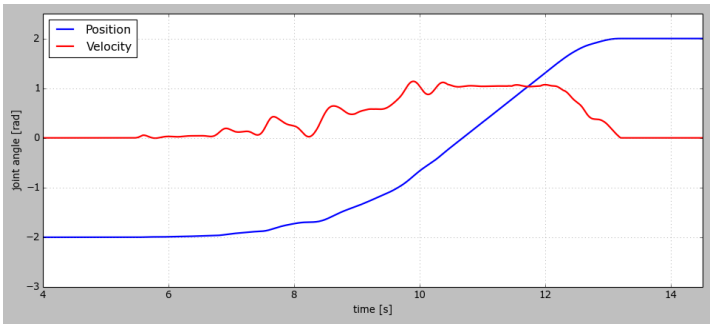


Figure 7-2: Single joint state evolution for a MoveIt! trajectory executed using the standard MoveIt! controller.

The MoveIt! controller is well tested and is capable of handling a tight control over the manipulator without incurring too many vibrations and corrections. Unfortunately, MoveIt! currently does not provide sufficient control over the execution of a trajectory to be used for these experiments. One option that might be available in the future is *ros_control* which supposedly has support for trajectory replacement[71]. This would allow the user to send a new trajectory to the controller, which will stitch the new trajectory to the current one, without halting execution. Details on the implementation are vague, and the driver for the UR5 is currently not working leading to a custom controller that was implemented for these tests.

Recently, an updated UR5 ROS driver was released that supports direct URScript pass through [72]. URScript is the language in which the UR5 can be commanded. Using the real-time interface, it is possible to steam new commands to the robot at 125 Hz. Unfortunately, not all control interfaces of the UR5 are suitable for being commanded at 125 Hz. An extensive analysis on the performance of the UR10 [73] has shown that the only interface that is capable of effectively handling command interruptions is the *speedj* command. Using the *speedj* command, a new joint velocity setpoint is sent UR5. As the main objective of the controller is to track the position, using velocity control is not ideal as it introduces errors that have to be corrected. This will be addressed in later sections. The complete system topology for the simulation experiments is shown in Figure 7-3.

- The *Motion Controller* node is written in C++ and performs the high level instructions as well as the actual motion control.
- The Scene updater, written in Python, can dynamically add obstacles or target objects to the scene. Using a simple UI, an object can be moved to observe how the system responds.
- The vision pipeline is used to test the system with an actual physical input. Using methods developed for the initial pick and place setup, A number of functions were implemented to observe how the system would react to noisy measurements and data.

RVIZ (ROS Visualizer) is used to visualize the scene, consider Figure 7-4 for a screenshot of the testing setup.

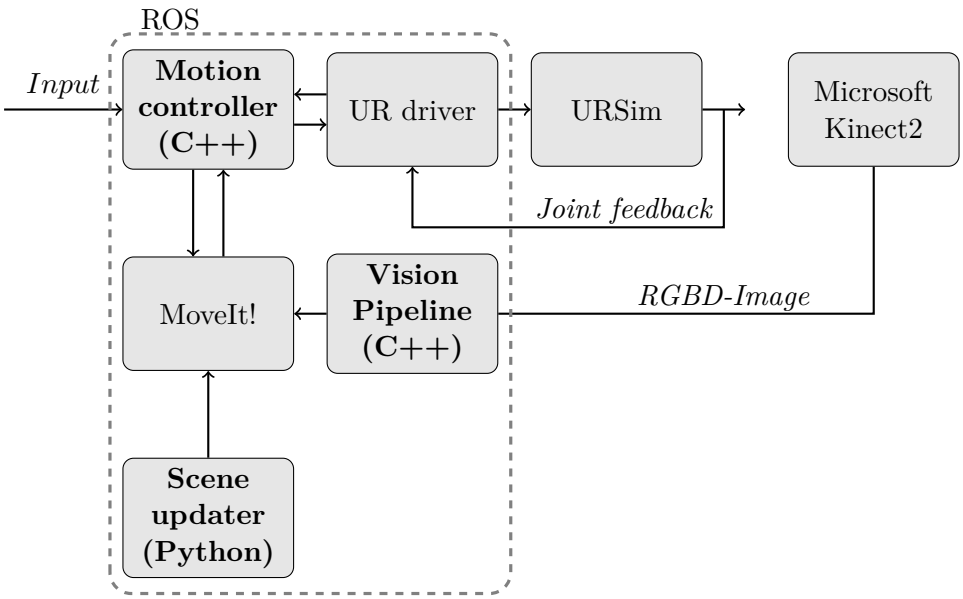


Figure 7-3: System topology for the experiments using a dynamic motion controller. Custom nodes are depicted in bold.

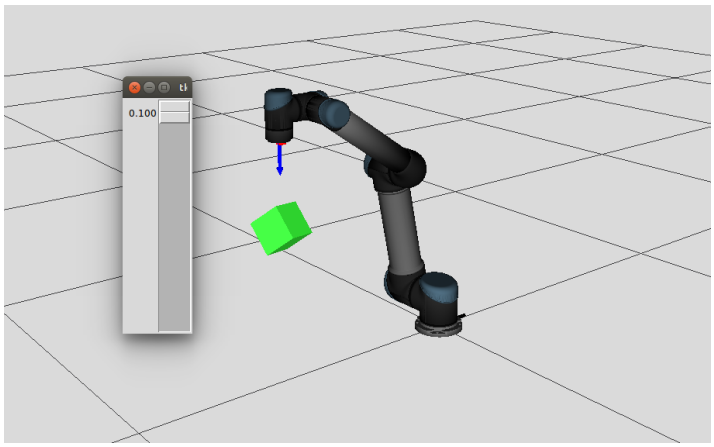


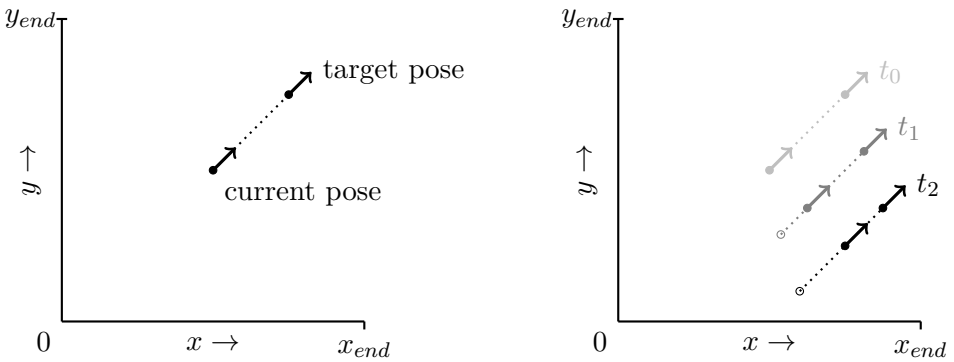
Figure 7-4: RVIZ view of the testing setup. An external obstacle that is to be tracking is depicted in green. The slider can be used to move the obstacle along a single axis. The blue arrow depicts the target pose of the UR5

These are the basic components for the experiments of next two chapters.and . Chapter 8 will consider the problem of performing a Cartesian approach when tracking a moving object. Afterwards, Chapter 9 will discuss dynamic obstacle avoidance when tracking a planned path.

Chapter 8

Pose tracking

If the target pose is in motion during a Cartesian approach, a large part of the motion will remain the same but super-positioned onto a moving frame. Consider Figure 8-1 for a 2D representation of what a moving grasp motion would entail.



(a) Position and grasp pose at $t_0 = 0$

(b) Motion states through timestates $t_0 < t_1 < t_2$

Figure 8-1: Execution of a grasp motion for a moving target pose. At $t_0 = 0$, the robot is in the pre-grasp pose. The dotted line denotes the Cartesian approach.

Using standard geometric transformations, it is trivial to compute where the end effector of the robot should be at what moment, even if the target pose is non-stationary. Then, the problem is not so much how to find this pose, as well as how to actually achieve a smooth motion that accurately tracks this pose *and* is able to adapt to any changes.

As discussed before, a moving target has a bigger impact on the implementation of Cartesian tracking than avoiding collisions. Section 8-1 will explore why this is the case and what strategies can be used for avoiding collision when performing a Cartesian approach. The next section, Section 8-2 will detail an experiment that uses Cartesian trajectories and cubic interpolation to achieve tracking. Section 8-3 will discuss the results of another method,

using on-line time parameterization techniques. Finally, Section 8-4 will contain a couple of concluding remarks regarding the topic of pose tracking.

8-1 Obstacle avoidance while tracking

As the tracking of a target is supposed to be designed for relatively small distances, dynamic obstacle avoidance was deemed to be less dominant for the Cartesian. Of course, this does not mean that collisions of any kind cannot happen. In order to devise an effective strategy of avoiding collisions, it is important to consider the context.

When tracking a certain pose, the goal of the end-effector of the UR5 is completely specified in all 6 dimensions. Recall that the UR5 is a *holonomic* robot, so the number of solutions to the IK problem are limited. What's more, the different solutions to a single pose are not *connected*. Consider Figure 8-2 for the 8 distinct solutions of an IK problem for the UR5.

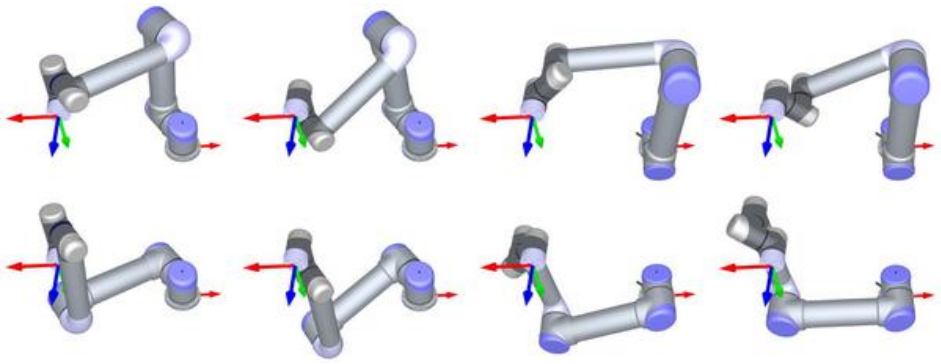


Figure 8-2: The 8 IK solutions to a single pose of the UR5[74]

If the UR5 is tracking a robot while close (in the configuration space) to one of these 8 configurations, it is not possible to quickly change to another of these configurations. Such a procedure, where the robot has to spin around to align itself back to where the end-effector started, is called a *configuration change*. Because this is so disruptive to the tracking motion, this means that for each target pose, there is effectively only *one* possible configuration. When that configuration is invalidated by a collision, the manipulator will have to stop actively tracking the target and devise an alternate strategy. One strategy would be something based on potential fields, but this has not been investigated any further.

For a 7 DoF robot arm, there would be an infinite set of configurations that would still have the targeted end-effector pose, so as long as the collision would not be at the end-effector, which would invalidate all target poses, there might be an alternate valid path without stopping tracking. Depending on the type of end-effector or gripper, it is possible to adjust the problem formulation such that the UR5 can indeed track an object even when obstacles are present. For instance, when grasping from above with a suction cup, you typically do not "care" what the rotation around the z-axis is. This effectively reduces the dimensionality of the target pose to 5 instead of 6. In this case, the UR5 has a redundant joint, which can be used to change between different configurations. A method that can be used for these problems is Descartes[75]. Descartes is a "Path Planner for under-defined Cartesian trajectories". Further

testing is needed to find whether Descartes is capable of dealing with these scenario's in real-time.

To limit the scope, the strategy that will be employed in these experiments regarding obstacle avoidance while tracking, is to stop when the controller knows it will run into an obstacle, be that self-collisions or external obstacles.

8-2 Computing Cartesian paths

The first approach to pose tracking attempts to use as much of the MoveIt! interfaces as possible. MoveIt! has a dedicated function to plan Cartesian paths that works by interpolating the poses between the start and the goal pose. IK is then used to find a valid configuration for each pose, and by specifying certain bounds on the IK solutions, it can be ensured that no configuration changes will happen. Then, time-parameterization is applied to the path so it can be executed.

One of the main objectives of this test is to examine the dynamic behavior when replacing the current trajectory 10 times a second. Instead of carefully checking each intermediate step, an optimistic approach is taken where the trajectory is sent to the controller without collision checking.

With the distance between the current pose and goal pose being small as it is, even a 2-waypoint trajectory has not been observed to produce paths that deviate from the expected path too much. This allows a new trajectory to be generated without any form of IK or collision checking. The waypoints are set as in Table 8-1. By adapting the starting waypoint to the current velocity, there will be no discontinuities in the velocity.

Waypoint number	Timestamp	Positions	Velocities
0	0s	Current Position	Current Velocity
1	0.5s	Target Position	0

Table 8-1: Waypoints of the naive tracking trajectory

Consider Figure 8-3 for a plot of the state evolution for a single joint receiving such a trajectory from standstill.

One disadvantage is that the controller does not perform any sort of constraint checking. This means that the timestamp of the 2$^{\text{nd}}$ waypoint, together with the maximum angle difference of the joints determine whether the joint limits will be satisfied. However, the *speedj* command allows setting a maximum acceleration and a velocity limiter can be used to limit the maximum velocity. Consider Figure 8-4, where the trajectory time is reduced to 200ms, causing the controller to run into both acceleration limits (set to 30 rad/s^2) and velocity limits (set to 3.3 rad/s).

A downside of both of these trajectories is that they are not jerk limited. Both at the start and endpoints of a trajectory and when hitting a joint limit, an large jerk changes the magnitude of the acceleration. Another problem with constraining the joint limits at such a low level is that a MoveIt! trajectory is parameterized to time, so when the actual manipulator starts

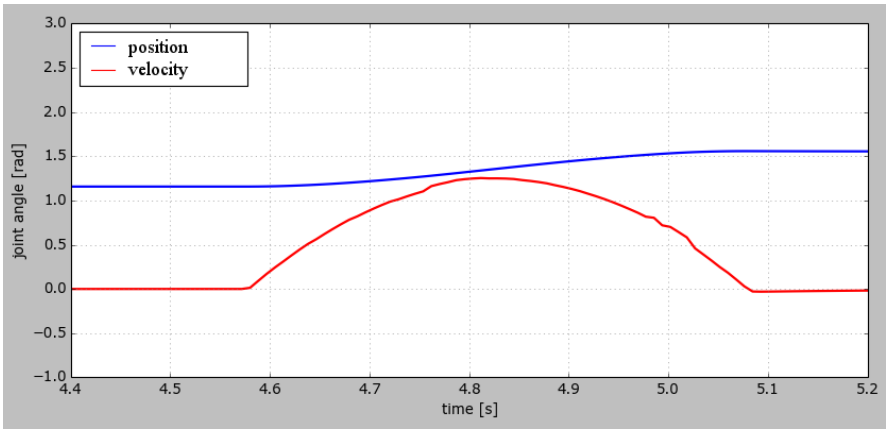


Figure 8-3: Single joint state evolution for a 2-waypoint trajectory with a duration of 500ms.

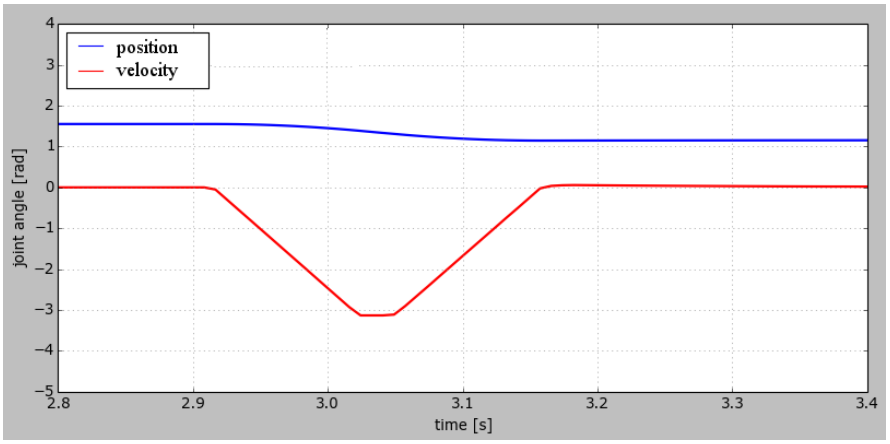


Figure 8-4: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits.

lagging behind, the trajectory interpolation routine might move to waypoint $n+1$ while the robot is not even close to waypoint n. When tracking a single waypoint, this leads to the manipulator not reaching the specified target. As a velocity setpoint is used, the setpoint at $t > t_{end}$ will be 0. This means that whenever the trajectory execution takes longer than expected, the controller will come to a halt before the targeted position is reached, as can be seen in Figure 8-5.

This can be solved by using the joint state position as an added feedback term, as done for Figure 8-6.

This causes a significant overshoot when posed with a such a large step. Better tuning is expected to improve the accuracy of the tracking. For the smoother motions that this controller is meant for, the current feedback gain works well, as can be seen in Figure 8-7, where a moving object is tracked.

In this example, the target object about 20cm in 1s. This leads to a fairly smooth motion. Again, large jerks can be observed when starting and stopping the trajectory, but with this is difficult to avoid.

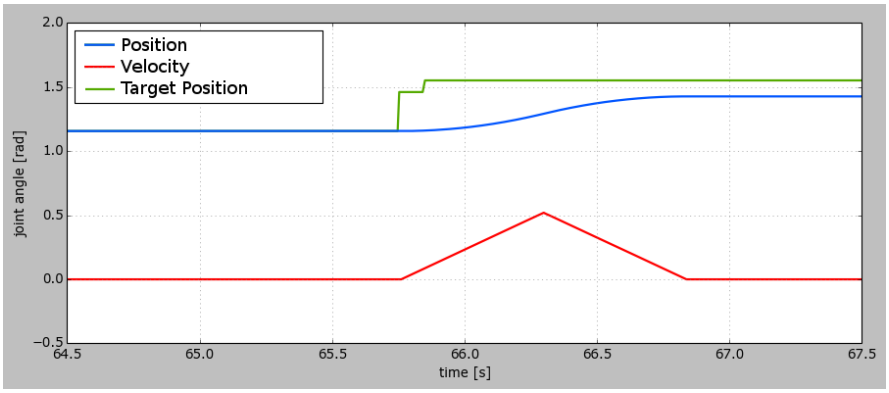


Figure 8-5: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits and not being able to finish the trajectory.

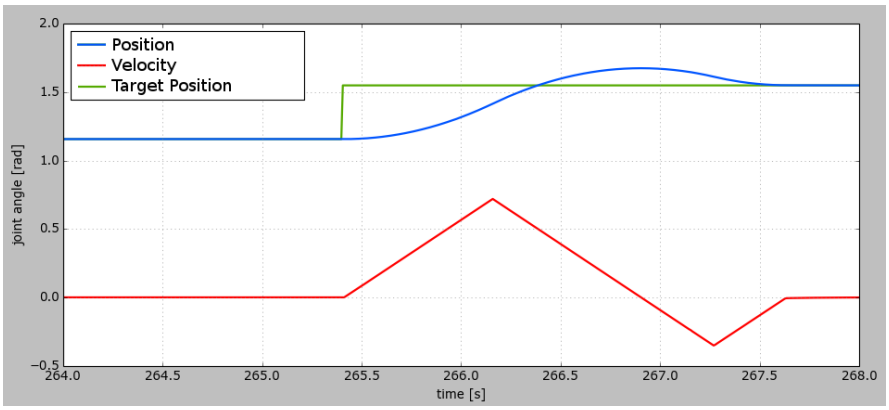


Figure 8-6: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits with joint state feedback.

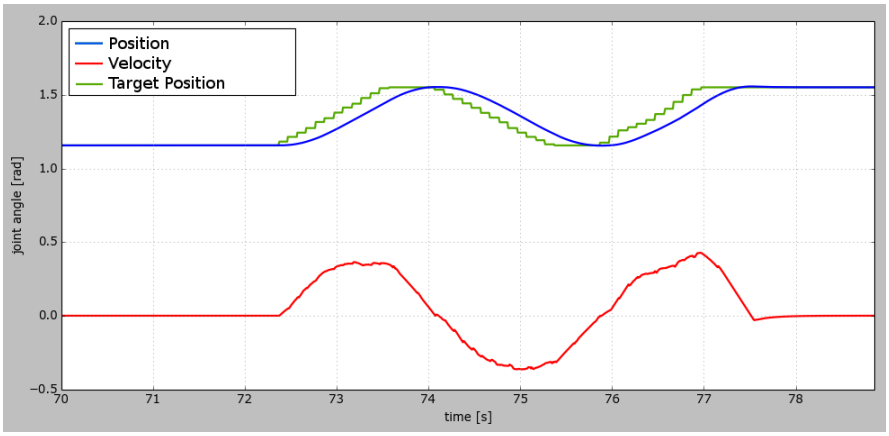


Figure 8-7: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits with joint state feedback.

Given the joint limits, this approach manages to efficiently track the target pose but two problems remain with this approach. Firstly, a small jerk can be observed on each target

change. This is not a problem in this scenario, but a relaxation of the assumptions might increase the time that is needed to find a new trajectory. This would make the merging of the current and the new target trajectory more difficult, possibly leading to unwanted jerks in the execution. An illustration of the effect can be seen in Figure 8-8.

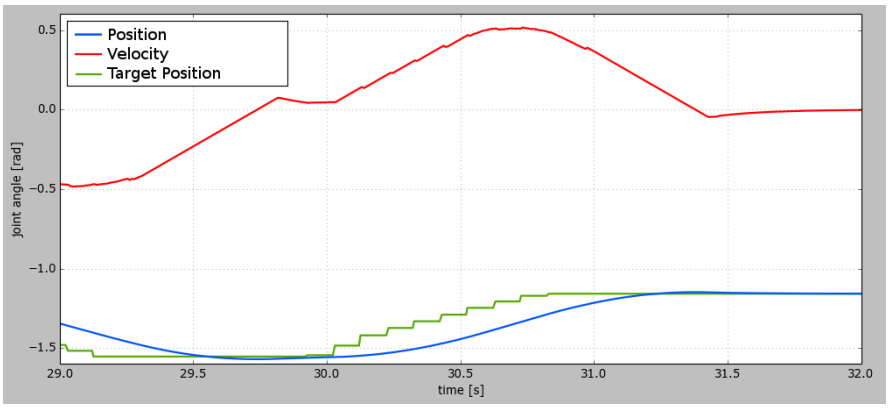


Figure 8-8: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits with joint state feedback.

8-3 On-line time parameterizaion

An alternative approach is to use on-line time parameterization. As discussed in Section 4-2, this can be used when the time parameterization function is both deterministic and continuous, and is capable of calculating a new setpoint within the interpolation period of the robot. By requiring the function to be both deterministic and continuous, the setpoint is guaranteed to be a smooth function in time. For the UR5, which is capable of accepting a new command at 125Hz, this means that a new target state has to be calculated every 8ms.

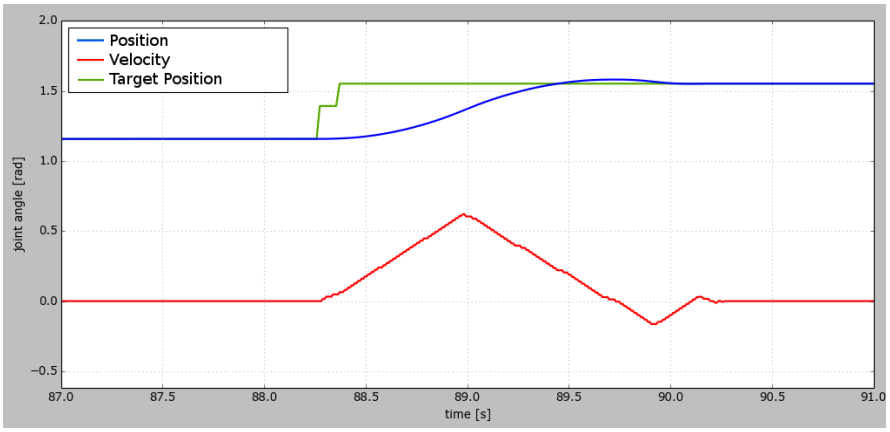


Figure 8-9

One of the few practically available on-line trajectory generation implementations that is guaranteed to be real time safe is the Reflexxes Library. Unfortunately, after being acquired

by Google, the type IV (jerk limited library) was made into a commercial product, but the type II (acceleration limited) library is open source and freely available. One limitation is that it only generates a trajectory until an arbitrary chosen state, rather than the complete trajectory. In order to use the library for executing a complete trajectory, the velocity of these target states can be set to an arbitrary value, as these are not calculated by the planner. Note that for the tracking of a target pose, this is not a problem. By setting the target velocities to 0, the robot will aim come to a full stop at each waypoint, after which a new trajectory will be calculated towards the next waypoint. To compare the Reflexxes based controller, consider the response to the same step as the Cartesian path controller was subjected to in Figure 8-9.

Here, the Reflexxes based controller performs better than the Cartesian path controller. Reflexxes generates bang-bang trajectories, and is time-synchronized for all the joints. In this graph, the pictured base joint has an acceleration of 1 rad/s^2, meaning that this is the joint that has to rotate the furthest in this trajectory and is not scaled. Other joints will have a similar trajectory, but with a smaller magnitude.

Next, consider a similar tracking behavior of the Reflexxes controller in Figure 8-10. Clearly,

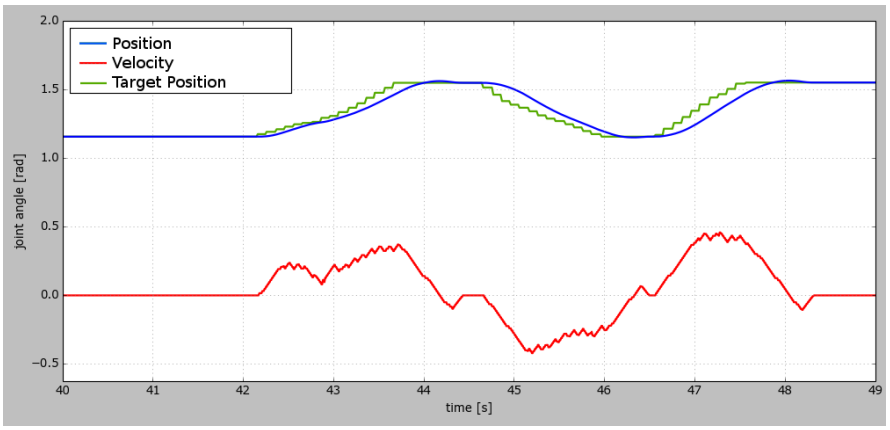


Figure 8-10: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits with joint state feedback.

the impact of a changing setpoint is smaller for the Reflexxes controller. In addition, the fact that the path is only stored as geometric waypoints, means that there is no notion of whenever a path *should* be finished, so lagging behind the target trajectory is not an issue using this controller. However, as Reflexxes is always trying to find the optimal trajectory and is not jerk limited, the motion is not as smooth as it could be. Unlike the previous controller, the Reflexxes based controller does not suffer from jumps in the execution when a new setpoint is received, as can be observed in Figure 8-11.

8-4 Conclusions

In this chapter, two different methods of pose tracking were introduced. Both have their respective advantages and disadvantages.

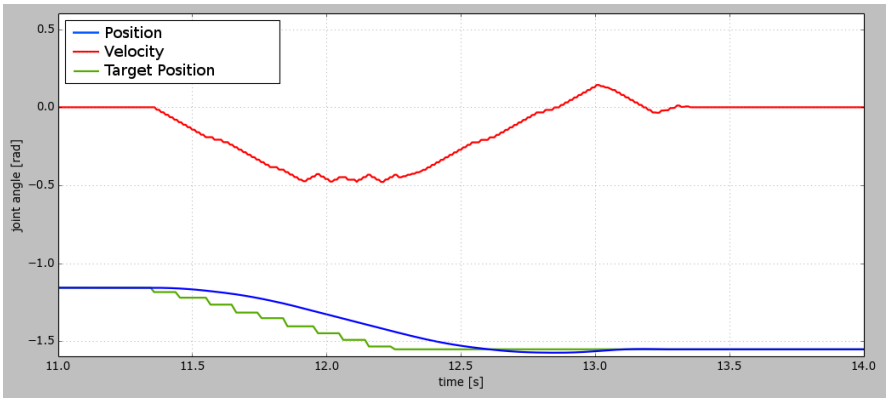


Figure 8-11: Base joint state evolution for a 2-waypoint trajectory running into acceleration and velocity limits with joint state feedback.

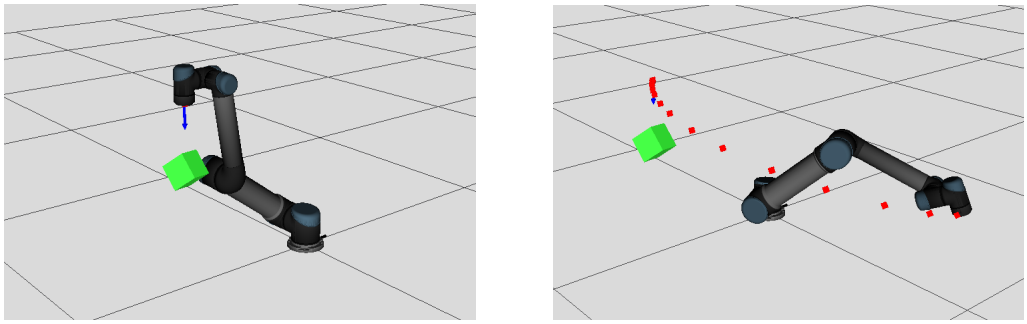
For the off-line controller, the simplicity of the approach is a definite advantage. Less computational steps are taken to formulate a control action, and it integrates nicely with the available MoveIt! infrastructure. If the UR5 controller would support MoveIt!'s trajectory replacement function, it could theoretically be integrated without any low level control to the UR5. The somewhat fuzzy time parameterization is a problem, but it is unclear to figure out how much that would lead to problems for more intricate scenario's. The off-line controller is capable of producing fairly smooth trajectories for small changes, as it will avoid running into the joint limits. This does mean that the tracking will be a little slower.

For the on-line controller, a lot more computation is needed. By numerical integration the controller does manage to find a time-optimal trajectory towards the target pose. Unfortunately, because of the bang-bang nature of the controller, even very short trajectories will be jerky when not using a jerk limited trajectory generator. This does mean that the tracking speed is superior to the speed of the off-line controller. An additional benefit is that the velocity can also be specified for the target state, this means that a predictive controller that fits a second or third order motion prediction to the target pose will be able to track much more accurately than the off-line controller. A disadvantage is the path tracking might not be as accurate, but for tracking this is only a minor problem. Integrating the on-line controller with a global path execution controller might prove to be more difficult than with the off-line controller.

Chapter 9

Dynamic Path Execution

As discussed in Section 7, dynamic global path execution is hindered more by dynamic obstacles than a moving pose. As the pose to be tracked is generally assumed to not be moving around a lot, any change could just be added to the back of the global motion. Observe Figure 9-1 for an example of the split between the global path execution and the pose tracking.



(a) Global path **(b)** Tracking

Figure 9-1: Example test trajectory, (a) depicts the reception of a new pose target, denoted by the blue arrow. A global path is found and sent to the trajectory controller. The red markers indicate the waypoints of the global trajectory. When the manipulator reaches the target pose in (b) the control mode is switched to Cartesian tracking.

For obstacle avoidance, the requirements on the system are a little different than for pose tracking. Namely, the execution speed is of a lesser importance, and accurately tracking the planned path *is* important. In Section 9-1, the off-line time-parameterization will be adjusted on-line, so that the path traversal slows down when moving towards an obstacle. In **??**, this method is extend to perform a plan a new path whenever the obstacle obstructs the path for a specified amount of time. Finally, the controller considerations that are relevant for path execution are discussed in 9-3.

9-1 On-line time scaling

By time stretching the execution of a given trajectory, the manipulator can be constrained to only occupy states that are part of the original trajectory. In a lot of cases, dynamic obstacles will only invalidate parts of the trajectory for a short while, so if the controller can slow down when moving towards a previously unseen obstacle, there is a good chance that by the time he reaches the invalidated state, the obstacle has moved out of the way. Consider Figure 9-2 as an example of how the time stretching has been implemented.

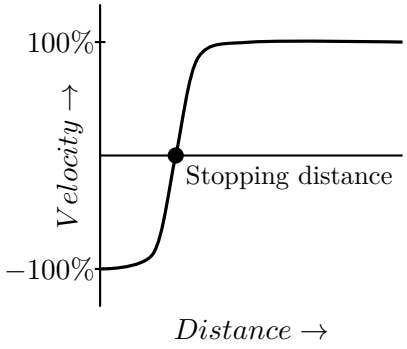
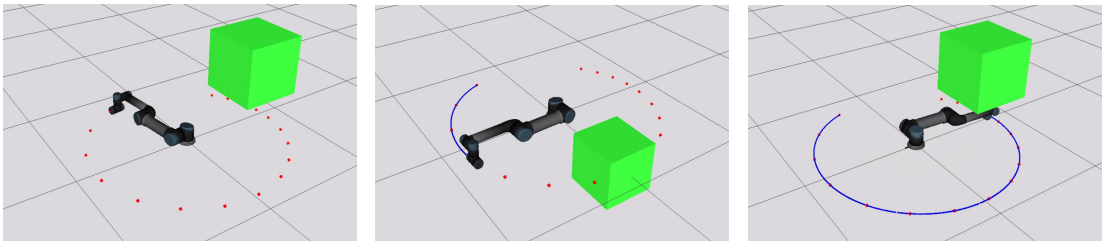


Figure 9-2: Obstacle distance to path traversal velocity mapping

The system was tested in simulation and found to perform well. See Figure **??** for such a simulation.



(a) Initial plan

(b) Standstill at safe distance

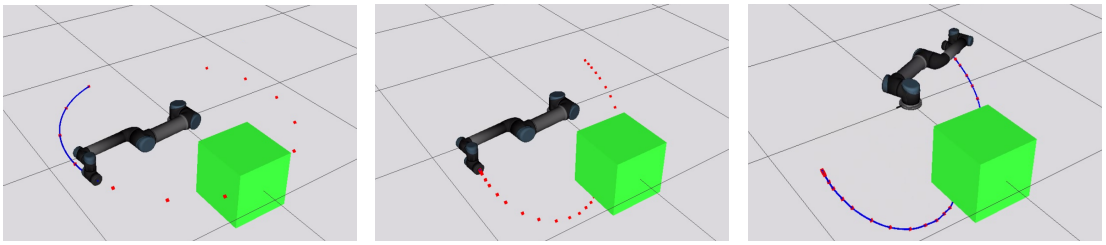
(c) Continue with original plan

Figure 9-3: time-scaling in action. The trajectory waypoints are depicted in red and the followed path in blue. (a) depicts the reception of an initial trajectory. When the green box drops into the trajectory, the manipulator slows down to come to a full stop at (b), and when it removes itself, the trajectory is finished as shown in (c).

A rate limiter was implemented to ensure that whenever an obstacle would clear the path, the traversal velocity would only slowly increase back to 100%. With the time scaling also capable of dealing with negative velocities, the manipulator is "pushed back" into the trajectory if an obstacle comes closer.

While this works for very simple cases, like a person walking through the workspace and removing itself again, it is very limited. In the next section, the system is extended with a planner to be able to avoid getting stuck waiting for an obstacle to remove itself.

9-2 Replanning



(a) Standstill at safe distance

(b) Replan

(c) Arrived at goal

Figure 9-4: Replanning in action. The trajectory waypoints are depicted in red and the followed path in blue. When the green box drops into the trajectory, the manipulator slows down to come to a full stop at (a). at (b), a new trajectory is planned and the total execution is finished at (c)

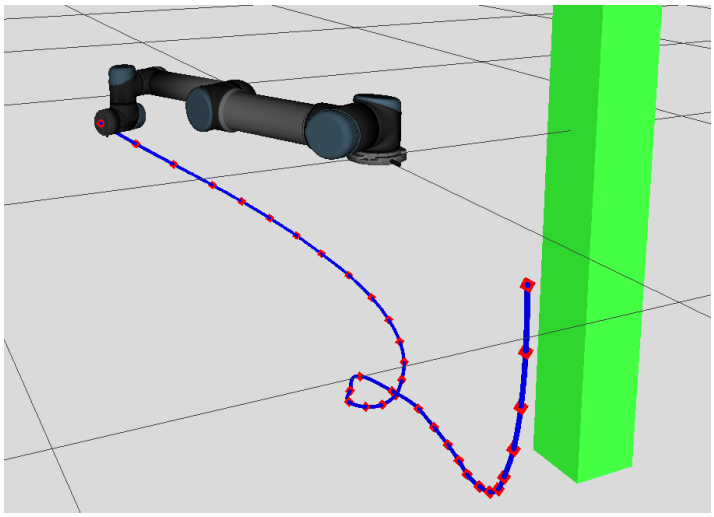
Very similar to the method described in [67], a planner has been integrated with time scaling. Instead of immediately finding a new plan when an obstacle was observed, the planner would only be invoked after a certain amount of time loss. So, if the obstacle would be much further in the trajectory, no replanning query will be invoked until it is evident that the obstacle poses an actual problem. As a consequence of being at standstill when finding a new plan, no shortcutting or trajectory merging is required. In addition, using a previously constructed tree or roadmap for a new planning query is a non-trivial affair in MoveIt! and was not used for these simulations.

While this is by no means a complete solution to the problem, it does demonstrate a method that is effective and simple in avoiding collisions, and only takes action when it is really needed. Consider Figure 9-4 for an illustration of the working principle. Although not a general solution by any means, the system performs well. Because of the stopping, the behavior of the robot can hardly be considered dynamic. In fact, the path planning will remain to be performed in a static environment.

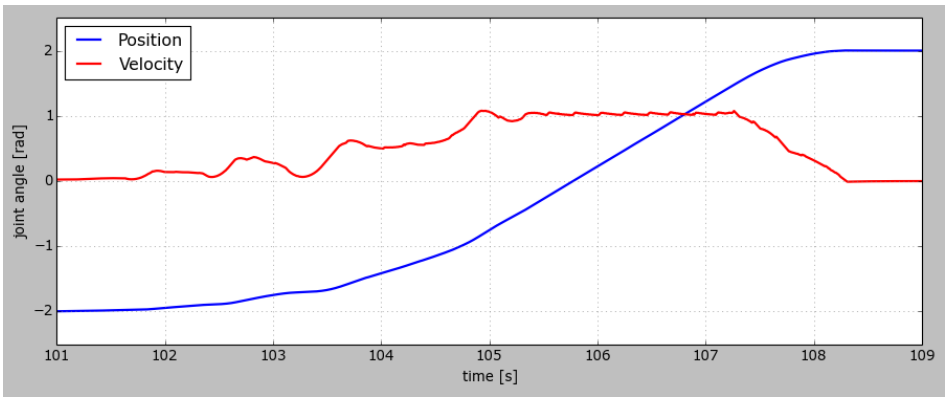
9-3 Path execution controller comparison

In order to directly apply changes to the current trajectory, an identical set of controllers was used as in Chapter 8. It turns out doing off-line trajectory generation is very difficult to accurately track using the velocity controller and the imposed joint limits. By slightly relaxing the lower level joint limits, the trajectory could be seen to occasionally violate them, but was in all able to follow the specified path quite accurately and without too many jerks, as demonstrated in Figure 9-5.

As with position tracking, applying on-line trajectory generation has some distinct advantages. Being able to adjust any waypoint further up the path without affecting time parameterization is such an advantage. Unfortunately, no real-time safe time parameterization methods seem to currently exist. Reflexxes, used in Chapter 8 is only able to generate a trajectory to a target arbitrary state. By setting the state velocity at 0 for each waypoint, it is possible to execute



(a) Executed path. Waypoints in red and executed path in blue markers.



(b) Base joint evolution

Figure 9-5: Path tracking of the MoveIt! based velocity controller. The controller manages to track the path quite accurately using mostly smooth motions.

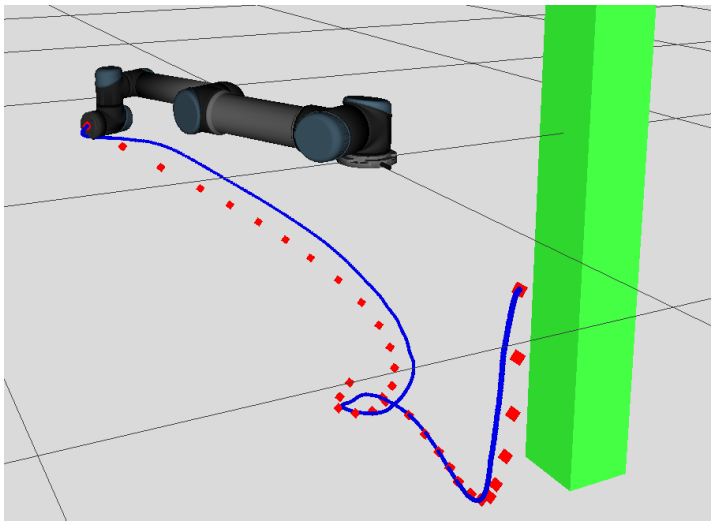
the trajectory, but the manipulator stops at each waypoint. In order to be able to achieve some sort of smooth execution using Reflexxes, the following algorithm was implemented:

Define W_p as a vector of n waypoints, with the final waypoint at $W_p(0)$ and the first at $W_p(n-1)$. Then, the control *next* waypoint is always $W_p(n-1)$. The current state is defined as θ_c and the target state as θ_c.

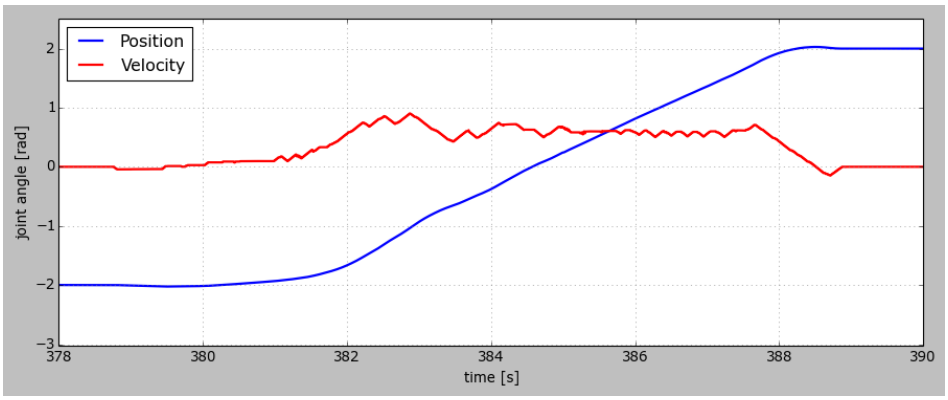
Then, $\theta_c = W_p(n-1)$, but $\dot{\theta}_t$ is unknown. Two options are defined:

- the motion between θ_c and $W_p(n-1)$ is in the **same** direction as the subsequent motion, between $W_p(n-1)$ and $W_p(n-2)$. In this case the target velocity is set to 0.3 times the joint limit in the corresponding direction.
- the motion between θ_c and $W_p(n-1)$ is in the **opposite** direction as the following motion, between $W_p(n-1)$ and $W_p(n-2)$. In this case the target velocity is set to 0.

Finally, whenever the manipulator is sufficiently close to the waypoint, the next waypoint is popped off the vector and tracking to the next one begins. As expected however, the path accuracy is not very good, as can be seen in Figure 9-6.



(a) Executed path. Waypoints in red and executed path in blue markers.



(b) Base joint evolution

Figure 9-6: Path tracking of the Reflexxes based controller. The tracking accuracy is not very good and the execution time is also rather long.

This is largely related to the heuristics that were used in order to decide when to move on the the next waypoint. Further research will have to determine whether this can be solved. In theory, on-line time parameterization provides for a motion controller that is very flexible, as the waypoints can be adjusted even when the path is being executed. At this point however, it is clear that a more standard controller topology offers far superior performance.

Chapter 10

Closing remarks

This final chapter will conclude the thesis and specify to what degree the original objectives have been met. The recommendations section attempts to formulate a clear focus for any future work.

10-1 Conclusions

This thesis presents the results and implementation details of several different methods that can be used to dynamically control a 6 DoF manipulator. Recognizing early on that formulating a method that was able to achieve all of the research goals was too ambitious for this scope, the work has focused on analyzing the merits and shortcomings of several approaches in order to better understand the challenges involved. The hope is that using this work, a more structured approach can be taken to analyze and tackle the problems related to operating a robot arm in changing conditions.

By combining the Cartesian path based pose tracking method with the dynamic replanning, a system has been created that is able to effectively and quickly respond to changes in its environment. Judging from the performance of the two different controllers, the off-line time parametrization controller has preference for such a system. However, being based on a random sampling based planner, it is unclear whether the paths that such a system will take can really be considered "smooth". In addition, further testing would be needed to accurately specify the performance of the system, which sadly was not possible in the allocated time. Hopefully, this work will be able to serve as a stepping stone for further research.

The main contributions of this thesis are:

- **A clear problem formulation** By clearly defining the problems related to dynamic motion execution, it is easier to get a clear overview of the set of methods that can possibly lead to improved performance.

- **A structured tuning method for configuration space planners** In order to make a meaningful assessment about the performance of a given planning algorithm, you need to be sure that the configuration of said algorithm is optimal for the target problem. In this work, an method has been developed that allows a user to tune any kind of planner without requiring in depth knowledge of the inner mechanics of said planner. The method was found to offer greatly improved performance over the default configuration for several different planners. A paper was submitted to IROS 2016 with hopes of further developing such methods.

- **An extensive benchmark** While several benchmarking results can be found in literature, they are often lacking information that allows you to predict performance on a practical system. A detailed analysis into the optimization properties of optimal planners for a practical scenario provides some useful information for anyone considering using optimal planners for practical applications.

- **A system topology capable of reacting to a changing environment** As described above, a system based on dynamic time-scaling, replanning and Cartesian path based tracking is capable of effectively reacting to changes in its environment, including tracking a moving target.

- **A controller topology based on on-line trajectory generation** Even though the tracking performance of the controller was not very good, it has been shown that a controller based on on-line trajectory generation is feasible and even outperforms the standard controller for tracking purposes.

10-2 Recommendations

A number of observations have been made which warrant further research. First of all, the proposed system would need to be tested much more thoroughly in order to assess its performance. A number of tests on an actual UR5 have been carried out, but this could use a more rigorous approach. Furthermore, the following topics are also regarded to deserve further research.

- **Further testing of the proposed system topology** Several simulation experiments have been carried out, but a better specification of the performance is definitely desirable. A small number of tests to verify the operation on the real UR5 have also been performed, but this has been very limited.

- **Optimization based path deformation** It is unfortunate that TrajOpt, as one of the more promising optimization based methods, it is not compatible with the widely adopted MoveIt! infrastructure. One advantage is that optimization based planners locally deform the path during planning, leading to the possibility of using an optimization based planner as a path deformation algorithm. Combined with the fact that these planners are also capable of finding a collision free path in difficult environment, it could potentially be possible to use a single algorithm for both planning and collision avoidance.

- **On-line time parametrization based path execution controller** As explained in the previous section, these methods are only scarcely documented in literature, despite having several desirable properties.
- **Further development of the tuning method** As the results achieved by the tuning method are very promising, further developing these methods and possibly releasing them as an open-source tool could provide many users with better functioning path planning algorithms.

Appendix A

IROS 2016 Paper

This paper was written in January and February of 2016 for submission to the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016) which will be held in Daejeon, Korea, October 9-14, 2016. IROS is one of the leading conferences in the field of Robotics, and annually brings together an international community of researchers, educationers and practitioners to explore the frontier of science and technology in intelligent robots and systems.

The theme for IROS 2016 edition is "Road to companionship with intelligent robots in everyday life and workspaces". We believe that a structured approach to tuning configuration space planners can aid in the adoption of planning methods in practical scenario's. When using robotic manipulators in collaboration with human workers, the current industry standard method of letting a robot perform a predefined motion will no longer be sufficient. Configuration space planning is one of the most promising methods to deal with changing environments, and we hope to stimulate further research into practical applications of these tools.

Automated Tuning and Configuration of Path Planning Algorithms

Ruben Burger[1], Mukunda Bharatheesha[1], Marc van Eert[2] and Robert Babuška[1]

Abstract— **Over the past decades, many different path planning algorithms have been designed for a variety of planning problems in robotics. Each algorithm is capable of solving the problem for which it was proposed, but it is often unclear how it generalizes to problems with slightly different characteristics. Tuning the parameters of these path planners so that they can achieve desired performance in practical scenario's has also been largely unexplored. At the same time, there is a rising interest in the planning and robotics communities regarding the application of the theoretically developed and simulation-tested planning algorithms to real world applications. As it stands today, this requires a well-trained expert with appropriate knowledge of planning algorithms.**

In this paper, we propose the use of Sequential Model-based Algorithm Configuration (SMAC) tools to address these concerns. We present a complete methodology that enables end-users to tune different planning algorithms to their specific problems without the need of in-depth knowledge of the algorithm itself. We compare five state-of-the-art planners on three typical industrial pick-and-place tasks.

I. INTRODUCTION

Ever since the early 1950's, the PID controller has served as a simple and a highly effective bridge from the academia to industry in terms of achieving a desired control performance without the need of expert knowledge. From the end-user perspective, the concept of PID practically reduced the complexity of a process control problem into turning three knobs. Although this seems like a very gross generalization, PID enabled a much wider group of users to apply control systems to a large variety of problems enabling a widespread use of control systems. Current (industrial) robotic manipulators are able accomplish a wide range of tasks. Typically, these tasks involve some form of (pick and place) motion planning. One of the fundamental issues that is currently holding back a widespread application of planning algorithms to accomplish tasks with robotic manipulators is the required background knowledge to work with these algorithms. Currently, a method is missing that would simplify the technical complexities of tuning a planning algorithm, like PID does for a control system.

Recently there has been a growing interest in the use of robotics software. With the increasing popularity of tools like ROS[1], MoveIt[2] and OpenRave[3], even hobbyists have now ventured into robotics. Unlike research, most of these users are primarily interested in leveraging the current state of robotics to solve practical problems. This new class of users would benefit greatly from a more structured and pragmatic approach to applying motion planning to real world problems.

In contrast, the development of novel planning methods with high theoretical merit makes up a large body of research within the motion planning community. Many of these methods are designed to solve a certain set of problems that have some specific characteristic. Consider Transition-based RRT (T-RRT) [4], which combines the exploration strength of RRTs with cost-map methods in order to guide the algorithms to paths that are low cost according to a specified cost metric. T-RRT manages to efficiently solve the posed problem, but how it translates to a different domain is unclear and difficult to estimate without a significant effort. A second example can be found in the set of informed planners like Informed-RRT and BIT* [5][6]. These planners work very well for scenario's in which the obstacles takes a certain shape in the configuration space, but it is unclear how they can be leveraged for other classes of problems. They also have a number of configurable parameters that are difficult to understand without significant background knowledge. Another set of planners that uses heuristics to guide the search are Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) methods[7]. These planners show good practical performance, but they use a grid discretization that makes it difficult to understand how the planning performance is affected by the configuration of the planning algorithm.

When presented with the problem of selecting a suitable planner for a given real-world problem, one faces a myriad of options. With several different available planning libraries and approaches like OMPL[8], SBPL[9] , CHOMP[10] and STOMP[11], it requires significant knowledge to determine which planning algorithms are suitable for each particular application. Even more involved than planner selection is planner configuration and tuning. The number of parameters ranges for different planning algorithms from just one for RRTConnect to 12 for RRT*, all of which may or may not have a significant impact on the performance. Especially the frequent use of heuristics make it very difficult to predict the behavior of an algorithm as they often interact with each other in unexpected ways. This makes manual configuration and tuning a tedious task. As most of these planners have been designed with a specific type of problem in mind, there is no guarantee that the default configuration will work well for a different type of problem. In the case of OMPL, some of the parameters are calculated using the characteristics of the environment. While this is indeed beneficial, a larger improvement can be expected with a more rigorous tuning approach.

[1]Authors are with the Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands R.B.Burger@student.tudelft.nl, {M.Bharatheesha,R.Babuska}@tudelft.nl [2]Applied Scientist with Technolution BV, 2803 VW Gouda, The Netherlands marc.van.eert@technolution.nl

In order to effectively apply motion planning techniques developed in research laboratories to factory floors, a more pragmatic method of selecting and tuning planning algorithms is needed. An end-user is typically only interested in the performance of the system for a single scenario. Ideally, it should be possible for an end-user to provide a geometric description of the robotic manipulator and the scene, along with a set of typical problems, and an automated tuning algorithm provides an optimal configuration for each planner considered.

II. RELATED RESEARCH

A general approach to algorithm tuning can be found in automatic algorithm configuration methods. Closely related to optimization methods, algorithm configuration methods have been used since the 1990's. These methods repeatedly query a target algorithm to solve a defined problem using different configurations, searching for the best performing configuration. Early approaches, such as ParamILS[12] and F-RACE[13], used a local search in order to select a new promising configuration, but recently *model-based approaches* such as SMAC[14] have emerged. SMAC continuously updates a regression model of the problem in order to better predict well-performing configurations. Applied to expensive black-box functions, SMAC has been shown to deliver promising performance on a diverse set of problems[15].

Automatic algorithm configuration methods have been applied to a wide variety of problems, most notably to the tuning of solvers for computationally hard problems. These solvers, often based on local search or tree search, often have many tuning parameters. As an example; the mixed integer programming solver IBM CPLEX has 76 parameters relating to its search strategy [14]. Tuning such a solver by hand would likely lead to a sub optimal solution.

Closer to our application, automatic configuration methods have also been applied to classical planning problems. One approach that is similar to our method can be found in [16], where SMAC was used to automatically tune the Fast Downward planning system. In [17], ParamILS was used to tune the topology, transition rules and parameter values of control software for a robot swarm. Another approach where automatic configuration methods are proposed can be found in [18]. Here, constrained Bayesian optimization is proposed as an optimization method that can be used to tune certain target parameters. Using constraints on the optimization, it is ensured that the tested configuration does not lead to safety-critical system failures. In [19] a model-free optimization is used to find optimal PID parameters.

Configuration and tuning of path planning algorithms in particular seems to be a less highlighted area of research. In [20], an infrastructure that can be used for benchmarking different configurations was introduced, but the problem of how to select competing configurations for each planner is left unexplored.

In this paper, we propose a planner tuning method based on Sequential Model-Based Algorithm Configuration, or SMAC. This method enables non-expert users to tune any path planner to their specific planning problem. By providing a model of the environment and problem complexity of the robot, SMAC is able to configure and tune any planning algorithm to suit the users needs. Tests have shown that a planning algorithm tuned by SMAC is able to achieve significantly better performance.

III. PROBLEM STATEMENT

Let $X \in \mathbb{R}^n$ be the complete planning space of target scenario. Denote by $X_{\text{obs}} \subset X$ the obstacle space, which consists of all invalid (collision) states of the system.

Let $X_{\text{free}} \subset X$, the complement of X_{obs}, be the set of all valid configurations in the planning space.

Define $\Theta_{\text{start}} \subseteq X_{\text{free}}$ and $\Theta_{\text{goal}} \subseteq X_{\text{free}}$ to be a set of states that are representative of the problem complexity. A planning problem is then defined by the start and goal state as follows:

$$\psi = \{\theta_{start} \in \Theta_{\text{start}}, \theta_{goal} \in \Theta_{\text{goal}}\}$$

The set of all possible problems derived from θ_{start} and θ_{goal} is denoted by Ψ. Denote by A a target planning algorithm with n configurable parameters. A configuration of A is then denoted by

$$\Phi = \{\phi_0, \ldots, \phi_{n-1}\}$$

with

$$\phi = \{\phi_{\text{name}}, \phi_{\text{value}}\}$$

A single problem *query*, where algorithm A, with configuration Φ, is required to solve problem ψ while satisfying the constraints from the environment X can then be written as:

$$A(X, \Phi, \psi)$$

Finally, a cost function is defined that assigns a quantitative metric to each query:

$$cost(A(X, \Phi, \psi))$$

The problem is to find a configuration that minimizes the cost function over the complete problem set:

$$\min_{\Phi} cost(A(X, \Phi, \Psi))$$

If the problem set Ψ is a good representation of the planning problems that the algorithm will face, the configuration Φ that minimizes the cost function can be expected to be well performing configuration for the target scenario.

IV. METHOD

The main mechanics of SMAC are as follows: Consider an algorithm A, which is used to solve a certain problem instance I, and judged using a cost function c. The configuration tool starts with finding the performance of the default configuration and then repeatedly attempts new configurations in order to find the configuration with the lowest associated cost. Often, multiple problem instances I are used to ensure that the tested configuration is not subject to over fitting.

One very useful feature of SMAC is the support of categorical parameters as well as numerical parameters. Using these categorical parameters, the planning algorithms can be tested with different configurations as well as different parameter values. In addition, parameters can be made to depend on the setting of different variables. Many planning algorithms have parameters specifically related to a certain heuristic. Having these depend on the variables that define the configuration allows these parameters to be disregarded when the relevant heuristic is disabled.

A. Formulating the problem instance I

In order for SMAC to optimize towards a meaningful configuration, care should be taken in how to formulate the problem instance I as a function of X and Ψ. One approach could be to treat each separate problem ψ as a problem instance. However, with the high variance that many planning algorithms are subject to, finding meaningful configurations proved problematic for SMAC, even when SMAC was aware that the target algorithm was non-deterministic.

Alternatively, better results were obtained by splitting Ψ into a tuning and a validation set. Instead of running each problem query separately, SMAC then runs *each* problem whenever it tests a new configuration. In addition, the variance was decreased by iterating each problem query several times. In choosing the number of iterations, a trade-off has to be made between decreasing the variance of the planning algorithm and allowing SMAC to quickly try new scenario's.

Using a single problem instance I, SMAC runs the risk of tuning the algorithm to work only on that specific instance. However, the diversity of the instance was deemed sufficiently high as no problems of this kind were observed in testing.

B. Formulating the cost function c

The cost function c is another key element in our method. The goal of the cost function is to transform the output of a single problem query into a qualitative measure that can be minimized by SMAC. There are several options that can be used to define such a cost function. Common cost functions include algorithm computation time, percentage of solved queries, path length, path smoothness or mechanical work. In many practical scenarios, the algorithm runtime and percentage of solved queries take precedence over other cost metrics.

In this light, the most logical cost function was found to be the algorithm computation time. One way to prioritize configurations that solve a high percentage of queries within a set time is to adjust the allowed planning time. Whenever a planning algorithm exceeds the specified planning time, it reports a failure to SMAC. In order for SMAC to be able to gather sufficient information regarding the performance of a configuration, it is important that the allowed planning time is not set too close to the average algorithm runtime.

The complete method is shown in Algorithm 1.

Algorithm 1 SMAC Tuning

Input : Algorithm A, Problemset Ψ with N problems, Environment X, Iterations n, default configuration Φ_0
Output : Incumbent configuration Φ_{inc}

$\Phi = \Phi_0$
best $= \infty$
while *SMAC.Runtime* $<$ *AllowedRuntime* **do**
  result $= 0$
  for $i = 0, \ldots, N-1$ **do**
    $\psi = \Psi(i)$
    for $0, \ldots, n$ **do**
      Timer.start()
      $A(X, \Phi, \psi)$
      result += Timer.stop()
  if *result* $<$ *best* **then**
    $\Phi_{inc} = \Phi$
    best $=$ *result*
  SMAC.UpdateModel(*result*, Φ)
  $\Phi =$ SMAC.GetNewConfig()
return ***Incumbent***

V. RESULTS

In order to validate our tuning method it was tested on 3 different environments with two different robots: two environments for the 6-DoF Universal-Robots UR5, and one for the 7-DoF KUKA LBR iiwa 7 R800. The following planners were selected to be tuned:

- **KPIECE** This algorithm uses several heuristics in order to guide the search. These heuristics make it hard to predict its performance and make it an ideal candidate for automated configuration. One configuration parameter that we do not consider for tuning is the *projection evaluator*. Though it undoubtedly has an impact on the performance, we omit its tuning in this work. We elaborate further in the Discussion section.
- **BKPIECE** For much of the same reasons, the bi-directional variant of KPIECE will also be considered. Again, the projection evaluator is not included in the configuration tuning.
- **BIT** While BIT* is actually an optimal planner, it can easily be configured to return on the first solution. With several parameters, BIT is another interesting candidate for using the SMAC tools for tuning them to a given problem requirement.
- **RRTConnect** As RRTConnect only has a single parameter, it is not expected to gain much from tuning. However, it is worthwhile including RRTConnect as it is a widely adopted planner in more practical applications, known for short computation times and high solving percentages.
- **BiTRRT** Although very similar to RRTConnect, it is interesting to see whether the extra parameters that can be tuned for BiTRRT have a greater impact on its performance. For BiTRRT, an optimization objective can be set. As with the projection evaluator for the KPIECE

methods, this was not considered for the tuning.

The experiments were conducted on a PC with a Intel i5-3470 CPU at 3.20GHz and 8GB of RAM running Ubuntu 14.04.3. To model the environment ROS Indigo was used in combination with Moveit. The planners that have been tested are part of the OMPL planning library.

For these experiments, Ψ consisted of 20 problems that were each iterated 5 times for a total of 100 queries. For the simple problems, this offered sufficient performance. For more difficult problems, the number of iterations were decreased to allow SMAC quicker testing of each configuration. The planning time was selected to be 2 seconds and SMAC was allowed 30 minutes to find the best configuration.

A. *UR5 simple pick-and-place problem*

The first UR5 problem represents a pick-and-place scenario where the robot is to rearrange objects on a table. A wall obstructs a portion of the workspace and a pillar on the opposite side of the table is just within reach. At the selected start and goal states, the end-effector is in close proximity and perpendicular to the table surface, ready to either pick or place an object. Figure 1 shows a photo of the environment of the UR5. This can be considered a relatively easy problem

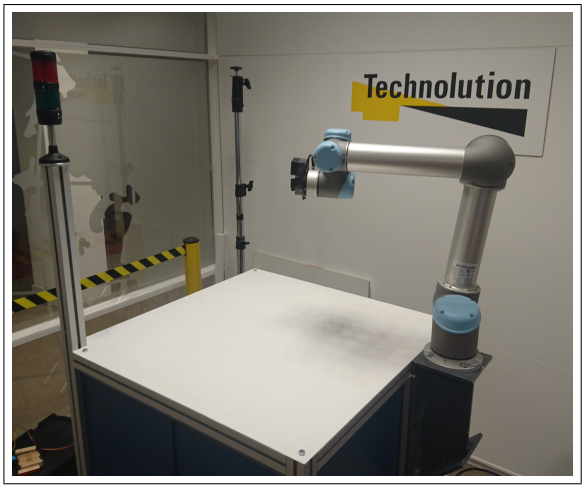


Fig. 1. Photo of the first target problem environment for the UR5 manipulator

for the planning algorithms, as the movement of the UR5 is mostly unobstructed. This allowed for SMAC to make several hundred calls to the planning algorithm, especially when tuning RRTConnect and BiTRRT.

Table I shows the averaged total results for the complete testing set. The path length is represented as the average 2-norm in the configuration space of the manipulator and is included to serve as a indicator for the planning algorithms ability to find short paths.

As expected, RRTConnect and BiTRRT have not improved much. For KPIECE, BKPIECE and BIT, the results are more striking. With computation times significantly improved, without finding longer paths, it is safe to say that the tuned configuration shows better performance than the default. For

TABLE I

SMAC RESULTS FOR THE FIRST UR5 PROBLEM

PLANNING TIMEOUT 1S

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	36.3	100	7.3
RRTConnect - SMAC	36.0	100	7.2
BiTRRT	46.8	100	7.1
BiTRRT - SMAC	45.8	99	7.2
BKPIECE	254	99	7.8
BKPIECE - SMAC	108	99	7.4
KPIECE	95.7	100	7.8
KPIECE - SMAC	40.3	100	7.5
BIT	102	92	9.0
BIT - SMAC	52.2	99	8.9

BIT, the percentage of solved problems was increased from 92% to 99%.

Closer inspection into the results shows that BIT was struggling with one of the problems in particular, only managing to solve about 40% within the allocated time. After tuning, BIT managed to solve this problem in over 95% of the cases.

B. *UR5 difficult pick-and-place problem*

The second scenario for the UR5 is a more difficult one. Aside from being in a more constrained environment, the UR5 is fitted with a 20cm long vacuum tool. Consider Figure 2 for a depiction of the problem. With significantly

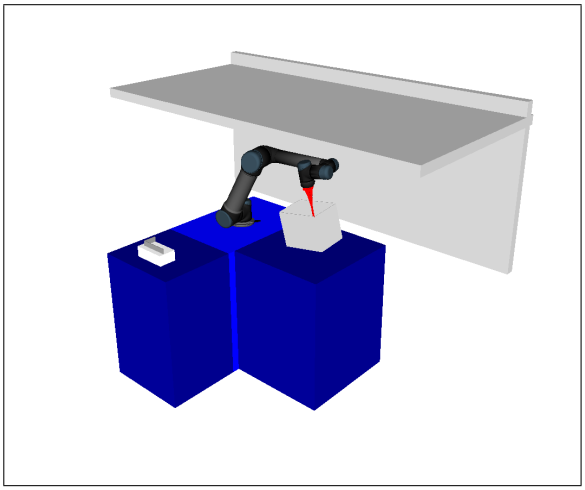


Fig. 2. Model of the second, more complex problem environment for the UR5 manipulator

longer planning times, SMAC could do fewer iterations in the allocated time of 30 minutes. Instead of 5 iterations of 20 problems, this scenario was tuned on 4 iterations of 25 problems. The start and goal states were selected as realistic picking and placing states, with the tip of the vacuum tool well inside the stow bin that can be seen on the right.

Consider Table II for the results of tuning on the second scenario.

As with the first scenario, the improvement of RRTConnect and BiTRRT is rather small. An impressive tuning result

TABLE II

SMAC RESULTS FOR THE SECOND UR5 PROBLEM

PLANNING TIMEOUT 1S

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	165	95	10.9
RRTConnect - SMAC	146	96	10.2
BiTRRT	183	96	10.6
BiTRRT - SMAC	172	96	10.3
BKPIECE	751	45	12.6
BKPIECE - SMAC	450	94	10.6
KPIECE	443	79	11.8
KPIECE - SMAC	337	90	10.6
BIT	286	80	9.7
BIT - SMAC	266	81	9.2

TABLE III

BENCHMARK RESULTS AFTER TUNING FOR KUKA PROBLEM

PLANNING TIMEOUT 2S

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	725	100	6.5
RRTConnect - SMAC	687	100	6.6
BiTRRT	851	99	6.7
BiTRRT - SMAC	807	99	6.9
BKPIECE	1868	17	18.3
BKPIECE - SMAC	1699	64	8.9
KPIECE	1399	68	7.2
KPIECE - SMAC	613	99	6.0
BIT	1239	54	8.5
BIT - SMAC	871	84	6.8

can be seen when considering BKPIECE, which manages to solve just 45% of the queries with the default configuration. Using the SMAC tuning, this was increased to 95%.

C. KUKA LBR iiwa 7 problem

A problem was designed for the KUKA KBR iiwa 7 manipulator that requires maneuvering to and from different poses inside a cabinet. Consider figure 3 for a model of the environment. With an extra degree of freedom and several

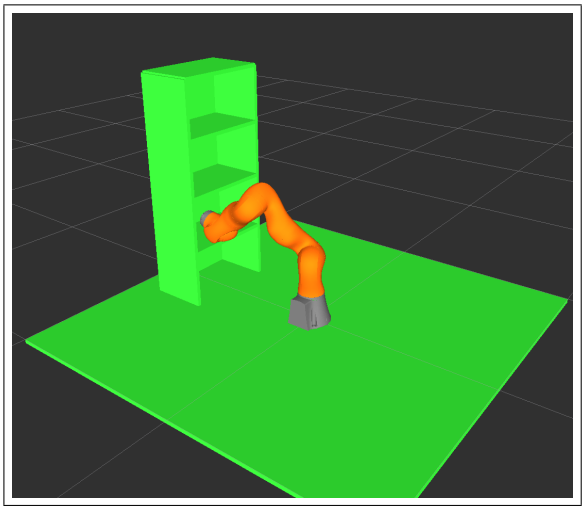


Fig. 3. Model of the environment for the KUKA problem

start and goal states that are difficult to reach, this poses a more challenging problem than the previous two.

In order for SMAC to be able to test a configuration within reasonable time, the amount of iterations was decreased. Consider Table III for the benchmark results after tuning.

Comparing the results of this scene with the UR5 scenes, it is obvious that all planners require more time to find a feasible plan. However, Some planning algorithms, like BiTRRT and BKPIECE seem to suffer more from the extra degree of freedom than others. Judging from the results, the planning timeout of 2 seconds is too short for BKPIECE. Aside from influencing the test results, this had to be accounted for in tuning, as will be discussed further in section VI.

VI. DISCUSSION

A running theme in the previous section that is substantiated with the tabulated results is that automated tuning and configuration is beneficial for all the presented algorithms. In this section, we highlight some of the important aspects from the obtained results.

The most significant improvement was achieved in the case of KPIECE which after tuning consistently managed to solve more problems and did so in significantly less time. This improvement in speed is obtained without sacrificing path length. With the inclusion of the *projection evaluator* parameter in the SMAC optimization, the performance can only improve further. For the other tested planning algorithms, the results were perhaps not as dramatic, but significant none the less.

It is important to note that the performance improvement is not necessarily uniform over all the problems considered. Complex problems typically take longer to be solved and, as there is no scaling between problems in the test set, configurations that improve on these more difficult problems are prioritized. With the BIT* algorithm, we have observed a number of problem queries that could not be solved even after tuning. While tuning improved the performance on the other problems of the set, these specific problems remained either very difficult or outright unsolvable to BIT*. A SMAC cost function that puts a bigger penalty on unsolved queries can potentially be used to seek for configurations that do manage to solve all queries, but this is a topic for further research.

In SMACs manual, it is advised to allow SMAC at least 3 to 400 attempts at finding a good configuration. In addition, the best results are achieved when SMAC is run several times with different starting configurations. However, in the 30 minutes of allowed planning time that was used in the experiments, SMAC often only had time for about 100 configuration tests. As SMAC is often used for algorithms with many more parameters than these planning algorithms, perhaps these requirements can be relaxed somewhat, but further research into the configuration of SMAC and the problem instance is highly encouraged.

It is pertinent to note the importance of correctly specifying the cost function and planning timeout. For the KUKA

scenario, BKPIECE only managed to solve 17% of the queries when the timeout was set for 2 seconds. Running SMAC with a 2 second timeout means that for each query that goes over the average even very slightly, a planning time of 2s is reported. This means that SMAC gathers very little information on the actual performance of a configuration, and is not able to construct a good model. Consider Table IV for different tuning results.

TABLE IV

BKPIECE TUNING RESULTS WITH A VARIABLE PLANNING TIME BENCHMARK WAS RUN WITH 2S TIMEOUT

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
Default	1868	17	18.3
2s tuning	1930	18	16.2
4s tuning	1699	64	8.9

Lastly, the combination of MoveIt and OMPL has been chosen specifically because of the seamless integration between the two software packages and the relative ease with which the combination allows one to use different planners for a given planning problem. Moreover, the available (graphical user) interfaces for these software packages also alleviate the need to elaborate on some of the technical details from a end-user perspective. However, we would like to reiterate that our approach itself is not limited to this specific software combination.

VII. CONCLUSIONS AND FUTURE WORK

We set out in the beginning with the idea of coming up with a tool to enable easy tuning of planning algorithms for a given problem. To this end, we present the SMAC-based tuning method in our work and substantiate the achieved performance improvement in three realistic scenarios using industrial manipulators. Our primary goal with this work is to minimize the amount of background knowledge required to use planning algorithms, particularly in industrial robotic applications. Our long term goal is to further develop this approach in an extensive manner to include any dynamic parameters as well. Eventually, we would like to enable robotics end-users to transition from teaching task specific motions to robots to using planning algorithms to perform task specific motions.

ACKNOWLEDGEMENT

The work leading to these results has received funding from Technolution B.V. The work leading to these results has also received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 609206.

REFERENCES

[1] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[2] Ioan A. Sucan and Sachin Chitta, MoveIt!, [Online] Available: http://moveit.ros.org

[3] Diankov, Rosen. "Automated construction of robotic manipulation programs." (2010).

[4] Jaillet, Lonard, Juan Corts, and Thierry Simon. "Sampling-based path planning on configuration-space costmaps." Robotics, IEEE Transactions on 26.4 (2010): 635-646.

[5] Gammell, Jonathan D., Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic." arXiv preprint arXiv:1404.2334 (2014).

[6] Gammell, Jonathan D., Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs." Robotics and Automation (ICRA), 2015 IEEE International Conference on. IEEE, 2015.

[7] Sucan, Ioan A., and Lydia E. Kavraki. "Kinodynamic motion planning by interior-exterior cell exploration." Algorithmic Foundation of Robotics VIII. Springer Berlin Heidelberg, 2009. 449-464.

[8] Ioan A. Sucan, Mark Moll, Lydia E. Kavraki, The Open Motion Planning Library, IEEE Robotics & Automation Magazine, 19(4):7282, December 2012. http://ompl.kavrakilab.org

[9] B. Cohen and M. Likhachev. (2009). The Search Based Planning Library (SBPL) [Online]. Available: http://www.ros.org/wiki/sbpl

[10] Ratliff, Nathan, et al. "CHOMP: Gradient optimization techniques for efficient motion planning." Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE, 2009.

[11] Kalakrishnan, Mrinal, et al. "STOMP: Stochastic trajectory optimization for motion planning." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[12] Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." Journal of Artificial Intelligence Research 36.1 (2009): 267-306.

[13] Birattari, Mauro, et al. "F-Race and iterated F-Race: An overview." Experimental methods for the analysis of optimization algorithms. Springer Berlin Heidelberg, 2010. 311-336.

[14] Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration." Learning and Intelligent Optimization. Springer Berlin Heidelberg, 2011. 507-523.

[15] Hutter, Frank, Holger Hoos, and Kevin Leyton-Brown. "An evaluation of sequential model-based optimization for expensive blackbox functions." Proceedings of the 15th annual conference companion on Genetic and evolutionary computation. ACM, 2013.

[16] Seipp, Jendrik, et al. "Automatic Configuration of Sequential Planning Portfolios." AAAI. 2015.

[17] Francesca, Gianpiero, et al. "AutoMoDe: A novel approach to the automatic design of control software for robot swarms." Swarm Intelligence 8.2 (2014): 89-112.

[18] Berkenkamp, Felix, Andreas Krause, and Angela P. Schoellig. "Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics." arXiv preprint arXiv:1602.04450 (2016).

[19] Killingsworth, Nick J., and Miroslav Krsti. "PID tuning using extremum seeking: online, model-free performance optimization." Control Systems, IEEE 26.1 (2006): 70-79.

[20] Mark Moll, Ioan A. Sucan, Lydia E. Kavraki, Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization, IEEE Robotics & Automation Magazine, 22(3):96102, September 2015.

Bibliography

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[2] R. Smits, H. Bruyninckx, and E. Aertbeliën, "Kdl: kinematics and dynamics library," *Avaliable: http://www. orocos. org/kdl*, 2011.

[3] R. Diankov, *Automated construction of robotic manipulation programs*. PhD thesis, Carnegie Mellon University, 2010.

[4] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pp. 928–935, IEEE, 2015.

[5] G. Carbone and F. Gomez-Barvo, *Motion and Operation Planning of Robotic Systems*. Springer, 2015.

[6] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Access, IEEE*, vol. 2, pp. 56–77, 2014.

[7] O. Brock, J. Kuffner, and J. Xiao, "Motion for manipulation tasks," in *Springer Handbook of Robotics*, pp. 615–645, Springer, 2008.

[8] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[9] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[10] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3859–3866, IEEE, 2012.

[11] D. Hsu and Z. Sun, "Adaptively combining multiple sampling strategies for probabilistic roadmap planning," in *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, vol. 2, pp. 774–779, IEEE, 2004.

[12] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," *arXiv preprint arXiv:1403.7714*, 2014.

[13] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 521–528, IEEE, 2000.

[14] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, J. P. How, *et al.*, "Motion planning for urban driving using RRT," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 1681–1686, IEEE, 2008.

[15] "Amazon picking challenge website." http://amazonpickingchallenge.org/. Accessed: 2016-05-17.

[16] C. Eppner, S. Höfer, R. Jonschkowski, R. Martın-Martın, A. Sieverling, V. Wall, and O. Brock, "Lessons from the amazon picking challenge: Four aspects of robotic systems building,"

[17] H. Zhang, P. Long, D. Zhou, Z. Qian, Z. Wang, W. Wan, D. Manocha, C. Park, T. Hu, C. Cao, *et al.*, "Dorapicker: An autonomous picking system for general objects," *arXiv preprint arXiv:1603.06317*, 2016.

[18] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Lessons from the amazon picking challenge," *arXiv preprint arXiv:1601.05484*, 2016.

[19] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[20] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2009.

[21] L. E. Kavraki, *Random networks in configuration space for fast path planning*. PhD thesis, stanford university, 1994.

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

[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[24] 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.

[25] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[26] T. Kunz, U. Reiser, M. Stilman, and A. Verl, "Real-time path planning for a robot arm in changing environments," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 5906–5911, IEEE, 2010.

[27] H. Schumann-Olsen, M. Bakken, Ø. H. Holhjem, and P. Risholm, "Parallel dynamic roadmaps for real-time motion planning in complex dynamic scenes," 2015.

[28] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2366–2371, IEEE, 2006.

[29] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm.," in *ICAPS*, pp. 262–271, 2005.

[30] S. M. LaValle, "Rapidly-exploring random trees a new tool for path planning," 1998.

[31] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[32] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1478–1483, IEEE, 2011.

[33] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, pp. 995–1001, IEEE, 2000.

[34] 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*, p. 0278364915577958, 2015.

[35] B. Cohen, I. Şucan, S. Chitta, *et al.*, "A generic infrastructure for benchmarking motion planners," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 589–595, IEEE, 2012.

[36] 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," *arXiv preprint arXiv:1405.5848*, 2014.

[37] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *Robotics, IEEE Transactions on*, vol. 26, no. 4, pp. 635–646, 2010.

[38] 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," *arXiv preprint arXiv:1404.2334*, 2014.

[39] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*, pp. 449–464, Springer, 2010.

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

[41] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3671–3678, IEEE, 2012.

[42] D. Coleman, I. A. Sucan, M. Moll, K. Okada, and N. Correll, "Experience-based planning with sparse roadmap spanners," *arXiv preprint arXiv:1410.1950*, 2014.

[43] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 18–47, 2014.

[44] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 489–494, IEEE, 2009.

[45] C. Park, J. Pan, and D. Manocha, "Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments.," in *ICAPS*, 2012.

[46] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4569–4574, IEEE, 2011.

[47] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.," in *Robotics: Science and Systems*, vol. 9, pp. 1–10, Citeseer, 2013.

[48] I. Siradjuddin, L. Behera, T. M. McGinnity, and S. Coleman, "A position based visual tracking system for a 7 dof robot manipulator using a kinect camera," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1–7, IEEE, 2012.

[49] S. Satorres Martinez, J. de la Casa Cardenas, J. Gamez Garcia, and J. Gomez Ortega, "Position predictive control of an anthropomorphic robotic arm using a time-of-flight camera," in *SENSORS, 2014 IEEE*, pp. 1718–1721, IEEE, 2014.

[50] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pp. 802–807, IEEE, 1993.

[51] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[52] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pp. 1–6, VDE, 2012.

[53] C. Rosmann, F. Hoffmann, and T. Bertram, "Planning of multiple robot trajectories in distinctive topologies," in *Mobile Robots (ECMR), 2015 European Conference on*, pp. 1–6, IEEE, 2015.

[54] C. Rosmann, W. Feiten, T. Wosch, F. Hoffmann, and T. Bertram, "Efficient trajectory optimization using a sparse model," in *Mobile Robots (ECMR), 2013 European Conference on*, pp. 138–143, IEEE, 2013.

[55] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *Robotics, IEEE Transactions on*, vol. 30, no. 6, pp. 1533–1540, 2014.

[56] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," in *Motion and Operation Planning of Robotic Systems*, pp. 3–27, Springer, 2015.

[57] K. G. Shin and N. D. McKay, "Selection of near-minimum time geometric paths for robotic manipulators," *Automatic Control, IEEE Transactions on*, vol. 31, no. 6, pp. 501–511, 1986.

[58] T. Balkan, "A dynamic programming approach to optimal control of robotic manipulators," *Mechanics research communications*, vol. 25, no. 2, pp. 225–230, 1998.

[59] E. A. Croft, B. Benhabib, and R. G. Fenton, "Near-time optimal robot motion planning foe on-line applications," *Journal of robotic systems*, vol. 12, no. 8, pp. 553–567, 1995.

[60] F. Debrouwere, W. Van Loock, G. Pipeleers, Q. T. Dinh, M. Diehl, J. De Schutter, and J. Swevers, "Time-optimal path following for robots with convex–concave constraints using sequential convex programming," *Robotics, IEEE Transactions on*, vol. 29, no. 6, pp. 1485–1495, 2013.

[61] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of time-optimal, jerk-limited trajectories," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3248–3253, IEEE, 2008.

[62] "Reflexxes motion library." http://reflexxes.ws/. Accessed: 2016-05-23.

[63] "Reflexxes motion libraries manual and documentation (type ii, version 1.2.6) - synchronization behavior." http://reflexxes.ws/software/typeiirml/v1.2.6/docs/page_synchronization_behavior.html. Accessed: 2016-05-23.

[64] F. Lange and A. Albu-Schaffer, "Path-accurate online trajectory generation for jerk-limited industrial robots,"

[65] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1243–1248, IEEE, 2006.

[66] L. Jaillet and T. Siméon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1175–1188, 2008.

[67] E. Yoshida, K. Yokoi, and P. Gergondet, "Online replanning for reactive robot motion: Practical aspects," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 5927–5933, IEEE, 2010.

[68] D. Hsu, J.-C. Latcombe, and S. Sorkin, "Placing a robot manipulator amid obstacles for optimized execution," in *Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on*, pp. 280–285, IEEE, 1999.

[69] E. Yoshida and F. Kanehiro, "Reactive robot motion using path replanning and deformation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5456–5462, IEEE, 2011.

[70] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, pp. 507–523, Springer, 2011.

[71] "ros-control documentation - understanding trajectory replacement." http://wiki.ros.org/joint_trajectory_controller/UnderstandingTrajectoryReplacement. Accessed: 2016-05-24.

[72] T. T. Andersen, "Optimizing the universal robots ros driver.," tech. rep., Technical University of Denmark, Department of Electrical Engineering, 2015.

[73] O. Ravn, N. A. Andersen, and T. T. Andersen, "Ur10 performance analysis," tech. rep., Technical University of Denmark, Department of Electrical Engineering, 2014.

[74] "Universal robots twitter account - tweet sent 24 september 2014." https://twitter.com/universal_robot/status/516682281155829760. Accessed: 2016-05-26.

[75] "Ros-industrial descartes." https://github.com/ros-industrial-consortium/descartes. Accessed: 2016-05-26.