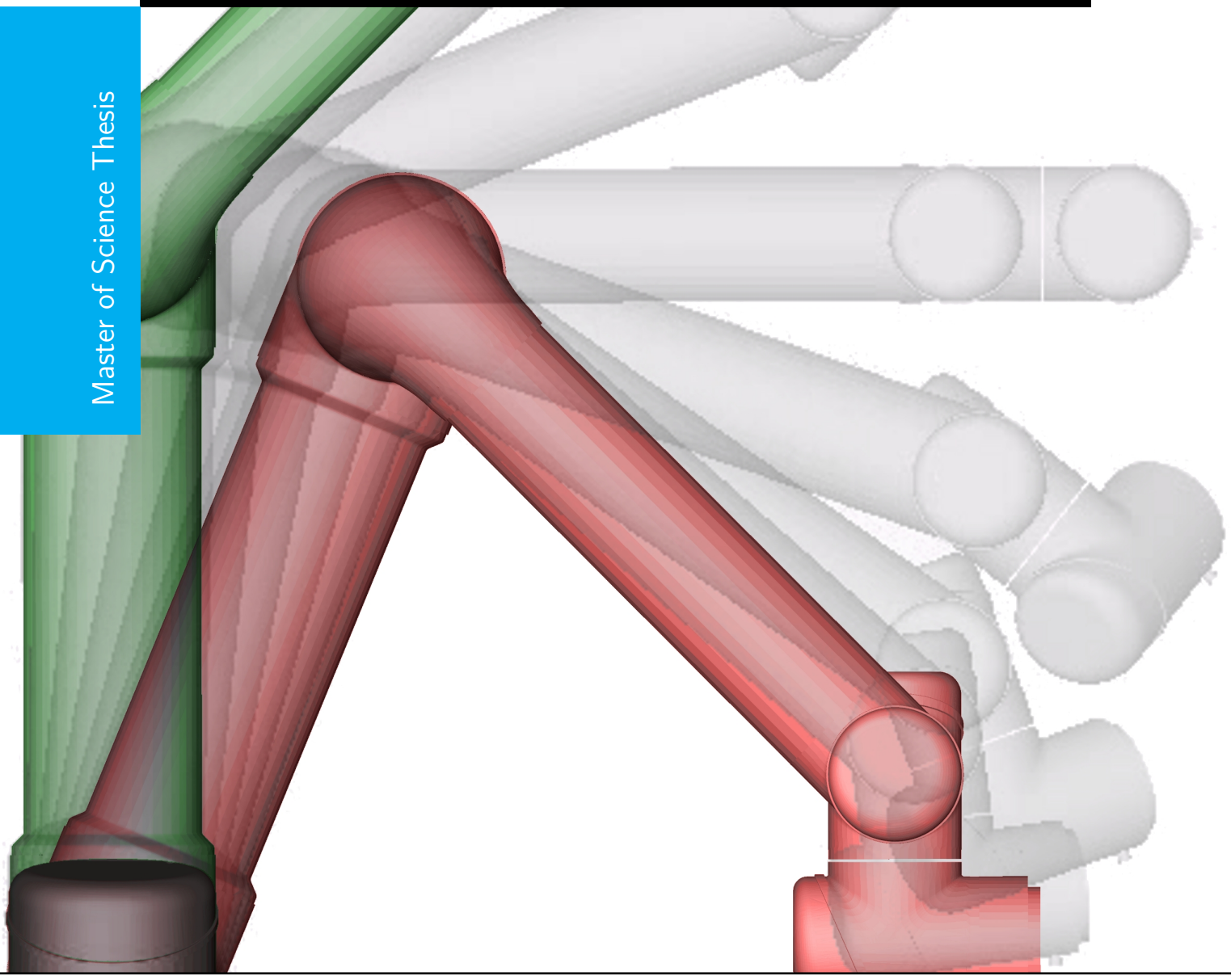
Motion Planning in the State Space

Distance- and Steering-Functions

W.W.F. Spierenburg

Master of Science Thesis



TU Delft Delft University of Technology

BioMechanical Engineering Department

Motion Planning in the State Space
Distance- and Steering-Functions

Master of Science Thesis

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

W.W.F. Spierenburg

August 10, 2016

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

We introduce a motion planning infrastructure, a new set of distance functions and a steering function.
Motion planners are used in robotics in order to plan realistic motions by allowing for variable velocities, as opposed to path planners which either neglect or consider velocities as constant. The main goal of this thesis is to design a planning infrastructure with minimal input, which can solve the motion planning problem for various robots. This all within a reasonable time frame and without preprocessing. Here the state space is the combination of all possible positions and velocities.
To find this solution the Rapidly-exploring Random Tree (RRT) [1] motion planner has been adapted for use in the state space and was tested using a UR5 robot arm model. By itself it was shown to be unable to connect two states, even after adding 100000 states to its tree.
Thus to verify the commonly used euclidean distance metric a new set of simple diIn conclusion the probabilistically complete motion planner RRT has been adapted for the state space, which in combination with the steering function can find a non optimal connection between two states on the gostance functions has been created, which have been verified by calculating their correlation with actual motions and considering their behavior in RRT. Both strong and weak correlations were observed for these distance functions, yet no connections were established between states.
Finally a steering function has been created, PID-connect, which connects both the start and the goal state to a zero velocity state using a PID-controller. PID-connect is able to compute a motion between two states reliably in a short time ($0.02 \pm 0.01s, n = 1000$).
In conclusion a probabilistically complete variant o RRT has been adapted for use in the state space, which in combination with the steering function can find a non optimal motion between two states on the go. This research provides a solid basis from which further advancements in this field can be made.

Table of Contents

1 Introduction **1**

1-1 Problem Formulation . . . 1

1-2 Difficulties of Motion Planning in the State Space . . . 2

1-3 Contributions . . . 3

1-4 Thesis Outline . . . 3

2 Motion Planners **5**

2-1 RRT . . . 6

2-2 RRT-Connect . . . 8

2-3 Discussion . . . 9

3 Modular Motion Planning **11**

3-1 State Space OMPL . . . 12

3-2 Planner . . . 13

3-3 Propagator . . . 13

3-3-1 Rigid Body Dynamics Library & Odeint . . . 14

3-3-2 UR5 Robot . . . 15

3-4 Discussion . . . 16

3-5 Conclusion . . . 17

4 Distance Function **19**

4-1 Simple Distance Functions . . . 20

4-1-1 Configuration Space Inspired . . . 20

4-1-2 Propagation Inspired . . . 25

4-2 Verification of the Distance Functions . . . 27

4-2-1 Total Time Calculation . . . 27

4-2-2 Correlation Comparison . . . 28

4-3 Actual Movement . . . 30

4-4 The Distance Function in RRT . . . 33
4-4-1 Calculation Time . . . 33
4-4-2 Greedy RRT . . . 34
4-4-3 Exploration RRT . . . 36
4-5 Discussion . . . 39
4-6 Conclusion . . . 40

5 Steering Function **41**
5-1 TOPP . . . 42
5-1-1 TOPP-RRT . . . 42
5-2 PID-Connect . . . 44
5-2-1 PID-Controller . . . 44
5-2-2 PID-Connect . . . 45
5-3 Discussion . . . 48
5-4 Conclusion . . . 49

6 Discussion **51**
6-1 Different Robot Types . . . 51
6-2 Distance Function Test & Verification . . . 52
6-3 Collision Detection and Optimality . . . 52

7 Conclusion **55**

8 Recommendation **57**

Appendices **61**

A How-To State Space OMPL **63**

B RRT Trees **67**

C Control Input Selection **71**

D LWPR **73**

E PID-Controller Weights **75**

List of Figures

2-1 An example RRT-tree for planning a motion from the start to the goal state in the state space for a single joint. The arrows depict motions between two states. Note that all motions originate from the start node. 7

2-2 An example RRT-connect-tree for planning a motion from the start to the goal state in the state space for a single joint. The arrows depict motions between two states. Note that motions propagate both from the start and goal state. 8

2-3 Comparison between exploration of RRT and RRT-connect, depicted by dark and light gray respectively. Here it can be seen that RRT-connect needs to explore less states in order to connect the start to the goal state than RRT. For higher dimensions this difference will increase. 9

3-1 In these two figures modules which need to be changed and added in order for OMPL to be used in the state space are depicted. Here Figure 3-1a shows the default OMPL form and Figure 3-1b shows the needed changes for the desired OMPL structure. 12

3-2 Effect of angular velocity ($\dot{q}$) on angle (q). A bigger velocity changes the position more. Thus it can be seen that it is not possible to maintain a position unless the velocity is zero, which also means that it is impossible to maintain a state. . . . 14

3-3 The UR5 robot model displayed in RVIZ. 16

4-1 The euclidean distance function in the configuration space. The arrow indicates the distance. 21

4-2 Contour plot of the DF_{pos} distance function around a goal state. High distances are red and low distances are green. 22

4-3 Two different start states with the same distance, but a different velocity. According to DF_{pos} both have the same distance to the goal state. 23

4-4 Contour plot of the DF_{vel} distance function around a goal state. High distances are red and low distances are green. 24

4-5 Two states, one the start and the other the goal. In the second instance the start and goal states are switched. According to the DF_{vel} distance function both state combinations have the same distance. 25

4-6 Contour plot of a distance function around a goal state. High distances are red and low distances are green. 26

4-7 Contour plot of DF_{EU} around a goal state. High distances are red and low distances are green. 27

4-8 The $PPMCC$ of the cost of various distance functions to time. 20 sets of in total 10000 start and goal state combinations have been used in order to obtain these coefficients. The y-error bars denote the standard deviation. Here it can be seen that DF_{vel} has a significantly lower correlation than the other distance functions. 29

4-9 Detailed view of stronger correlated distance functions. The y-error bars denote the standard deviation of the coefficient. 30

4-10 Example movement plotted over the distance function as depicted in Figure 4-4, with interesting states noted with roman numerals. 31

4-11 Example movement plotted over the distance function as depicted in Figure 4-2, with interesting states noted with roman numerals. 32

4-12 Example movement plotted over the distance function as depicted in Figure 4-7, with interesting states noted with roman numerals. 33

4-13 A RRT-tree consisting of 1000 nodes using DF_{vel} as the distance function and a goal bias of 1. Nodes are shown as circles and the connecting lines depict motions. Here a lack of movement towards the goal state can be seen. 35

4-14 A RRT-tree consisting of 1000 nodes using DF_{EU} as the distance function and a goal bias of 1. Nodes are shown as circles and the connecting lines depict motions. There exists movement towards the goal, but the goal state is not reached. . . . 36

4-15 A RRT-tree consisting of 1000 nodes using DF_{vel} as the distance function and a goal bias of 0.05. Nodes are shown as circles and the connecting lines depict motions. Here it seems that the state space is not properly explored. 37

4-16 A RRT-tree consisting of 1000 nodes using DF_{EU} as the distance function and a goal bias of 0.05. Nodes are shown as circles and the connecting lines depict motions. It can be seen that the state space is properly explored. 38

4-17 Coverage of a 1000 node RRT, in order to show state space exploration. This process was repeated 20 times to obtain the standard deviation which are shown as y-error bars. It can be seen that the mean exploration of DF_{vel} is lower than the other tested distance functions. 39

5-1 When the calculation of the motion from II to III takes longer than the actual motion from I to II, the movement cannot be planned on the go. 43

5-2 Actual movement between two states using the TOPP method on the UR5 robot model. 44

5-3 Example of a good movement using PID-connect. 46

5-4 Example of a bad movement using PID-connect. 47

5-5 When the calculation of the motion from II to III takes shorter than the actual motion from I to II, the movement can be planned on the go. 48

5-6 Actual movement between two states using the PID-connect method on the UR5 robot model. 48

B-1 A RRT-tree consisting of 1000 nodes using different distance function and a goal bias of 1. Nodes are shown as circles and the connecting lines depict motions. . 68

B-2 A RRT-tree consisting of 1000 nodes using different distance function and a goal bias of 0.05. Nodes are shown as circles and the connecting lines depict motions. 69

List of Tables

4-1 Amount of nodes created by distance functions in a 1 second time frame repeated 100 times. 34

5-1 Motion time and computation time of the TOPP-RRT method. Of the 1000 attempts 842 attempts were successful. 43

5-2 Motion time and computation time of the PID-Connect method. All 1000 attempts succeeded. 47

D-1 The MSE of an LWPR model created with varying sample sizes, compared to a verification set of 4000 samples. 74

E-1 The weights used for the PID-controller in PID-connect. 75

Chapter 1

Introduction

In the present day society robots are used for many purposes. They started out as completely human controlled systems, but over time they became more independent. So much that robots can now traverse complex environments on their own, being given only a start and a goal position. In this thesis a system is introduced which further improves the ability of robots to find these different movements.
Path planning is an often used method which is able to plan the movements between two positions using the knowledge of static systems. This method is used in diverse fields such as robotics, manufacturing, drug design and computer animation.
However, the method of path planning greatly simplifies the planning problem, since it often neglects variable velocities or even the entire dynamics of a system. In practice this is either done by using constant velocities, or by using trajectory planners. Trajectory planners attempt to add velocities to a path in some way. Adding velocities to paths might be viable in some cases, however there is no guarantee that a given path is traversable by a robot. Ignoring the velocities or dynamics of a system thus means that not all possible movements between the start and goal position can be found.

1-1 Problem Formulation

A main problem in robotics is determining a possible movement for a system. In this thesis a solution to this problem is proposed, which consists of a method which can be used to plan movements for different systems, and does allow the use of all viable position and velocity combinations.
The following components need to be introduced in order to clearly communicate the difficulties which come with the design of a system used to solve this problem.

1. A world, $\mathcal{W}$, in which $\mathcal{W} = \mathbb{R}^3$.
2. A robot, $\mathcal{A}$, is defined in $\mathcal{W}$, with n joints, $\mathcal{A}_{1,2,..n.}$.

3. A configuration, $q \in \mathcal{C}$, where $\mathcal{C}$ is the n-dimensional configuration space.
4. A state, $X \in \mathcal{S}$, is defined as $X = (q, \dot{q})$, where $\mathcal{S}$ is the $2n$-dimensional state space.
5. A control input, $u \in \mathcal{U}$, where $\mathcal{U}$ is the control space containing all control inputs of $\mathcal{A}$.
6. A path is a subsequent set of q from an initial configuration, q_i, to a goal configuration, q_g.
7. A motion is a continuous set of X from an initial state, X_i, to a goal state, X_g.

Motion planning is thus finding a continuous motion, which connects a start state to a goal state of a robot, while satisfying the constraints of said system.

1-2 Difficulties of Motion Planning in the State Space

The creation of a system which can solve the motion planning problem is not an easy task and comes with many difficulties. The main difficulties will be discussed in this section.

Due to the complex dynamics of the robot it is not possible to directly control the state. This is because the input of a robot consists of a force/torque combination. This input influences the acceleration which changes the velocity, which in turn changes the position. The results of this input are also dependent on the positions and velocities of the robot. This means that all the control input effects are affected by the current state of the robot.

A problem of most states is that velocities influence positions, therefore it is impossible to maintain states over time. Except if all the velocities of a state are zero. Thus it is not possible to solve the motion planning problem one joint at a time, as all joints need to reach their respective position and velocity simultaneously.

Many sampling based methods have some concept of proximity in the space for which they are used. This proximity indicates the distance between two points, and is free to be defined at will. The behavior of sampling based methods depends highly on the definition of the proximity. This distance method is desired to be perfect and quickly computable. However, computing the true proximity is as computationally costly as solving the actual motion planning problem.

A commonly used proximity notion is the euclidean distance metric. This metric works for holonomic systems in the configuration space, but does not work properly in the state space.

For path planning there exist complete algorithms, however two problems exist with implementing them in the state space.
The first problem is that due to the computational complexity of the motion planning problem complete solutions are computationally intractable [2]. Due to this computational cost, methods with a weaker completeness guarantee are used. One such type of approach are the sampling based methods. These methods are probabilistically complete, which means that a solution will be found given enough time, but they cannot determine the non existence of a solution.
The second problem is that motion planning algorithms are difficult to implement in the state space. This is due to the difficulties mentioned in this section and also due to the complexity

of the self designed motion planning algorithm. The implementation of a state-of-the-art comparison, designed by a third party, is potentially even more difficult.

1-3 Contributions

Due to the previous difficulties it is currently no simple task to plan motions through the state space for any type of robot. The goal of this thesis is to ease the future attempts of planning motions. More specifically;
The main goal of this thesis is to design a planning infrastructure with minimal input, which can solve the motion planning problem for various robots, on the go and without preprocessing.

This statement requires two clarifications. First, "on the go" means that a subsequent motion can be calculated, while the current motion is being performed. The second is that the minimum input consists of the start state, the goal state and a robot model, however the use of tuning variables is often unavoidable.

In order to achieve this goal three major contributions are made during this thesis.
The first contribution is the creation of a minimal input infrastructure able to solve the motion planning problem in the state space. As of yet every new motion planner has to be created from scratch. The creation of this infrastructure allows for easier sharing and reproducing of the various motion planners. The system has been divided into multiple modules in order to enable customization per user. Using this modular motion planning algorithm as a basis it is possible to plan motions through the state space.

This system is heavily dependent on the used proximity measures, so a logical second contribution is to discuss the functionality of the often used euclidean distance metric. Other simple distance functions are also introduced as comparison. From comparing these distance methods it can be seen that the commonly used euclidean metric is not qualified for use as a distance function in the state space.

Since a perfect distance function cannot be created, the motion planning problem is difficult to solve. Therefore the planning problem is simplified and solved, this is done using a steering function. A steering function is able to connect two states exactly. The newly created steering function is named PID-connect. The benefit of this method is that it is able to connect two random states in a short time frame.

Due to the time frame for this thesis both optimality and collision detection are not considered.

1-4 Thesis Outline

This thesis has a modular structure, where each contribution has its own chapter. These chapters are preceded by a short explanation of two often used sampling based path planners and the thesis is ends with a discussion and a conclusion.
In chapter 2 the path planners which have been rewritten for use in the state space are discussed. These motion planners are used throughout the thesis. In chapter 3 the modular

motion planning infrastructure is discussed, this infrastructure mainly consists of the extension of the Open Motion Planning Library (OMPL) [3] with a state propagator. In chapter 4 the simple distance functions are introduced. These functions are verified in comparison with actual motions, their behavior during a motion is then discussed, after which their performance is tested in an actual motion planning environment. In chapter 5 two steering functions will be discussed. Both are able to connect two states under strict limitations.
The results obtained from the tests performed in this thesis are discussed in chapter 6, After which conclusions are drawn in chapter 7.
Recommendations for future work are given in chapter 8.

Chapter 2

Motion Planners

This thesis provides a solution to the general motion planning problem in a reasonable time frame, which is tested using a model of the UR5 robot arm. In this chapter the approach to solve the motion planning problem will be discussed. Inspiration for the solution of the motion planning problem can be gained from the solutions of the path planning problem.

Complete planners can successfully indicate whether a solution exists and what it is. In the configuration space complete path planners exist, however they are usable in low dimensional systems only, due to the complexity of the planning problem [4].

The approach which is used has to be reliable, without the need for extensive preprocessing. The following three motion planners are considered the base for the various existing sampling based algorithms. The first motion planner combines fast exploration with fast movements through the configuration space and is called the Rapidly-Exploring Random Tree (RRT) [1]. RRT explores the configuration space by moving towards random configurations from the start configuration. This motion planner is probabilistically complete, which means that given enough time it will explore the entire space. This method is shown to be efficient in the configuration space and can be adapted for use in the state space. It can be seen as the root of most recent sampling based path planning algorithms. RRT will be discussed in section 2-1. There are many methods based on RRT, however most have a clear downside [7]. A method based on RRT which improves the efficiency of RRT and does not have a clear downside is called RRT-connect [8]. It does so by, not only exploring from the start, but also from the goal configuration. This bi-directionality can greatly improve many different sampling based motion planners. RRT-connect will be discussed in section 2-2.
Another method is the Probabilistic Road Map (PRM) [6], which creates a map of the configuration space by selecting multiple configurations and connecting them. It then proceeds to connect the start and the goal configurations to this map, thus finding a solution. This map creation is computationally costly, which makes it not time efficient, and is thus performed beforehand in the configuration space. The creation of the map in PRM can also be seen as multiple sequential runs of RRT, thus in order to create this map first an individual RRT run

needs to be possible.

2-1 RRT

Due to the added complexity of planning in the state space the configuration space path planners can generally not be used in the state space. Sampling based path planners form an exception, as they can be adapted for use in the state space. A sampling based path planner does not explicitly model the entire configuration space. Instead it only needs to map the section of the space which is needed for the current inquiry.
Sampling based motion planners are interesting as the dynamics of the system are not seen as a problem or limitation which needs to be overcome. When using a sampling based motion planner subsequent states follow from the dynamics of the system, instead of struggling against them.
There are multiple sampling based path planners which can be adapted to find solutions for the motion planning problem in the state space [7]. Most of these methods are based upon RRT. The RRT motion planner is especially interesting, because it is probabilistically complete, this means that the probability of finding an existing solution becomes one as the time increases. It should be noted that the effectiveness of this planner in state space dramatically decreases, when compared to its use in the configuration space. This is not only due to the increase in dimensions of the state space, but also due to the interdependency of these dimensions.

RRT is a tree-based algorithm, these type of algorithms generally start from a single state. When a state is part of a tree it is called a node and the start state of a tree is called the seed or root node of the tree. From this root node the tree expands, creating other nodes from which it can then also expand. The tree based planner continues to expand the tree until a state has been reached which is within a given distance of the goal. This distance is called the connection distance and is calculated in the same way as the distance function discussed in section 4-1-1. The pseudo code of RRT can be seen in algorithm 1.

Algorithm 1: GENERATE_RRT(x_{init}, K, Δt)

Data: x: viable state consisting of positions and velocities, $\mathcal{T}$: Tree, $Condition$: Ending condition, Δt: time step, u: robot input

$\mathcal{T}.init(x_{init})$;
while $Condition$ **do**
  $x_{rand} \leftarrow RANDOM\_STATE()$;
  $x_{near} \leftarrow DISTANCE\_FUNCTION(x_{rand}, \mathcal{T})$;
  $u \leftarrow RANDOM\_INPUT()$;
  $x_{new} \leftarrow NEW\_STATE(x_{near}, u, \Delta t)$;
  $\mathcal{T}.add\_node(x_{new})$;
  $\mathcal{T}.add\_edge(x_{near}, x_{new}, u)$;
return $\mathcal{T}$

A short explanation of the used functions will now be given. $\mathcal{T}.init(x_{init})$ creates the tree and adds the start state as a root node to the tree. The next step is to grow the tree. This is done until the termination $Condition$ is reached, this might be a time limit or reaching a

state close to the goal. The first step in expanding the tree is to choose a random state. In practice the goal state can also be used with some probability as the random state in order to speed up the planning process. This probability is the goal bias. Then the closest node to this state is selected for expansion. This closest node is determined using the distance function which will be discussed in chapter 4. The tree will then expand from this closest node. It will do so by selecting a random viable input and applying this input for a duration of time. The state reached through this applied input will then be added to the tree, as well as the motion or edge used to reach this state. At this point the planner either reaches a *Condition* or a new random state is chosen.

In order to find a motion between the start and goal state faster, the goal state can be set as the random state towards which the tree moves. The more this is done the greedier the tree moves towards the goal state. A completely greedy RRT algorithm, however, is unable to reach the goal state due to the imperfections of the distance function, this is discussed further in section 4-3.

A random input is in the algorithm in order to assure that the motion planner stays probabilistically complete. A steering function can be used to choose the control input in order to find an exact motion between two states. The steering function will be discussed in chapter 5.

In Figure 2-1 an example of a possible RRT-tree can be seen. Here the different circles are nodes. The arrows indicate the movements, also called edges, between these states. The start state is shown in red and the goal state is shown in green.

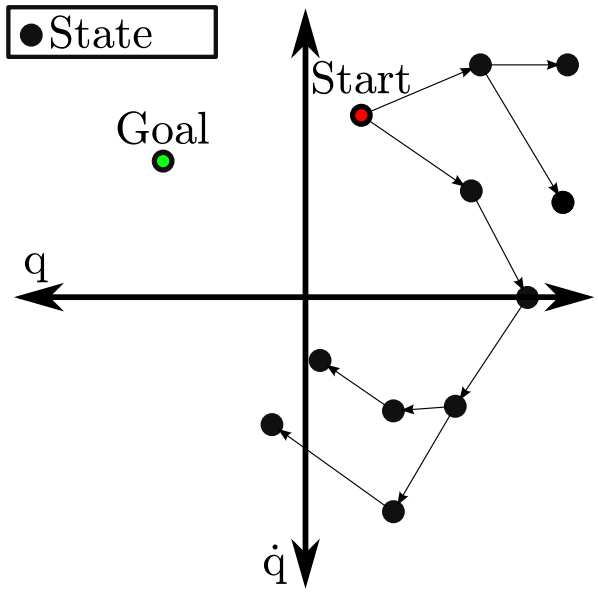


Figure 2-1: An example RRT-tree for planning a motion from the start to the goal state in the state space for a single joint. The arrows depict motions between two states. Note that all motions originate from the start node.

2-2 RRT-Connect

An improvement to the RRT motion planner is the RRT-connect motion planner. The main difference is that it does not use one tree, as RRT does, but two trees. One seed is placed at the start state and the other seed is placed at the goal state. RRT-connect then proceeds to attempt to connect these two trees. This means that the motion planner moves both forwards and backwards in time. A planner which is able to do so is called bi-directional. The expansion still behaves similarly to RRT when moving towards a random state. It differs, however, when it moves towards the goal. When RRT would attempt to connect to the goal, RRT-connect attempts to connect to the opposing tree. The exact nodes which it attempts to connect are determined as follows: First on the target tree a random node is selected, then a node is chosen for expansion, this node is the closest to the current target node. The target node is now reevaluated, The target node is then the node on the opposing tree closest to the node which is to be expanded.

In Figure 2-2 an example of a possible RRT-connect-tree can be seen.

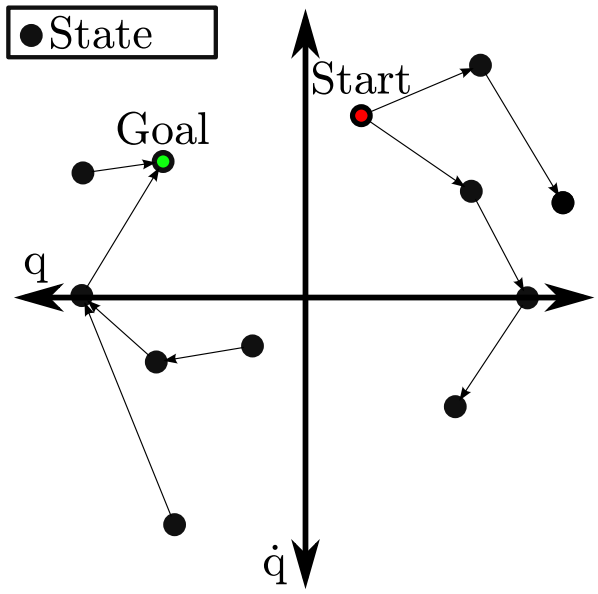


Figure 2-2: An example RRT-connect-tree for planning a motion from the start to the goal state in the state space for a single joint. The arrows depict motions between two states. Note that motions propagate both from the start and goal state.

The reason why RRT-connect is faster than RRT is simplified and depicted in Figure 2-3. Here the expansion of a RRT-tree is depicted by blue, note that it expands from the start and is finished when it reaches the goal state. Here the gray surface depicts the states which has been explored. RRT-connect is depicted by green, it expands from both the start and the goal state and is finished when the two trees connect. In this figure it can be observed that RRT-connect needs to cover less surface in order to connect the start to the goal state, than a RRT-tree needs to cover. For higher dimensions this difference becomes larger. Keep in mind, however that this depiction is a simplification of the functionality of the actual motion planner in state space.

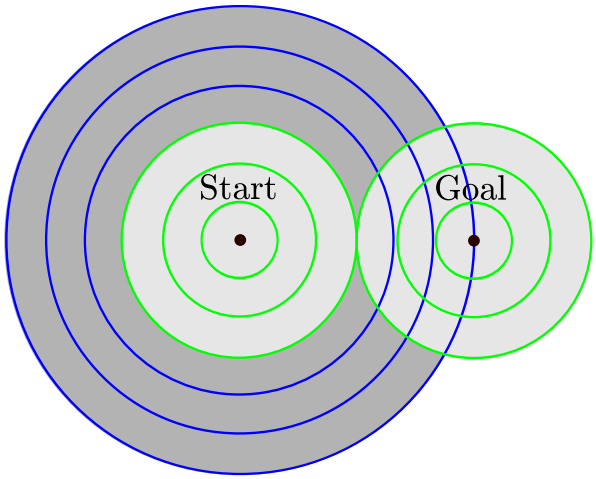


Figure 2-3: Comparison between exploration of RRT and RRT-connect, depicted by dark and light gray respectively. Here it can be seen that RRT-connect needs to explore less states in order to connect the start to the goal state than RRT. For higher dimensions this difference will increase.

2-3 Discussion

Both motion planners RRT and RRT-connect are probabilistically complete, but this says nothing about the practical efficiency of these motion planning systems. This means that there is no realistic time frame within which a probabilistically complete motion planner is guaranteed to find a motion between two given states, except that it will be found between zero and infinity seconds. These planners are probabilistically complete due to the randomness of both the target state selection and the control input selection. If either of these two characteristics of these algorithms would be carelessly removed the algorithms would no longer be probabilistically complete [9].

There are other sampling based motion planners based on RRT, besides RRT-connect. These methods might also be adapted for use in the state space. Most of these motion planners have, as opposed to RRT-connect, clear downsides to their presented improvements. Before any such methods are even considered it is important that the RRT algorithm itself is fully functional. A discussion of these different methods can be found in [7].

A theoretical alternative to the sampling based motion planner would be to explicitly model the entire state space. For these methods it would be practically impossible to solve the path planning problem in the configuration space, let alone the motion planning problem in the state space, due to the sheer size and complexity of the problem [4].

Chapter 3

Modular Motion Planning

In chapter 2 motion planners were discussed. As shown in that chapter the base concepts used in order to describe the motion planners are relatively easy. Their actual implementation, however, is non-trivial.
A problem is that the basic algorithm slows down significantly as the created trees grow larger. Writing an efficient version of the RRT motion planner in C++, compensating for this memory drain is not an easy task.
Here Matlab and Python might be preferred to prototype new planners due to their ease of use, but for the actual implementation C++ is the better option thanks to its improved performance. During this chapter the implementation of these motion planners will be discussed.

In order to properly estimate the functionality of a new motion planning algorithm both the new algorithm and a state of the art comparison are needed. For this comparison all modules should be shared by both algorithms and only the changes which are to be tested are allowed to differ. Such a test also needs to be verifiable by a third party, this means that it needs to be reproducible. This is not the case if the program is recreated from scratch, as many small differences, for example based on user preferences, might be introduced to the algorithms. In order to ease the implementation of various motion planners, a basic infrastructure has been created, which is able to solve the various motion planning problems.

There already exist different infrastructures, which are able to implement the different motion planners with minimum extra input for the configuration space. Examples are the Motion Strategy Library (MSL)[1], Reflexxes[2], the Robotics Library [10] and the Open Motion Planning Library (OMPL) [3]. The infrastructure created for this thesis is an extension to OMPL as the other example libraries are either not open source or no longer maintained.

The infrastructure of OMPL houses various different sampling based motion planners. These motion planners are solely created for the configuration space, which means that it needs to be reconfigured for use in the state space. The difficulty of the reconfiguration lies in

[1] http://msl.cs.uiuc.edu(August 10, 2016)
[2] http://www.reflexxes.ws/(August 10, 2016)

determining what is already in place, what needs to be adapted and what needs to be newly created.

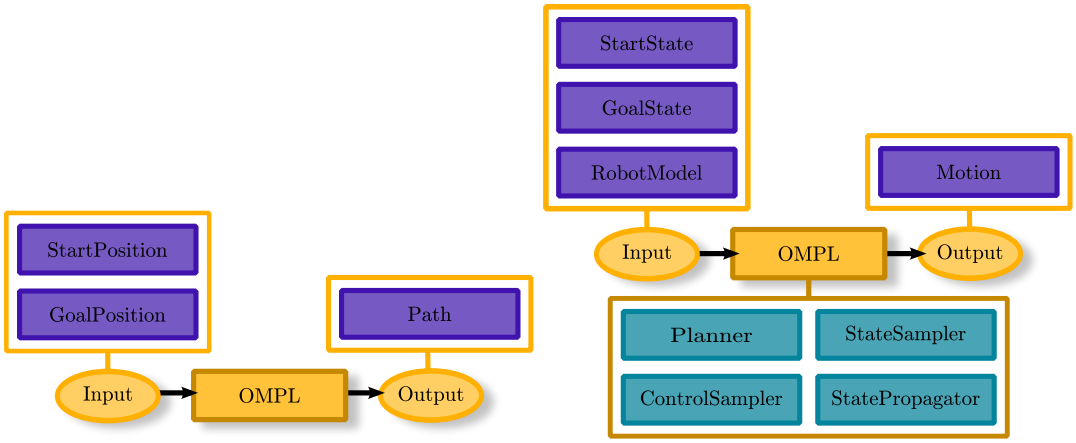
In section 3-1 the configuration space OMPL will be compared to what is desired from the state space variant. In the subsequent sections these adaptations and additions will be discussed. First the changes made in the planner will be discussed in section 3-2. After that the addition of a state propagator will be discussed in section 3-3, which is necessary to determine the movement from one state to another. This state propagator mainly consists of the combination of two libraries, namely Odeint [11] and Rigid Body Dynamics Library (RBDL)[3].
The installation guide of the state space variant of OMPL is discussed in Appendix A.

3-1 State Space OMPL

A comparison can be made between the configuration space OMPL and the adaptation for the state space. The desired output in these situations changes and thus the input is affected as well. The input and output for both the configuration and the state space variant of OMPL can be seen in Figure 3-1a and Figure 3-1b respectively.

Due to the inclusion of velocities the output changes from a path to a motion, were a motion consists of subsequent states. It further differs from a path in that it also stores the efforts, or control inputs, applied to the system in order to reach each subsequent state.

For the input, positions change to states, but more importantly a dynamics model of the robot is now needed in order to determine the effect of the applied efforts on the system.



(a) Configuration Space. **(b)** State Space.

Figure 3-1: In these two figures modules which need to be changed and added in order for OMPL to be used in the state space are depicted. Here Figure 3-1a shows the default OMPL form and Figure 3-1b shows the needed changes for the desired OMPL structure.

These alterations change many modules of OMPL, some of these changed modules are shown in Figure 3-1b. The most important changes will be discussed in the next sections. First the adaptation of the planners will be discussed in section 3-2. After which the addition of the propagator will be discussed in section 3-3.

[3] http://rbdl.bitbucket.org/(August 10, 2016)

3-2 Planner

There are multiple demos available which show of the capabilities of OMPL[4]. There is one particularly interesting demo for this thesis, which is RigidBodyPlanningWithControls. This demo shows how to perform planning for a car like vehicle using RRT and contains many modules also needed for the state space variant of OMPL and RRT.

Three planners will be described in the next section. First the changes are described to adapt RRT for use in the state space. In section 2-2 RRT-connect was also discussed, this planner can be created by using RRT as a basis and will be discussed in the second section. The last section will discuss the needed changes to create a multi seed planner from RRT-connect.

- RRT
  In section 2-1 the different steps of RRT were discussed.
  The first step is to select a random state within bounds, this means that the array holding the configuration has been extended to also include the velocities. In order to determine whether a state is within bounds, the boundaries have been extended to include the velocity bounds. These extensions are an example of the main change which has been made throughout OMPL.
  The second step is to determine the nearest neighbor, this change is non-trivial and will be discussed separately in chapter 4.
  The third step is to choose a control input. The configuration space demo has a targeted method to choose a control input. There is, however, no targeted method for robots in the state space, since their complex dynamics now have to be considered. Thus a random control input has been selected.
  The last step, of adding a new node to the tree, is done in the same way as in the configuration space. This can be done since individual nodes have already been adapted for use in the state space.

- RRT-connect
  In section 2-2 RRT-connect was discussed. In order to use RRT-connect two trees have to be used. One of these trees moves forwards and the other moves backwards in time. A second addition is that when a tree normally moves toward the goal state, they should instead attempt to connect with the closest node of the opposing tree. Both these additions were not yet available in the motion planner and have been added.

- multi seed RRT
  Something not discussed in chapter 2 is the use of multiple seeds, or trees. Many extensions of RRT make use of more than two trees. During the creation of the RRT-connect planner algorithm the addition of multiple trees is considered, so that it can be easily implemented with future planners.

3-3 Propagator

The propagator is used to determine the resulting state of a system, when a control input is applied to it at a given state. This propagation comes with three main difficulties. The first

[4] http://ompl.kavrakilab.org/group__demos.html(August 10, 2016)

problem is that the new position of a state is determined by the previous position and velocity. A positive angular velocities increase the size of an angle, a negative velocity decreases the size of an angle, and zero velocity means that the angle remains the same. This effect is depicted in Figure 3-2.

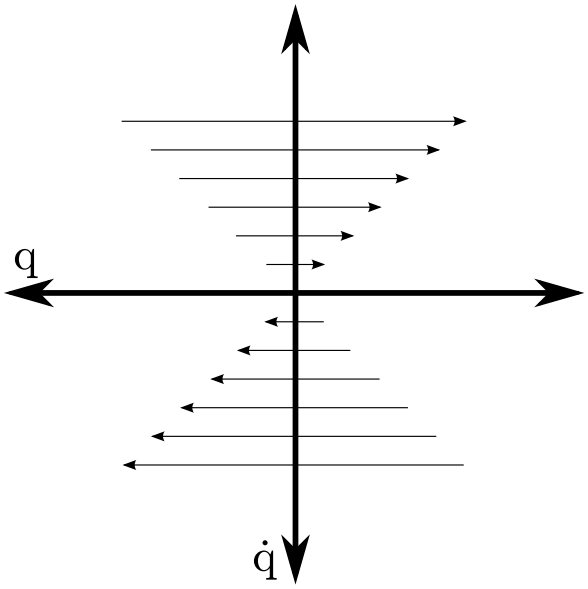


Figure 3-2: Effect of angular velocity ($\dot{q}$) on angle (q). A bigger velocity changes the position more. Thus it can be seen that it is not possible to maintain a position unless the velocity is zero, which also means that it is impossible to maintain a state.

During this chapter and most of this thesis it is important to note that, even though only one joint is shown, robots generally have multiple joints. This introduces a second difficulty, which comes from the dependency of the various joints. The new velocities of a joint are dependent on the state of the entire system, not just the joint itself.
Lastly the control input of the system does not directly influence the state, but influences the accelerations of the system, depending on the current state. These accelerations then in turn affect the velocities.

The next section discusses the method which is used to simulate a system in the state space variant of OMPL. This method primarily uses two libraries namely the Rigid Body Dynamics Library and Odeint.

3-3-1 Rigid Body Dynamics Library & Odeint

As discussed before, a difficulty in propagation is finding the accelerations resulting from the input controls on a robot in a specific state. RBDL helps solve this problem.
Two physics engines which can be used instead of RBDL are Bullet[5] and Open Robotics Automation Virtual Environment (OpenRAVE) [12]. These physics engines have many functionalities which RBDL does not have, but these are not needed for this thesis. The reason

[5] http://bulletphysics.org/(August 10, 2016)

RBDL is chosen over these engines is that it boasts having a highly efficient code to calculate the forward dynamics for robot models.

Calculating the forward dynamics, in this case, means calculating the accelerations of a robot at a given state for the provided control input. This can be written as in Equation 3-1. Here $\ddot{q}$ are the resulting accelerations. Positions and or angles are depicted by q and velocities and or angular velocities are depicted by $\dot{q}$. The control inputs are given in τ. Lastly, *model* is a mathematical representation of the used robot.

$$\ddot{q} = ForwardDynamics(model, q, \dot{q}, \tau) \tag{3-1}$$

The robots consecutive states can be determined using ordinary differential equations which can be solved using Odeint [11], which is an abbreviation for Ordinary Differential Equation INTegrator. Odeint is used since it is a part of Boost[6], which is a library already used by OMPL. The ordinary differential equations are as in Equation 3-2, and can be completed using RBDL. When given a time duration and a time step size Odeint can numerically solve these equations.

$$\begin{aligned} dq/dt &= \dot{q} \\ d\dot{q}/dt &= \ddot{q} \end{aligned} \tag{3-2}$$

The robot model has been provided to RBDL using the Unified Robot Description Format (URDF)[7]. The specific system used throughout this thesis is the universal robots arm UR5, which will be further discussed in subsection 3-3-2.
Instead of using the combination of RBDL and URDF, the non-linear dynamics equations can also be created manually. However this is no easy task, due to the high complexity of these systems. The creation of a URDF file which describes the complex dynamic system is easier. With the general usability of the motion planning infrastructure in mind this combination is implemented, while it is also possible to exchange it for the explicit dynamic equations.

3-3-2 UR5 Robot

As discussed previously, planning in the state space requires a specified robot model. In this case a URDF file can be used to implement the actual model. For the tests run throughout this thesis a robot arm was used, more precisely the universal robots UR5 robot. This robot arm has six different rotary joints and thus a total of six angles and six angular velocities. The URDF model has been taken from a github repository[8]. This robot was chosen for two main reasons. The first reason is that both the robot arm itself and the robot model are readily available for use in this thesis. This means that it is possible to test the results of the motion planner on an actual real life robot. The second reason why it is a good idea to use this robot is because it has a multitude of joints. Finding a movement between states for a robot with only one or even two joints is trivial, if it is possible to plan a movement for a complex robot arm, however, this planner would be much more consequential. A depiction of the UR5 robot can be seen in Figure 3-3.

[6] http://www.boost.org/(August 10, 2016)
[7] http://wiki.ros.org/urdf(August 10, 2016)
[8] https://github.com/ros-industrial/universal_robot(August 10, 2016)

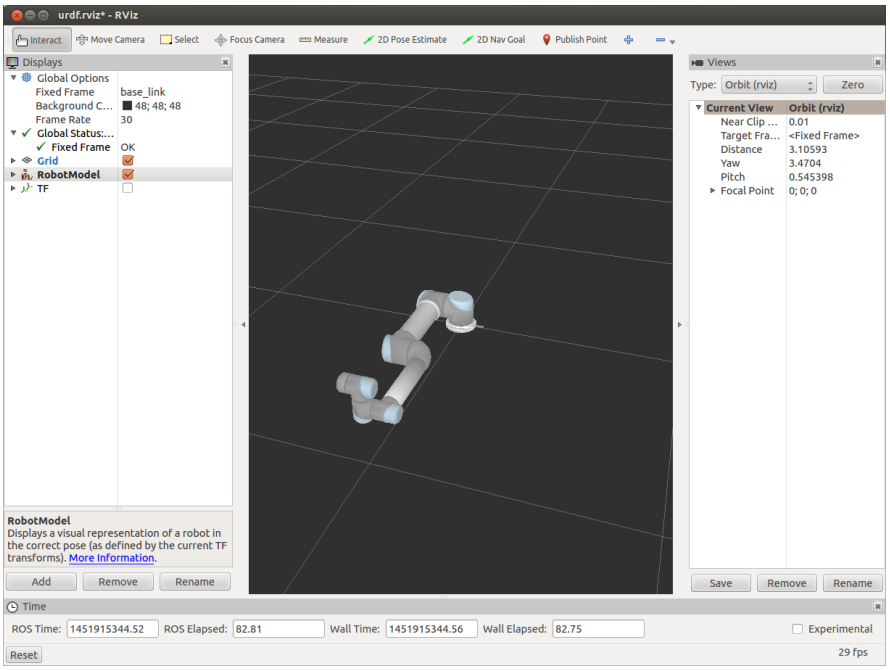


Figure 3-3: The UR5 robot model displayed in RVIZ.

3-4 Discussion

The modular motion planner described in this chapter is an extension to OMPL. Using a modular approach to motion planning, and clearly describing the programs used for the separate modules, allows for the easy reproduction of the various tests performed in this thesis.

Designing a motion planner system solely suited for a specific task might result in a system better suited for its specific task than a more generic motion planning approach as seen in this chapter. That might be especially useful in an industrial environment. However, for research purposes, where reproduction is key, such a system could be without merit as it is very difficult to reproduce the results of such a specific system.

The chosen programs in each module fulfill their intended tasks well. There are, however, many different programs which can also be used to perform these tasks [13]. The question thus remains whether the used programs are the best choice for their intended tasks. This can only be verified by extensive testing of the different programs. These extended tests have not been performed, but due to the modular build of the motion planning algorithm a better suited program can replace its lesser counterpart if found.

Instead of replacing modules, new ones can also be added to the OMPL variant. One such addition is RVIZ. RVIZ is a ROS [14] visualization program which can show the current configuration of the robot. This can be helpful for the interpretation of the output motion of OMPL. By showing the various configurations of the system in sequence a motion can be simulated. RVIZ can also use a URDF file in order to obtain the data of the robot. Different details of the system can be visualized using RVIZ, as is shown in Figure 3-3.

3-5 Conclusion

In this chapter a modular motion planner has been created, by extending the use of OMPL into the state space. This modular motion planner is able to make use of the various background functionalities introduced by OMPL itself.
For all needed modules suitable programs and algorithms have been proposed.

The main extension to OMPL is the addition of a state propagator, in this case by the combination of RBDL and Odeint.

There are two great boons to using the modular motion planner.
The first is that it is much easier to recreate any research previously performed with the modular motion planner. Secondly all the modules of the modular motion planner can be replaced separately per the users desire.

Chapter 4

Distance Function

During the previous chapters the implementation of the state space version of the Rapidly-exploring Random Tree (RRT) [1] was discussed. An essential part of RRT is the nearest neighbor function, which determines the distance from every node in the tree to a specific state. The specifics of the distance function can heavily influence the behavior of the RRT motion planner. The distance is defined as the cost of a movement from one state to another and the cost is generally defined as the time or effort needed to complete this motion. The method used, to calculate the cost from state to state, is called the distance function.

There are two major difficulties concerning the distance function.
The first is that determining the (preferably optimal) motion, from state to state, is often impossible. This then means that it is also impossible to define the cost of a movement between two states. Due to this lack of cost of the optimal motion it is not possible to truly verify a distance function.
The second problem is that there are two competing desires. On one side the best indication of a cost should be found, which is computationally expensive. On the other side there is the desire to execute the nearest neighbor function in a short time frame, even more so since it is used often during RRT. These two opposing desires limit the potential accuracy of the distance function.

This chapter consists of the following sections. First the new simple distance functions are introduced in section 4-1. After which their results are verified by calculating their correlation to the cost of actual motions, this is done in section 4-2. The distance functions, however, cannot be used to successfully reach a goal state in a purely goal oriented RRT, as is shown in section 4-3. Here the output of the distance functions are considered along a specific motion. Despite this limitation, the distance function can be used to explore the state space. This is shown in section 4-4, where the distance function is used in a exploration oriented RRT environment.

4-1 Simple Distance Functions

In this section the used distance functions are introduced. With regards to the desires of computational efficiency and increased accuracy, the introduced distance functions are focused on the desire of computational efficiency. The first two distance functions are inspired by the euclidean distance metric often used in the configuration space, they are explained in subsection 4-1-1. The euclidean distance is used in the original version of RRT and it is still used as a distance function or a baseline for new distance functions in more recent papers [15, 16, 17]. The benefit of this distance function is that it is very fast, however the accuracy of this distance function is lacking. The second two methods attempt to use the information, provided by the velocities, more sensibly, in order to improve the accuracy of the distance function without greatly increasing the computational cost. These methods are inspired by motion propagation and are explained in subsection 4-1-2.

4-1-1 Configuration Space Inspired

When a RRT planner is used in the configuration space a trivial distance function can be used. The calculation of this distance function is specified in Equation 4-1. Here the start configuration is defined by X_s and the goal configuration by X_g. The angles and positions of the robot are defined in q, where n defines the number of degrees of freedom of the robot. The actual calculation of the distance function, is done by taking the euclidean distance between the start and goal configuration, which results in DF_{conf}.

$$\begin{aligned} X_s &= [q_{s1}, ..q_{sn}] \\ X_g &= [q_{g1}, ..q_{gn}] \\ DF_{conf} &= \sqrt{\sum_{i=1}^{n} (q_{gi} - q_{si})^2} \end{aligned} \tag{4-1}$$

When the robot has only two degrees of freedom the calculation of the distance by the distance function can be depicted as in Figure 4-1. The red circle is the start state and the green circle the goal state. The arrow is a depiction of the distance between the two states.

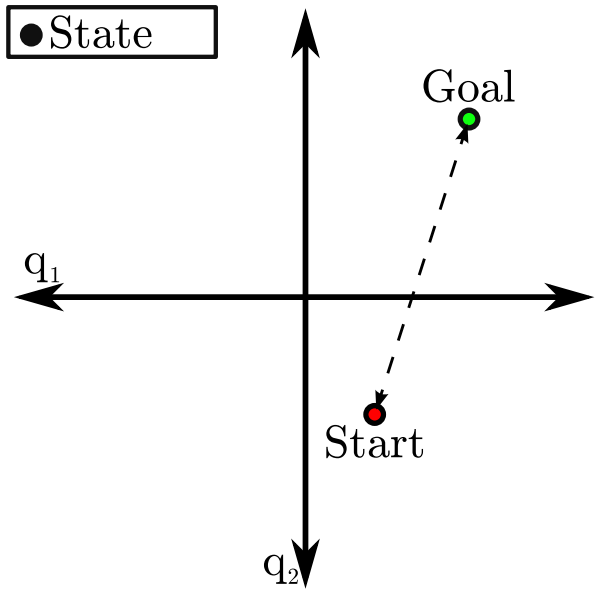


Figure 4-1: The euclidean distance function in the configuration space. The arrow indicates the distance.

In the state space velocities are added. The distance function needs to account for this addition of velocities.
The two distance functions, which are based on the configuration space distance function, attempt to convey the essence of DF_{conf} into the state space. The first distance function which will be discussed is called DF_{pos} and the second distance function is called DF_{vel}.

Euclidean Distance: Position

The first distance function uses the same information as the configuration space distance function, namely only the positions of the robot. The new distance function is thus called the euclidean position distance function or DF_{pos} for short. A benefit of this method is that the calculation is very fast, however in turn it completely ignores velocity. The specific calculation for DF_{pos} can be seen in Equation 4-2, here q signifies the positions and $\dot{q}$ the velocities of the robot.

$$\begin{aligned} X_s &= [q_{s1}, ..q_{sn}, \dot{q}_{s1}, ..\dot{q}_{sn}] \\ X_g &= [q_{g1}, ..q_{gn}, \dot{q}_{g1}, ..\dot{q}_{gn}] \\ DF_{pos} &= \sqrt{\sum_{i=1}^{n} (q_{gi} - q_{si})^2} \end{aligned} \tag{4-2}$$

In Figure 4-2 the contour plot for DF_{pos} is visualized for a single joint of the UR5 robot arm. Here the blue dot is the goal state. In this figure it can be seen that for multiple start and goal state combinations the value of the distance function is the same.

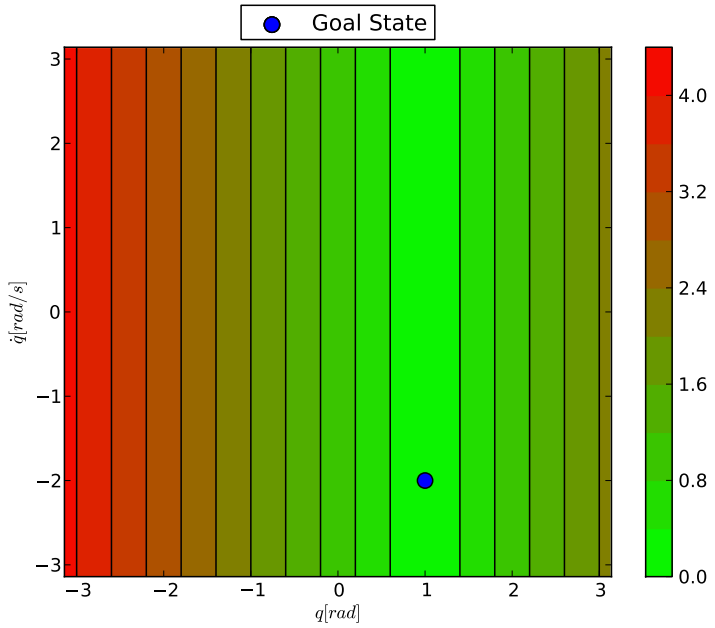
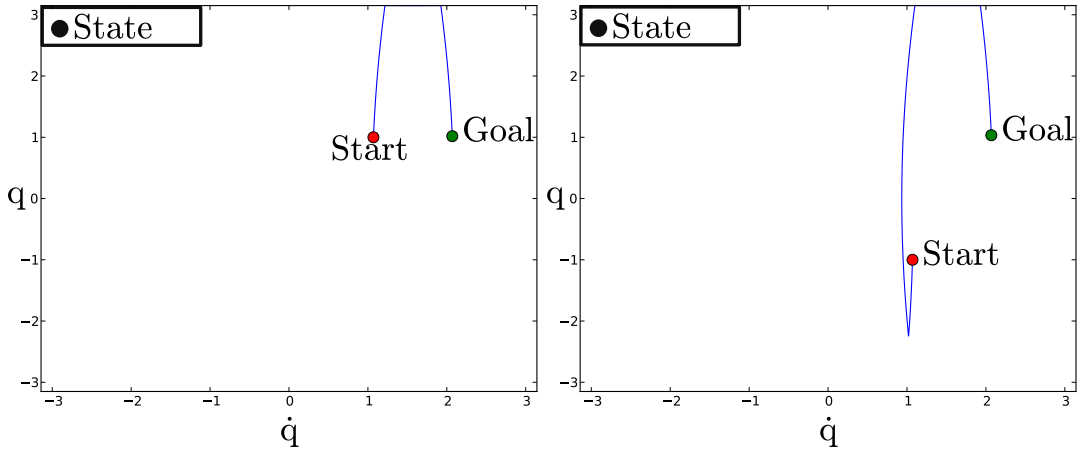


Figure 4-2: Contour plot of the DF_{pos} distance function around a goal state. High distances are red and low distances are green.

The distance function DF_{pos} only uses the positions and ignores the velocities. This can lead to a bad indication of the cost needed to move from state to state, an example of this problem is shown in Figure 4-3. These motions have been obtained using TOPP (section 5-1), for the UR5 robot arm. Here all the start and goal states have been set to zero except for $joint : 2$ for which the motions are depicted. Both these start and goal state combinations have the same distance according to DF_{pos}, however due to the ignored velocity of both states the path in Figure 4-3a between states is much shorter than the path in Figure 4-3b. This means that DF_{pos} is potentially a bad distance function.



(a) Low cost movement **(b)** High cost movement

Figure 4-3: Two different start states with the same distance, but a different velocity. According to DF_{pos} both have the same distance to the goal state.

Euclidean Distance: Position & Velocity

The next distance function can also be seen as a logical continuation of the configuration space distance function DF_{conf} into the state space. This is because it uses the same input as DF_{conf} does; the entire array of the state. This means that not only the positions are used, but the velocities of the robot are used as well. The newly introduced distance function calculates the euclidean distance between two states and is called DF_{vel} for short. The actual calculation method can be seen in Equation 4-3.

$$\begin{aligned} X_s &= [q_{s1}, ..q_{sn}, \dot{q}_{s1}, ..\dot{q}_{sn}] \\ X_g &= [q_{g1}, ..q_{gn}, \dot{q}_{g1}, ..\dot{q}_{gn}] \\ EU_{vel} &= \sqrt{\sum_{i=1}^{n}(q_{gi} - q_{si})^2 + \sum_{i=1}^{n}(\dot{q}_{gi} - \dot{q}_{si})^2} \end{aligned} \tag{4-3}$$

This distance function uses all the available state information, but it is still potentially inaccurate. In Figure 4-4 a contour plot of DF_{vel} is depicted. Here the goal state is again depicted by the blue dot. In this figure it can be seen that when the goal state is defined there is a circle of states around the goal which all have the same DF_{vel} distance to the goal state.

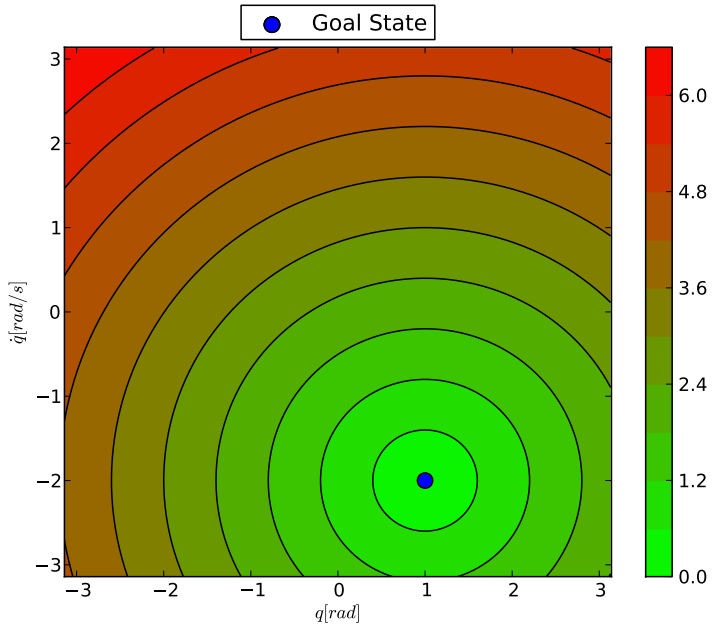
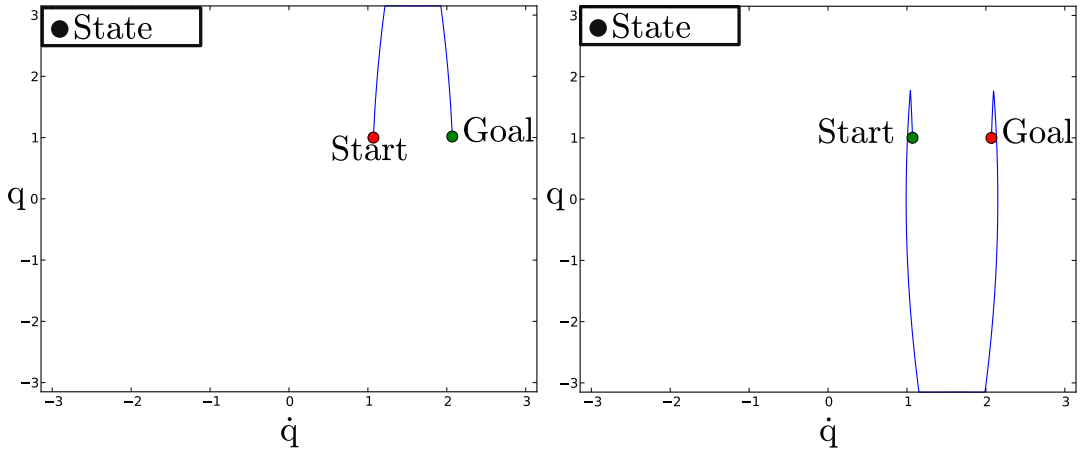


Figure 4-4: Contour plot of the DF_{vel} distance function around a goal state. High distances are red and low distances are green.

This method can again lead to a bad indication for the cost of a movement between states. This potential inaccuracy is depicted in Figure 4-5, where two combinations of states are shown which have the same distance according to DF_{vel}. These motions have been obtained using TOPP (section 5-1), for the UR5 robot arm. Here all the start and goal states have been set to zero except for $joint : 2$ for which the motions are depicted. It can be seen that taking both positions and velocities and treating them similar in a distance function can lead to a incorrect indication of the cost of a movement.



(a) Low effort movement **(b)** High effort movement

Figure 4-5: Two states, one the start and the other the goal. In the second instance the start and goal states are switched. According to the DF_{vel} distance function both state combinations have the same distance.

4-1-2 Propagation Inspired

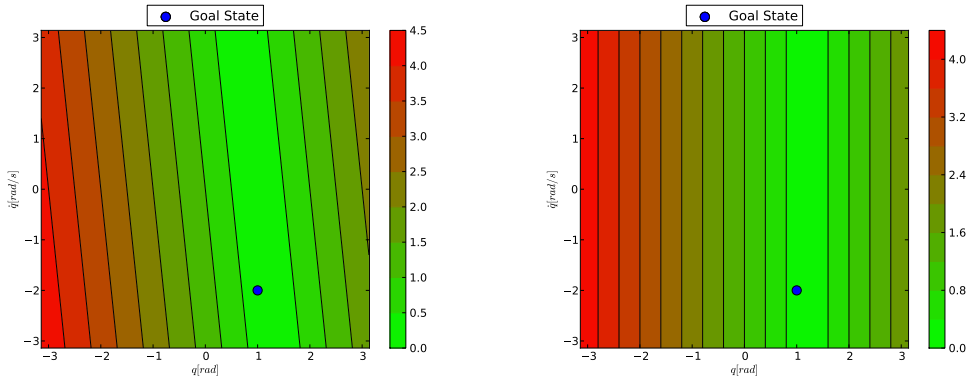
The distance function inspired by the propagation of a robot does take the influence of velocity on a position into account. This means that this distance function not only takes into account the available state information, it also uses this state information in a sensible manner. It still, however, only serves as an indication of the cost of a movement between states and not of the actual cost. The calculation of the proposed distance function is shown in Equation 4-4. Here the start state is propagated a certain number (m) of time steps (dt) forward in time and compared to the goal position, this is the DF_{Forw} section. The second part of the calculation is where the goal state is propagated m time steps backward in time and then compared to the start position, this is the DF_{Back} section. The combination of these two values results in the distance function called DF_{EU}, as it is based on the Euler method.

$$\begin{aligned} q_s &= [q_{s1}, ..q_{sn}, \dot{q}_{s1}, ..\dot{q}_{sn}] \\ q_g &= [q_{g1}, ..q_{gn}, \dot{q}_{g1}, ..\dot{q}_{gn}] \\ q'_s &= [q_{s1} + m * dt * \dot{q}_{s1}, ..q_{sn} + m * dt * \dot{q}_{sn}] \\ q_g, &= [q_{g1} - m * dt * \dot{q}_{g1}, ..q_{gn} - m * dt * \dot{q}_{gn}] \\ DF_{EU} &= \sqrt{\underbrace{\sum_{i=1}^{n} (q_{gi} - q'_{si})^2}_{DF_{Forw}} + \underbrace{\sum_{i=1}^{n} (q_{g,i} - q_{si})^2}_{DF_{Back}}} \end{aligned} \tag{4-4}$$

As opposed to the Euler stepper function seen in Equation 4-4, this propagation can also be done with the Runge-Kutta-Dopri5 stepper function provided by Odeint in combination with RBDL. This variation on the distance function is called DF_{RK}. The benefit of DF_{RK} is that it is theoretically more accurate than DF_{EU}. In order to determine the propagated states the control input provided to RBDL needs to be determined. In order to reflect effect caused by the different control inputs on the propagated states, the control inputs are set to the average

of the lower and upper input bounds, which are zero for each individual joint.
An alternative this control input is defining the needed input for zero acceleration, thus maintaining the current velocity. This can be done using the Inverse dynamics function provided by RBDL. However using this method means that the DF_{RK} distance function would become even more computationally inefficient.
The next section explicitly mentions DF_{EU}, however the same principles also apply to the DF_{RK} distance function.

In order to increase the understanding of DF_{EU} first the contour plots of both the DF_{Forw} and the DF_{Back} section are shown. This is done in Figure 4-6. In Figure 4-6b it can be seen that the resulting values for DF_{Back} are very similar to the resulting values for DF_{pos}, the difference between these two methods is that the global minimum of DF_{Back} is not at the exact position of the goal state. In Figure 4-6a the contour plot of DF_{Forw} is shown. Here it can be seen that the result from DF_{Forw} looks like a slanted version of DF_{Back}, this is due to the influence of the velocities of the initial state on the propagated to position.



(a) Contour plot of DF_{Forw} **(b)** Contour plot of DF_{Back}

Figure 4-6: Contour plot of a distance function around a goal state. High distances are red and low distances are green.

The DF_{EU} distance function combines both the DF_{Forw} and the DF_{Back} sections. The section of DF_{Forw} is used in order to improve the cost of a state where the position m time steps away is the same as the goal position. This means that for the state which has a distance of zero according to DF_{Forw} the goal position will be reached in m time steps. The section of DF_{Back} is incorporated in DF_{EU}, since it has a lower cost for a position which the m time steps backwards propagated goal state would reach. These two methods separately do not define an ideal start and goal state combination, but as they are combined in DF_{EU} there is a single ideal state to which a RRT-tree can converge. This can be seen in the contour plot of DF_{EU} depicted in Figure 4-7. The distance function is designed so that this single state should reach the desired goal state after m time steps. Since the DF_{EU} uses the influence of the velocity on the position, it has the potential to provide a good insight in the cost of a movement between states. It is however still only an estimate of the cost of this movement.

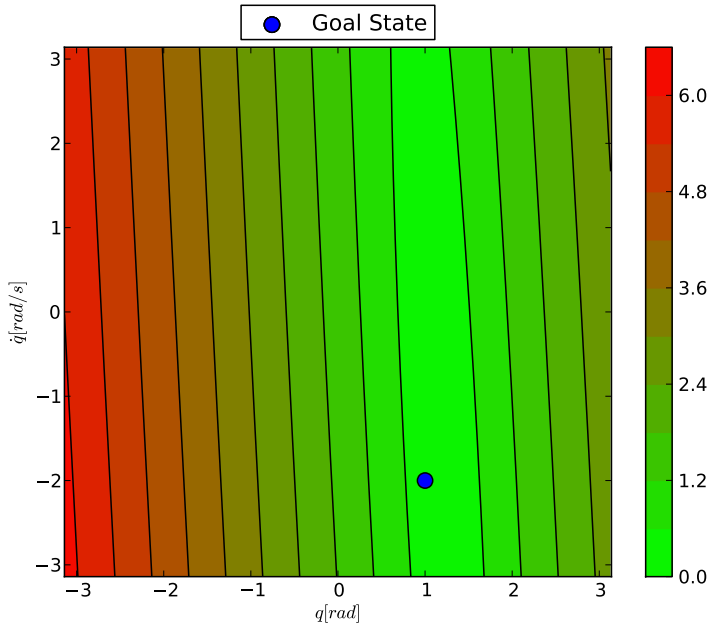


Figure 4-7: Contour plot of DF_{EU} around a goal state. High distances are red and low distances are green.

4-2 Verification of the Distance Functions

Now that several possible distance functions have been proposed, it is necessary to verify the potential of these different distance functions. According to the definition of a distance function, a distance function gives an indication of the cost of a movement between two given states. The distance function can thus be verified by calculating the correlation of the distance functions and the time needed to perform a motion between two states.

Needed for this verification is a set of start and goal states. For these state combinations the motions will be determined, at which point the time needed is known. The distance function uses the start and goal state to calculate the motion cost. The details of this process are discussed in subsection 4-2-1.

The calculation of the correlation between the time and the distance function cost will be discussed in subsection 4-2-2.

4-2-1 Total Time Calculation

The first step, in the verification process, is to define a start and goal state combination. These states can be chosen at random as long as they are within the predefined boundaries of the robot.

The second step is to define a movement between these two state combinations. In order to obtain this motion Time Optimal Path Parameterization (TOPP) [18] is used. This method uses a simple path for the UR5 robot arm and determines the time of each subsequent configuration in order to change the path into a motion. This method will be discussed

in detail in section 5-1.
The motion resulting from this method is not the optimal movement between the two states. It is only able to provide the cost of a possible motion.
The time duration of this obtained motion is stored for the determination of the correlation.

The third step is to calculate the cost according to the distance functions. The distance functions can be calculate this cost using the start and the goal state. This cost is stored for every tested distance function.

At this point both the cost according to the distance function has been obtained as well as the cost of an actual motion for one state combination. This process can be repeated for several start and goal state combinations in order to obtain the actual datasets. The correlation between these two data sets can then be calculated, which is discussed in the next section.

4-2-2 Correlation Comparison

Using the gathered data of the movements between start and goal state combinations, the correlation can be calculated between the effort and the distance functions. More specifically the Pearson Product Moment Correlation Coefficient or $PPMCC$ can be calculated for the common logarithm of the values of the time cost and the distance functions. The $PPMCC$ is a measure of the linear correlation between two variables. Its value ranges from minus one to one. Where zero means that there is no correlation, minus one means perfect negative correlation and one indicates a perfect positive correlation.
The correlation coefficient has been calculated for 20 sets of in total 10000 start and goal combinations. The different sets are used to determine the mean and standard deviation of the coefficient. The results of this comparison can be seen in Figure 4-8. Here the vertical axis denotes the $PPMCC$ value and the y-error bars denote the standard deviation. On the horizontal axis the various distance functions are displayed. In total 12 different distance functions were compared to the time duration. The number after DF_{EU} and DF_{RK} denotes the number of time steps, or m, for which has been propagated.

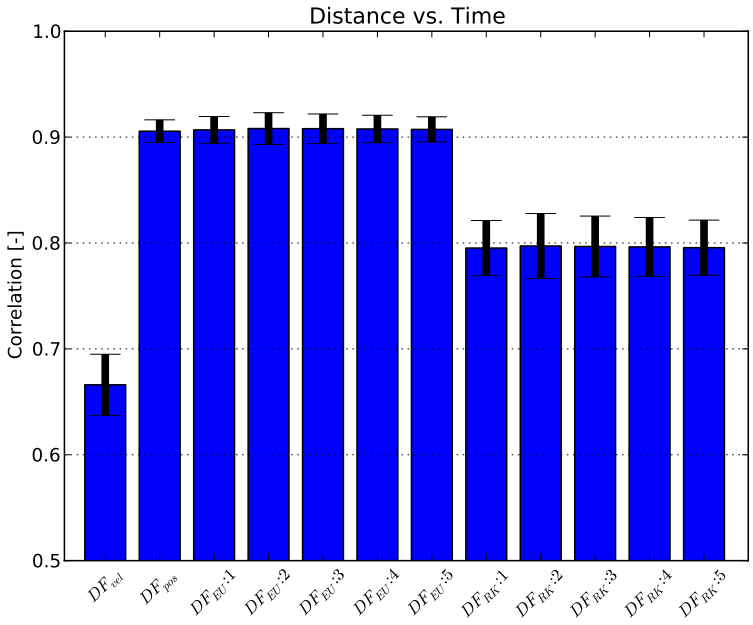


Figure 4-8: The $PPMCC$ of the cost of various distance functions to time. 20 sets of in total 10000 start and goal state combinations have been used in order to obtain these coefficients. The y-error bars denote the standard deviation. Here it can be seen that DF_{vel} has a significantly lower correlation than the other distance functions.

In Figure 4-8 it can be seen that DF_{vel} has a significantly lower correlation than the other distance functions, but not much can be said for the differences between the other correlations. Therefore a closer inspection is needed for the higher correlation coefficients, this is provided by Figure 4-9. Here it can be noted that DF_{EU} has the highest mean correlation, even though this is not a significant difference with the other shown distance functions. Most important to note is that there is a clear difference between the correlation of DF_{EU} and DF_{pos} in respect to the other distance functions.

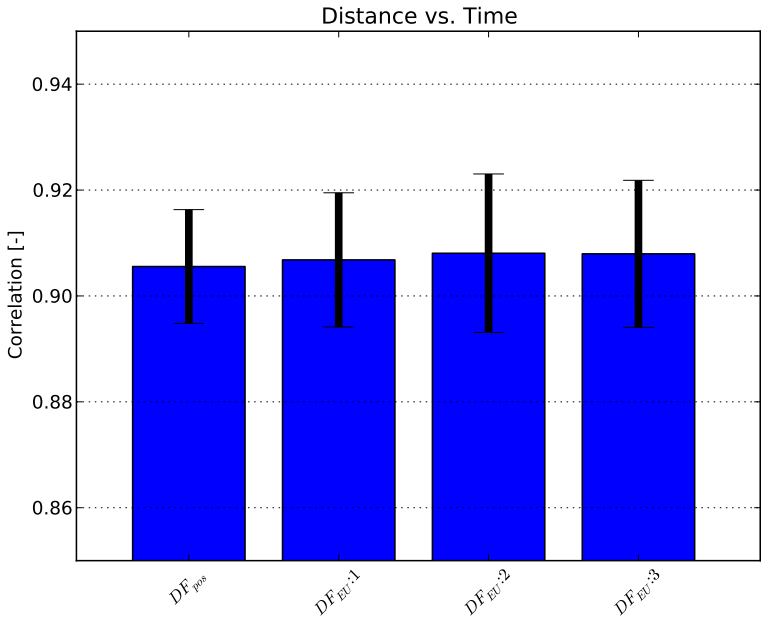


Figure 4-9: Detailed view of stronger correlated distance functions. The y-error bars denote the standard deviation of the coefficient.

4-3 Actual Movement

At this point it is important to note that even though the correlation between motion and distance function is important, it is not final. This section was written in order to show the problems of the different distance functions. These disadvantages are shown by laying a possible movement from a start to a goal state over a contour plot of a distance function.
The first distance function for which this is done is DF_{vel}, which is shown in Figure 4-10. Here high costs are shown in red and low costs in green. The circles are nodes, with the start state and goal state being red and green respectively. The movement is given by the black arrow. Along this movement there are interesting states. These points are the start and goal states, along with the state indicated by an I.
The start state is interesting since the states along the movement following this point have a higher distance function value, than the start state itself. This means that the movement cannot be found using a random controller with a maximum goal bias. After point I the distance value of the movement decreases. This means that DF_{vel} can successfully determine a nearest neighbor to the goal which is actually closer on the defined motion.
When the goal bias is lower than maximum it is possible to find this movement if the random exploration reaches a state beyond point I.

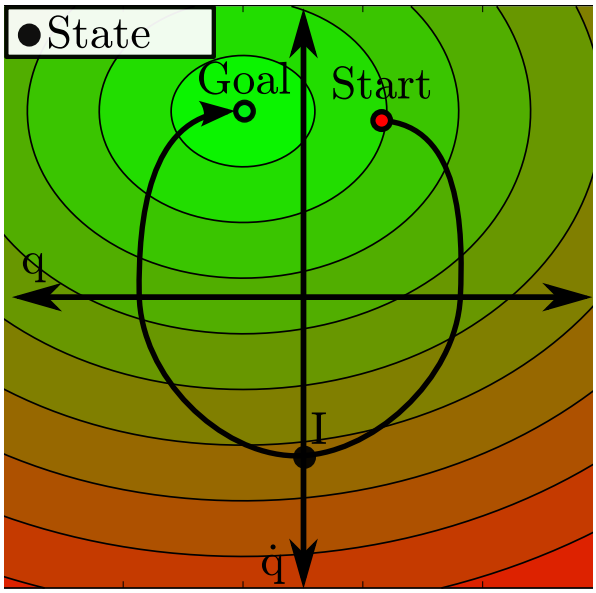


Figure 4-10: Example movement plotted over the distance function as depicted in Figure 4-4, with interesting states noted with roman numerals.

For the cost function DF_{pos} the problem is very similar, but a little better. For the DF_{pos} cost along a motion there are three additional points of interest, next to the start and goal state: namely state I, II and III.
The start state and II are points on the movement which are located at local minimums according to the distance function. State I and state III on the other hand are placed at local maximums along the movement. The fact that the distance function does not continuously decrease along the movement again means that the movement cannot be found with a simple random controller with a maximum goal bias. If the goal bias is lower than maximum it is possible to find this movement, when the random exploration reaches a state on the movement beyond point III.
It is also able to follow the movement between point I and point II. This means that the leap between descending distance sections is smaller than with the use of the DF_{vel} distance function. However point II has the same distance to the goal as the goal itself. This means that there is no reason to move away from point II towards the goal if only the distance function is concerned.

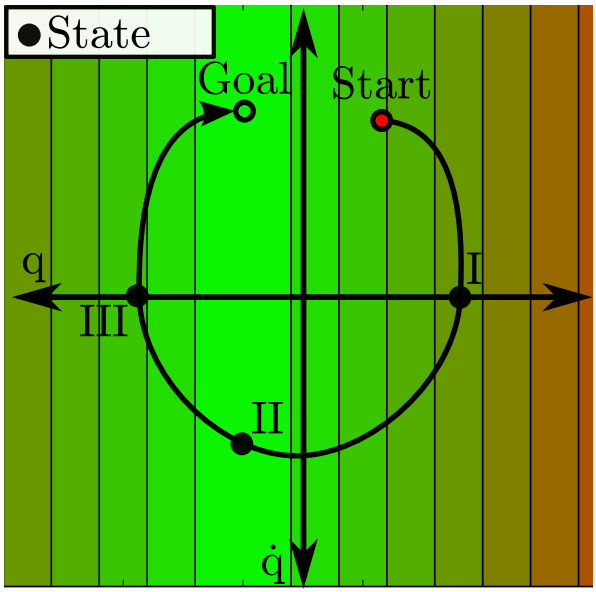


Figure 4-11: Example movement plotted over the distance function as depicted in Figure 4-2, with interesting states noted with roman numerals.

The last method discussed is the DF_{EU} distance function. This methods behavior along the motion is similar to DF_{pos}, but there are two major differences between these two distance functions.
The first difference is that DF_{EU} has a single global minimum which is located close to the goal state. The second difference is that point I, II and III are shifted along the movement. More specifically the section of movement between point III and the goal state is increased, while the section of movement between the start state and point I is decreased. These two sections have a respectively easy and difficult to follow movement due to the change of the distance function. This point shift is thus theoretically beneficial.

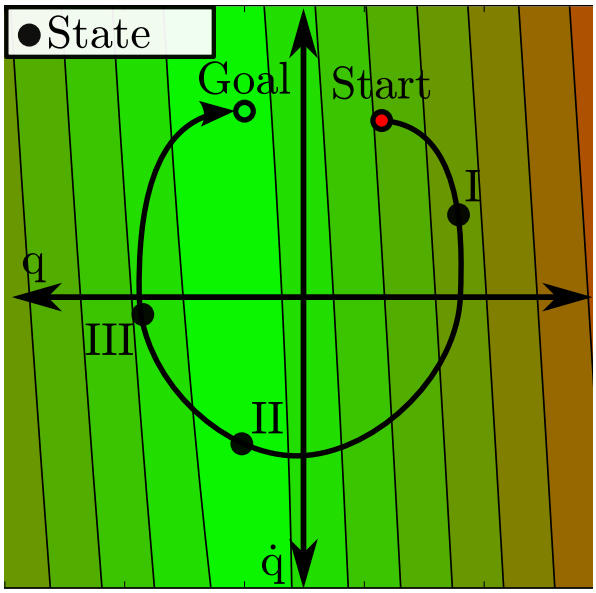


Figure 4-12: Example movement plotted over the distance function as depicted in Figure 4-7, with interesting states noted with roman numerals.

4-4 The Distance Function in RRT

In the previous sections it was discussed whether the distance functions provide an accurate indication of the cost to go. The expected problems of these distance functions have also been discussed. In order to assess the actual functionality of the distance functions in combination with a motion planner, they need to be used in an actual planning environment. The next sections discusses this functionality. The motion planner used in this case is RRT in combination with the UR5 robot arm.

As a preliminary test the start and goal states have been chosen at random. The RRT motion planner then attempted to find a movement which connects the start state and the goal state. The test was performed many times for extensive durations of time, however only approximate solutions were found. Approximate solutions come close to the desired goal state, but do not enter the specified connection distance. Is a small area around the target, which is used to decrease the computational cost in comparison to finding the exact soluAttion.

Here there are three specific points concerning the distance functions which can be investigated and discussed. In subsection 4-4-1 the first point is discussed which considers the calculation time needed by each distance function. The second point is discussed in subsection 4-4-2, it concerns the behavior of a purely goal oriented, or completely greedy, RRT. In subsection 4-4-3 the third point is discussed, namely whether the distance functions allow for decent exploration of the state space.

4-4-1 Calculation Time

The two main demands of a distance function are that it provides an indication of the cost to go between two states, and that it has a low calculation time. In order to show the time

consumed by each distance function they are used in the RRT environment to create nodes. RRT is allowed to create nodes for 1 second and this is then repeated 100 times using the various distance functions. Beforehand we can note that for bigger trees it is computationally more costly to add new nodes, than for small trees. This is since all nodes of the tree need to be compared to the targeted state. The number of nodes that have been created in 1 second are shown in Table 4-1. These values have been obtained on *ubuntu 12.04 LTS x64*, with 3.8 *GiB* memory, *Intel Core2 Duo CPU T9600 @ 2.80GHz * 2* processor and *Quadro FX 770M/PCIe/SSE2* graphics card.

Distance Function	Mean $[nodes]$	SD
DF_{vel}	1092.29	135.04
DF_{pos}	1290.08	189.61
DF_{EU}	1097.98	116.08
DF_{RK}	130.36	14.97

Table 4-1: Amount of nodes created by distance functions in a 1 second time frame repeated 100 times.

Here it can be seen that DF_{RK} is significantly slower than the other tested distance functions. When all nodes are placed in sequence, the maximum motion duration is equal to the created nodes times the control input duration. For an input duration of $0.05[s]$, this means that the RRT and DF_{RK} combination can reach, at average in an ideal situation, a state $6.5[s]$ away. This means that if the shortest motion between two states is longer than this duration, it cannot be found within the time frame of $1[s]$. However, since RRT is exploration oriented the maximum motion duration will turn out much lower than this ideal average duration. If the goal bias of RRT is set to 0.05, the possible motion duration comes closer to $.33[s](= 6.5*0.05)$ for DF_{RK}.

4-4-2 Greedy RRT

In this section the behavior of RRT and distance function combinations are considered when it moves towards a single state. In order to provide an insight in the workings of the distance function a single representative attempt will be discussed. Here RRT is used with the goal bias set to 1 for each of the different distance functions. It then proceeds to attempt to connect the start and the goal state by creating a tree consisting of 1000 nodes. Figure 4-13 and Figure 4-14 shows the RRT-trees created by the motion planner, which is the multi joint variant of the tree shown in Figure 2-1. Since it is not possible to show the complete tree in one graph, it has been displayed for each joint separately.
The start state is shown as a red X, the goal state is shown as a green X and the state which is closest according to DF_{vel} is shown as a blue X. The nodes of the tree are depicted as circles and the lines depict the motions which connect parent and child node. It can be seen for all tests that the goal state has not been reached.

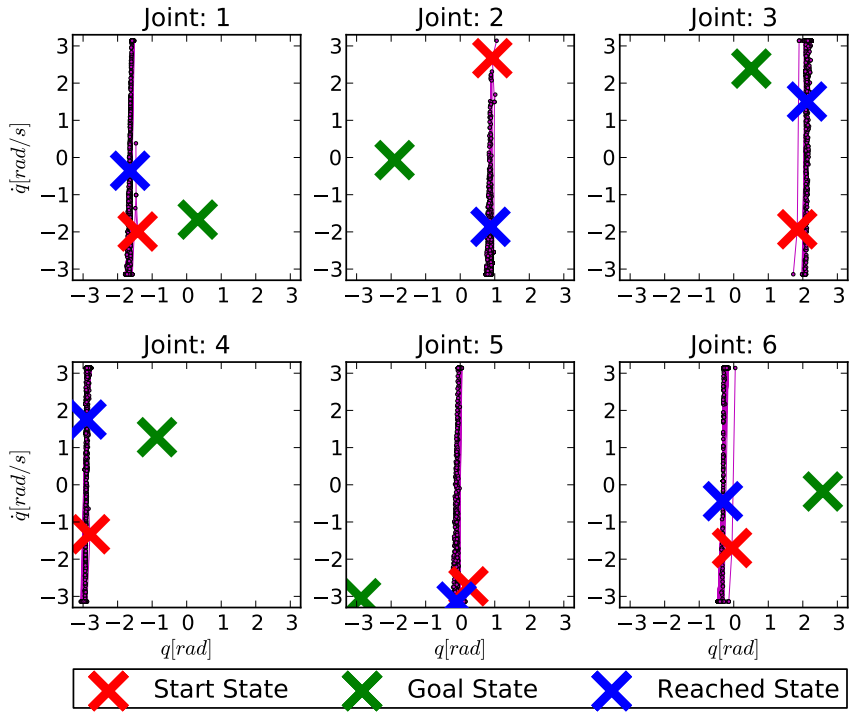


Figure 4-13: A RRT-tree consisting of 1000 nodes using DF_{vel} as the distance function and a goal bias of 1. Nodes are shown as circles and the connecting lines depict motions. Here a lack of movement towards the goal state can be seen.

In Figure 4-13 the 1000 nodes obtained by the RRT motion planner using the distance function DF_{vel} and a goal bias of 1 can be seen. The desired behavior is that the nodes gradually move closer to the green X from the red X. This is not the case, as the attained nodes maintain the same position as the starting node, while only varying in velocity. For DF_{vel} no significant movement can be observed towards the goal state.

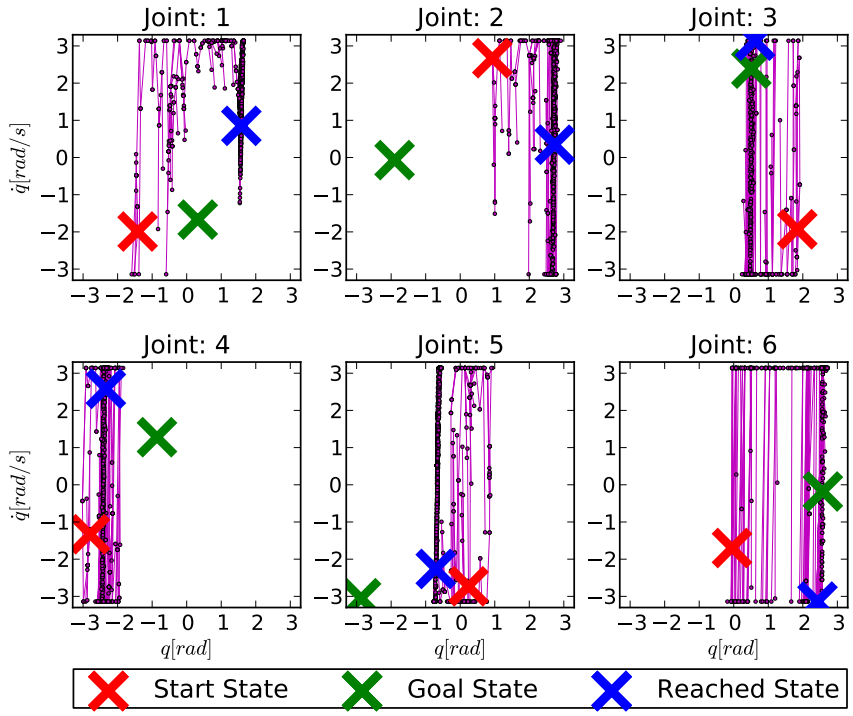


Figure 4-14: A RRT-tree consisting of 1000 nodes using DF_{EU} as the distance function and a goal bias of 1. Nodes are shown as circles and the connecting lines depict motions. There exists movement towards the goal, but the goal state is not reached.

The trees created by DF_{pos}, DF_{EU} and DF_{RK} all look similar. Therefore only the tree created with DF_{EU} is shown in Figure 4-14. These remaining trees are shown in Appendix B. Here it can be seen that a tree using the DF_{EU} distance function does gradually move closer towards the goal, even though the goal itself is not reached.
Keep in mind that in section 4-3 it is discussed that it is impossible for most joints to reach their desired state when the goal bias is 1.
Due to the complexity of the dynamics of the UR5 model not much else can be said on the basis of these figures.

4-4-3 Exploration RRT

The next two tests examine the exploration of the state space using the distance functions in combination with RRT. In this section the RRT motion planner is again used to test the different distance functions, however the goal bias is now set to 0.05. This change results in a motion planner more oriented at exploration.

The first test shows a single representative tree for the different distance functions, since again no attempt succeeded in reaching the goal. The second test discusses the coverage of the state space for multiple trees.

In Figure 4-15 and Figure 4-16 the results of the first test can be seen. Having a goal bias of 0.05 the RRT method is probabilistically complete, however with a tree consisting of only

1000 nodes none of the distance functions could reach the goal.

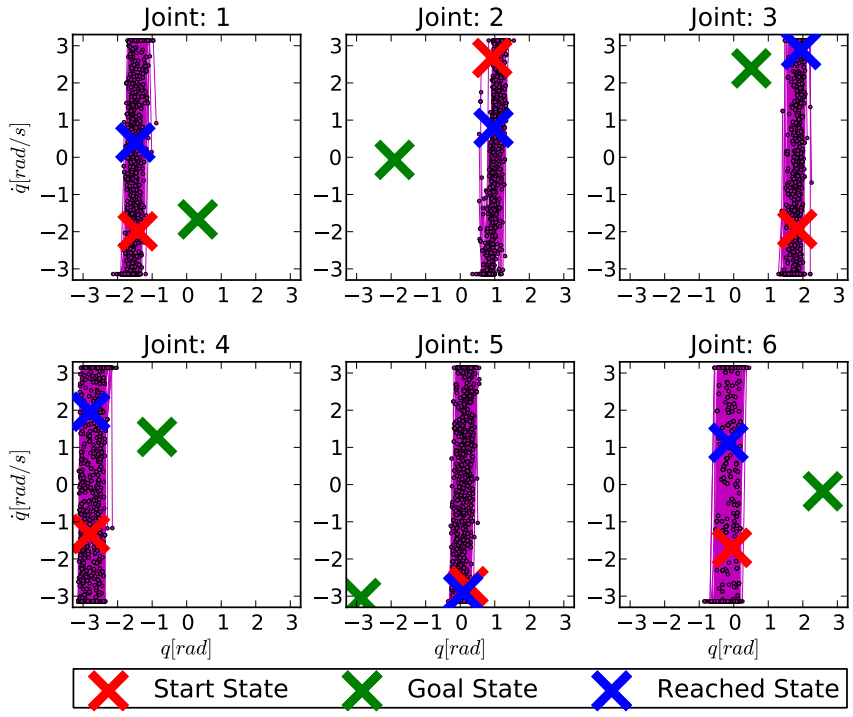


Figure 4-15: A RRT-tree consisting of 1000 nodes using DF_{vel} as the distance function and a goal bias of 0.05. Nodes are shown as circles and the connecting lines depict motions. Here it seems that the state space is not properly explored.

The goal of the low goal bias is to increase the exploration of the state space. This exploration can help evade the downfall caused by local minimums of a distance function. The first distance function which will be shown is DF_{vel}. The results of the test are shown in Figure 4-15. Here it can be seen that the DF_{vel} distance function does not explore positions far beyond the start position.

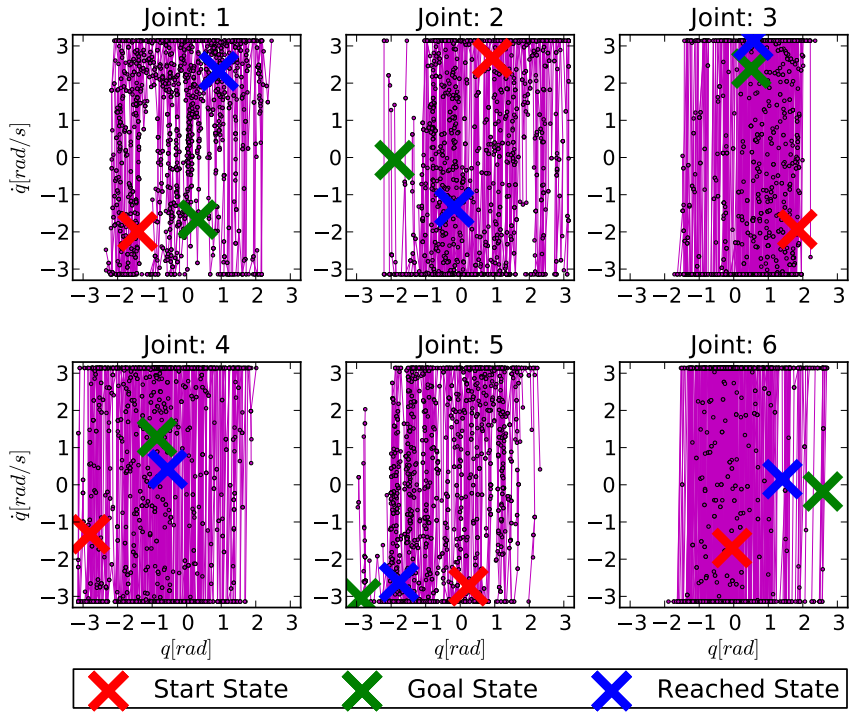


Figure 4-16: A RRT-tree consisting of 1000 nodes using DF_{EU} as the distance function and a goal bias of 0.05. Nodes are shown as circles and the connecting lines depict motions. It can be seen that the state space is properly explored.

The trees created with the other proposed distance functions look similar, thus only the tree of DF_{EU} is shown. These remaining trees are shown in Appendix B. As expected according to the correlation the exploration of the state space is significantly better with DF_{pos}, DF_{EU} and DF_{RK}. Not much more can be said, however, due to the complex nature of the system.

In order to show that the previous examples are the rule and not the exception, the second test is performed. The second test divides every dimension of the state space into 4 sections, thus creating 10^{12} bins. It then allows the RRT trees to grow with a goal bias of 0.05 and a random start and goal combination. This continues until all trees added 1000 nodes. At this point the number of unique nodes reached by the trees with each distance function can be counted. This process has been repeated 20 times and the results of this test are shown in Figure 4-17.

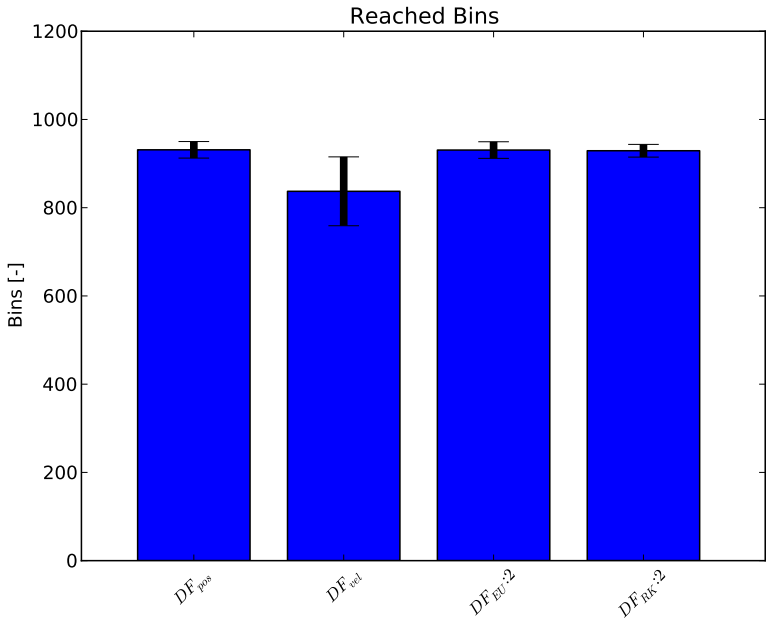


Figure 4-17: Coverage of a 1000 node RRT, in order to show state space exploration. This process was repeated 20 times to obtain the standard deviation which are shown as y-error bars. It can be seen that the mean exploration of DF_{vel} is lower than the other tested distance functions.

Here it can be seen that DF_{pos}, DF_{EU} and DF_{RK} all have a significant higher number of reached unique bins than DF_{vel}. However, it is interesting to note that DF_{vel} has a high number of reached unique nodes considering the low amount of position exploration observed in Figure 4-15.

4-5 Discussion

Throughout the literature the distance function is often called a metric. In this thesis the name metric is less used, since the term indicates symmetry, namely that the distance from a state A to B is the same as the distance from state B back to A. In the state space this is in general not the case. According to this information the term pseudo-metric is also acceptable. As to avoid this confusion the term distance function is used instead.

During the literature survey it was observed that the distance function DF_{vel} is often used as a baseline. However, in this chapter it is shown that this distance function has no strong correlation to motions in the state space and subsequently has a bad performance in the RRT environment. Therefore several simple distance functions have been proposed which are better suited as a baseline for new distance functions.

The distance functions is used in order to place all nodes of a tree on an ordinal scale, with respect to the proximity to a certain state. Here only the closest node is of any particular interest. This means that the exact distance is of no importance. This is especially useful during the coding of the distance functions.

For example instead of calculating the absolute value of a distance, only the squared value has to be calculated.

In this chapter four distance functions have been introduced. These methods take the sum of the distance of each individual joint as their output. Another method is to take the maximum, namely the cost of the joint furthest away, as the output of the distance function. Using the maximum cost with the same dataset as has been used in subsection 4-2-2, the correlation of the distance functions DF_{pos}, DF_{vel}, DF_{EU} all increased. DF_{RK} has not been tested. At this point it is important to note that there are an infinite amount of different distance functions possible. These distance functions, however, will not be discussed further in this thesis.

In order to verify the distance functions, which is defined as the cost to go from state to state, their results have been compared to the cost of actual motions in the state space. These motions were not necessarily optimal movements between two states, but since they are motions in the state space they do define the cost to go. A further verification of the distance functions can be performed by comparing them to the cost of motions obtained in a different method. This is discussed later in the thesis, more specifically in section 6-2.

In section 4-3 it was shown how accurate a distance function needs to be in order to correctly reach the target state. Notice here that only a single joint is shown, whereas a robot ordinarily consists of multiple joints. This means that a simple distance function in combination with a greedy RRT motion planner will be even more unlikely to reach a targeted state. The desire to create a perfect distance function for a motion planner, however, results in a catch-22 situation, as the motion planning problem itself needs to be solved in order to solve this sub-problem.

4-6 Conclusion

In this chapter several simple distance functions have been introduced for comparison with the simple but commonly used DF_{vel}. The various proposed distance functions have been verified and tested using the UR5 robot model and the actual RRT motion planner. Here it was shown that a distance function which uses the dynamics of a complex system is significantly slower than a distance function which does not.
As is shown the various distance functions are significantly better than DF_{vel} concerning most aspects. It is seen that a distance function with a low correlation, in comparison with the cost of an actual movement, also performs badly in a RRT environment.
The distance function with a high correlation reached closer to the desired state, however they as well did not manage to reach the targeted states within a reasonable number of attempts using the RRT environment.
Thus it can be seen that using a simple distance function in RRT with a random controller will not result in the finding of a desired motion from a set state to another within a reasonable time frame.

Chapter 5

Steering Function

The Rapidly-exploring Random Tree (RRT) [1] methods main characteristic is that it explores the state space. A motion planner, however, mainly needs to reach its goal. In chapter 4 the methods were discussed which determine which node of the tree needs to be expanded to best reach a target. Up until this point the expansion itself was based on a random control input. Using a random control input is probabilistically complete for extensive time durations. It is, however in practice undesired to let a motion planner run for an extended duration of time.

A basic idea to speed up the motion planning process is to use multiple random inputs and then using the distance function to determine the best newfound state. This, however, makes it so that the planning method is potentially no longer probabilistically complete [9]. A possible solution to this problem is discussed in Appendix C.

An actual viable method to speed up this expansion is by means of steering functions. A steering function is a method which when provided with a robot model, a start and a goal state, is able to connect these two states. The output of a steering function is the control input and the durations of these inputs needed to perform the movement between the two states. The steering function is able to find a solution, because it simplifies the actual motion planning problem. However extensive computation time of a steering function, in combination with a non optimal path, provide a serious downside. Before this thesis there was one likely candidate which could be used as a steering function, this candidate is called Time Optimal Path Parameterization (TOPP) [18]. TOPP uses a path and defines the time at which the configurations in this path are reached. Doing this it is able to transform the given path into a motion within the velocities and force/torque bounds. TOPP is discussed in section 5-1. There are other trajectory planners available besides TOPP [19, 20, 21], however none have been found which have a standalone library available. Without such a library the implementation of these methods would be significantly more difficult and they are thus not suited for this thesis. During this thesis another steering function was found which will be discussed in section 5-2. This steering function is called PID-Connect and it is able to find the necessary motion by using a PID-controller in order to reach a state with zero velocities from both the start and the goal state. The testing of the steering functions will be done using a UR5 robot arm model.

5-1 TOPP

The Time-Optimal Path Parameterization library (TOPP) [18] is a library which as the name suggests parameterizes time for a path in order to change the path into a time optimal movement. In order to perform this parameterization TOPP needs to have a path and a model of the system for which this optimization is done. The TOPP algorithm[1] obtains this path by using a third order polynomial to interpolate between a start and goal state. This path is a combination of subsequent configurations and contains no information at all about time. This is were TOPP itself comes into play. TOPP will parameterize the time needed to move between the subsequent configurations of this path. Using the distance between the two configurations and the time needed for this motion allows for the calculation of the system velocity. When moving between multiple configurations the velocity might change allowing for the calculation of the acceleration. The change in acceleration along with the dynamics of the robot itself allow for the calculation of the input controls of the robot. TOPP thus needs to define the time needed to travel from configuration to configuration while considering the velocity and input control constraints imposed by the provided system. When TOPP is successful a time is determined for all the subsequent configurations and a motion has been created, which can be executed by the defined system.

5-1-1 TOPP-RRT

Due to the complex nature of kinodynamic systems it is not always possible to move along a predefined path. This is also the case for the interpolated path used for the TOPP algorithm. It is however inadvisable to not use the method due to this potential inability. In order to remedy this preset path problem TOPP can be used in combination with RRT. Due to the combination of RRT and TOPP, if a movement exists along a set of paths which consists of an interpolation of multiple states, this motion will be found. If this motion does not exist it does not mean that there is no movement possible from the start state to the goal state. In order to verify the effectiveness of the motion finding of TOPP-RRT a test was run. In this test a RRT based motion planner has been set up which uses TOPP as a steering function and a Locally Weighted Projection Regression (LWPR) [22] model of TOPP as a distance function. The method of LWPR is discussed in Appendix D. The motion planner differs from RRT in that it does not have a goal bias which determines when to connect to the goal, but attempts to connect to the goal from every newly created node. The planner is allowed to attempt the creation of 5 new nodes, if after these attempts the start has not been connected to the goal the motion planner fails. In total the motion planner was used 1000 times in order to verify the computation time, the motion time and whether or not the motion planner could find a solution at all. The results are shown in Table 5-1. Here the computation time is the time needed for the algorithm to find a motion and the motion time is the time it takes for a robot to perform this motion. These values have been obtained on *ubuntu 12.04 LTS x64*, with 3.8 *GiB* memory, *Intel Core2 Duo CPU T9600 @ 2.80GHz * 2* processor and *Quadro FX 770M/PCIe/SSE2* graphics card.

From the table it can be seen that the time it takes to calculate the motion is longer than the time it takes to perform the motion on average. This means that the next motion cannot be

[1] https://github.com/quangounet/TOPP

Time	Mean $[s]$	SD $[s]$
Motion Time	1.64	1.10
Computation Time	13.92	11.75

Table 5-1: Motion time and computation time of the TOPP-RRT method. Of the 1000 attempts 842 attempts were successful.

calculated during the current motion, thus TOPP-RRT cannot be used for a robot "on the go". In Figure 5-1 this would mean that the motion from II to III is not yet known when II is reached from I. In the state space it is also not possible to maintain a state, which means that the entire movement needs to be found before the motion is started.

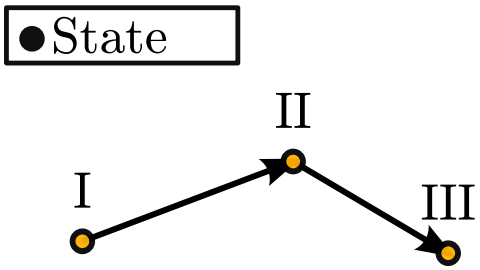


Figure 5-1: When the calculation of the motion from II to III takes longer than the actual motion from I to II, the movement cannot be planned on the go.

In order to further understand the TOPP steering function a motion created with this function is shown in Figure 5-2. This motion has been created by providing TOPP with a random start and goal state until it succeeded in creating a motion. Here effort is the control input defined by TOPP.

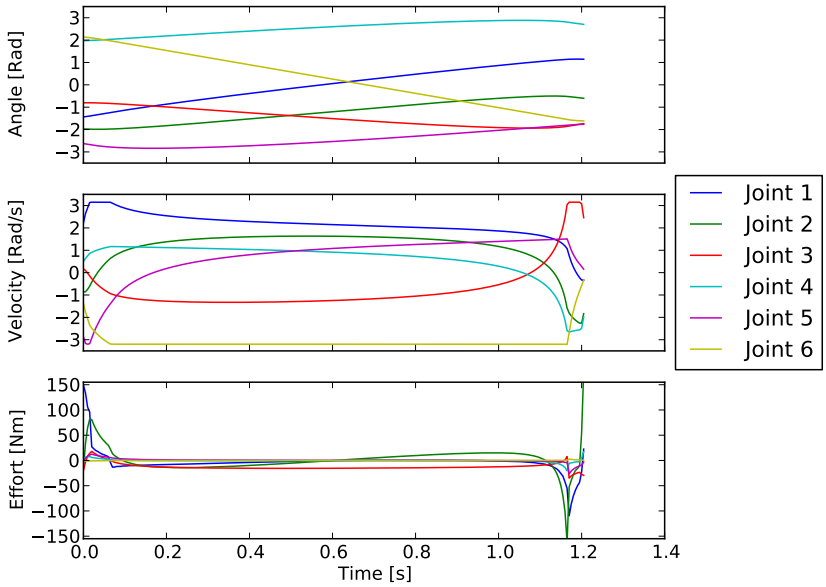


Figure 5-2: Actual movement between two states using the TOPP method on the UR5 robot model.

There are three downsides of TOPP that need to be clearly stated. The first downside is that it uses a direct interpolation between the two states in order to determine the subsequent angles. The second is that even though TOPP finds an optimized motion along this path, this motion might not reach the desired state since the reached velocities are not close enough to the target velocities. The third downside can be seen in the figure as the velocity and the acceleration limits are not adequately enforced. The second and third downside can be negated by making TOPP more accurate, however this also makes the calculation time longer.

5-2 PID-Connect

For the PID-connect steering function to be understood, the PID-controller needs to be explained, this is done in subsection 5-2-1. After this the PID-connect steering function can be properly explained in subsection 5-2-2.

5-2-1 PID-Controller

The Proportional-Integral-Derivative Controller [23], or PID-controller, is an often used feedback controller for many different industrial processes. This controller is used to stabilize a system and consists of proportional, integral and derivative elements.
The proportional element applies a control input proportional to the error between the target and the current state. The integral element applies a control input proportional to the sum of this error up to the current state. The derivative element applies a control input proportional

to the derivative of the current error. The weights assigned to the different elements is shown in Appendix E.

Another well known feedback controller, besides PID, is the Linear Quadratic Regulator (LQR)[24]. In order to use this controller a quadratic cost function needs to be defined and a linear approximation of the dynamic equations needs to be obtained around the point of interest. This controller can then be used to minimize this cost function. However, instead of manually creating the explicit dynamics equations the Rigid Body Dynamics Library (RBDL)[2] is used to define these equations using the model provided by the Unified Robot Description Format (URDF)[3]. This means that the explicit dynamics equations are not available, and the LQR controller can thus not be used.

5-2-2 PID-Connect

The PID-controller minimizes the error between the current position and a specific configuration over time. When this controller is used in the state space this minimization over time results in a state with zero velocities. The possibility to connect a random state to a state with zero velocities forms the basis of the PID-connect steering function. The PID-connect steering function is able to move between two random states in the state space. It does so by moving forwards in time from the start state to a state with zero velocity, and it also moves backwards in time from a chosen goal state to this same target state with zero velocity. The start state is considered connected to the target state, if the distance between the target state and the state reached by the PID-controller from the start state is smaller than a set connection distance. This distance is calculated using the EU_{vel} distance function. The goal state is considered connected under the the same conditions. When a properly tuned PID-controller is able to propagate both the start and the goal state to a single target state with zero velocity, it is able to connect the start state to the goal state.

A basic way of choosing the target state with zero velocity, is to calculate the average of the start and goal positions and setting this as the target position. Another method to determine the connection state is to first determine the fastest way for either the goal or the start state to reach zero velocity and then use this resulting position as the target state with zero velocity. Using the second method it can be seen that PID-connect can generate a decent motion from start to goal, a potential motion is depicted in Figure 5-3. Here the green edges are created by propagating backwards in time and the red edges by propagating forward in time.

[2] http://rbdl.bitbucket.org/(August 10, 2016)
[3] http://wiki.ros.org/urdf(August 10, 2016)

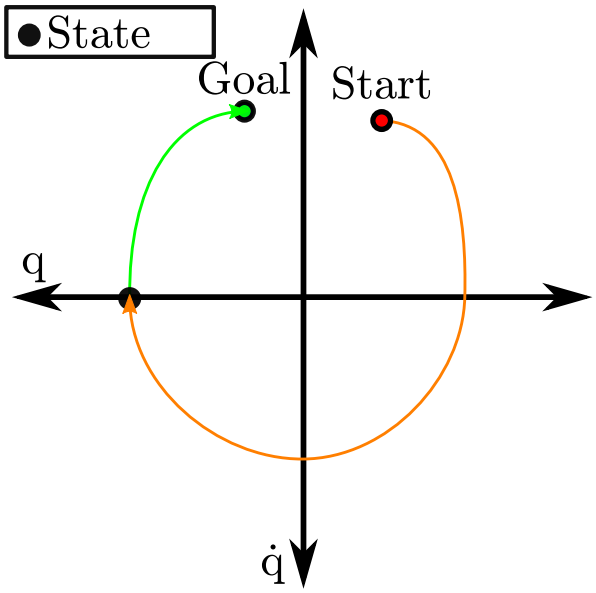


Figure 5-3: Example of a good movement using PID-connect.

This steering function should, however not be considered to be anywhere near optimal, due to its need to always move to a state with zero velocity. A depiction of this problem can be seen in Figure 5-4. Here the gray edge depicts the ideal path, which does not need to pass through a zero velocity state.

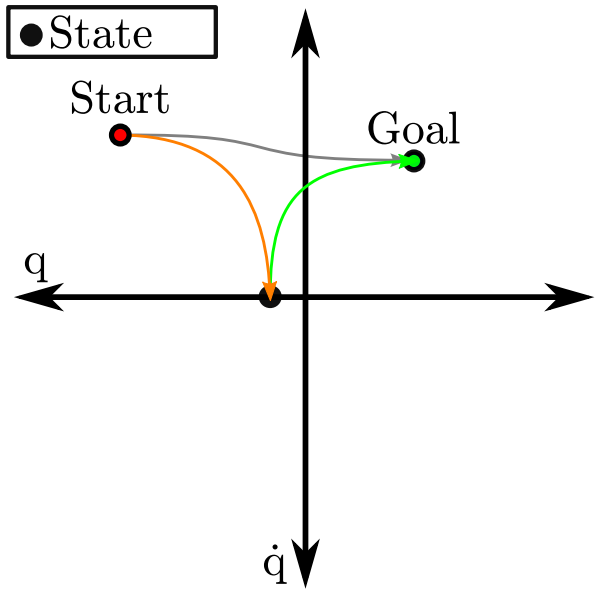


Figure 5-4: Example of a bad movement using PID-connect.

The PID-connect motion planner can be used and tested on an actual system, in this case the UR5 robot arm is again used. The target state has been created by taking the mean of the start and goal position and setting all its velocities to zero. The necessary connection distance, which is calculated with EU_{vel}, is set to 0.05. In order to reach this target state from both the start and the goal the PID-controller needs to be properly tuned. The PID-connect steering function has then been tested on 1000 randomly chosen start and goal state combinations, the results of this test can be seen in Table 5-2. Where the motion time is the time needed for the robot to perform the motion and the computation time is the time needed for the algorithm to compute the motion. These values have been obtained on *ubuntu 12.04 LTS x64*, with 3.8 *GiB* memory, *Intel Core2 Duo CPU T9600 @ 2.80GHz ÃŮ 2* processor and *Quadro FX 770M/PCIe/SSE2* graphics card.

Time	Mean $[s]$	SD $[s]$
Motion Time	2.52	0.67
Computation Time	0.02	0.01

Table 5-2: Motion time and computation time of the PID-Connect method. All 1000 attempts succeeded.

From this table the main benefit of PID-connect can be seen, namely that the motion time is significantly longer than the computation time. This means that it can generally calculate the following motion during the execution of its current motion. In Figure 5-5 this means that the motion from II to III can be calculated during the motion from I to II. This means that it can be used to solve the motion planning problem within a reasonable time frame.

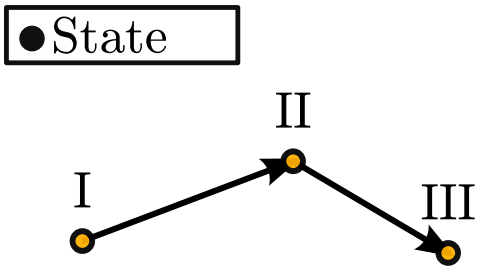


Figure 5-5: When the calculation of the motion from II to III takes shorter than the actual motion from I to II, the movement can be planned on the go.

In order to further understand the PID-connect steering function a motion has been created with this method. This motion is shown in Figure 5-6. This motion has been created by providing PID-connect with a random start and goal state. Here effort is the control input defined by the PID-controller.

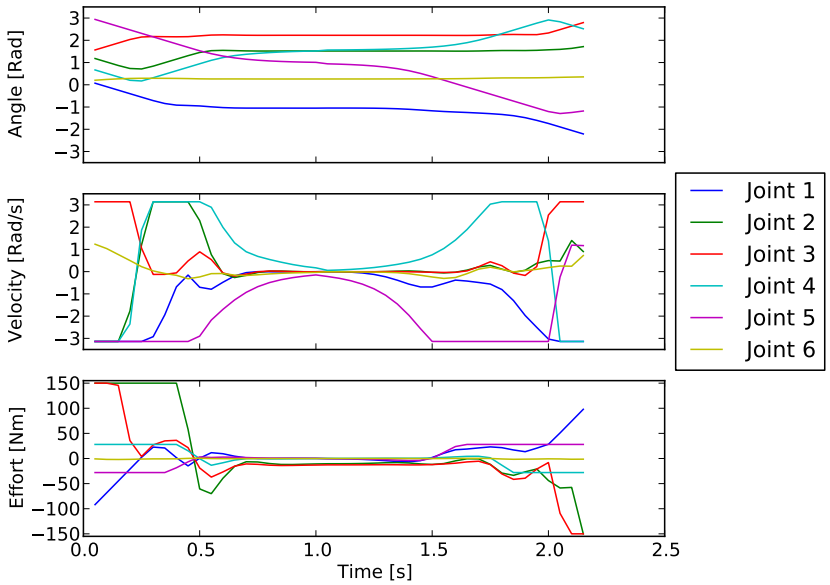


Figure 5-6: Actual movement between two states using the PID-connect method on the UR5 robot model.

The two downsides which need to be stated about PID-connect can be seen clearly in the previous figure. The first is that planned motion is not optimal. The second is that the motion created with PID-connect needs to have a state with zero velocities.

5-3 Discussion

In the literature several methods are found which can potentially be used as steering functions, an example is TOPP. These methods are generally not fast and not optimal. In this sense

PID-connect is an improvement to such methods as it is non optimal, but fast.

The method used in TOPP, which adds velocities and torques to a predefined path is called time scaling. TOPP interpolates between two states to obtain a path. The main drawback of time scaling is that this path is not necessarily traversable by the robot for which it is intended.

The PID-connect method has three main drawbacks.
The first is that the final motion needs to contain a state with zero velocity. This results in a motion between two states which is most likely not optimal. If the initial or final state already has no velocity there is no other state needed, along the motion, which also has zero velocity.
The second drawback of PID-connect is that the manual tuning of the PID-controller can be time consuming. Some software packages exist which can be used to tune the PID-controller automatically ensuring consistent results, diminishing this downside.
The third drawback is that the PID-controller is not concerned with applying the control input efficiently. This means that if the distance is large the control input will be maximum, even if the velocity is already maximal and the control input only needs to maintain this velocity. This problem can be solved if the obtained motion is fine tuned, which is shown in Equation 5-1 for a positive velocity. Here the inverse dynamics are used to calculate the needed control input to maintain the maximal velocity for every state and corresponding zero acceleration.

$$\begin{aligned} \ddot{q} &= ForwardDynamics(model, q, \dot{q}, \tau) \\ (\dot{q} = \dot{q}_{max} \wedge \ddot{q} > 0) &\Rightarrow \ddot{q} = 0 \\ \tau &= InverseDynamics(model, q, \dot{q}, \ddot{q}) \end{aligned} \tag{5-1}$$

The basic principle of PID-connect uses a working configuration space controller to reach a zero velocity state both forward and backwards in time. This means that controllers other than the PID-controller, can be used as well. An example of a different controller is the LQR controller. The LQR controller, however needs to have a description of the system in the form of a set of linear differential equation. Automatically obtaining this model from the URDF file is not yet possible.

5-4 Conclusion

Two steering functions have been introduced in this chapter, both of which are able to find a motion between two given states under strict assumptions. On their own neither of these steering functions is guaranteed to find a motion between two states.

The combination of RRT and these steering functions results in a set of motion planners able to find motions in the state space, within an adequate time frame. This does not mean that an optimal solution is likely to be obtained.
The TOPP steering function is slow, but obtains decent results if it is able to find a motion. The PID-connect steering function is fast and finds a non-optimal solution reliably. Since both methods fail to obtain an optimal solution the speed an reliability with which they find a motion become the defining factors. This makes PID-connect the better steering function. When PID-connect is used in combination with RRT it can be used on the go.

Chapter 6

Discussion

In this chapter the results of the tests performed in this thesis are discussed. This discussion is divided into three different sections. In section 6-1 the used robot is discussed, more specifically whether the obtained results are also viable for other types of robots. Limitations to the test and verification of the distance function are discussed in section 6-2. Throughout this thesis the aspects of optimality and collision detection are not considered, in section 6-3 they are discussed shortly.

6-1 Different Robot Types

All tests in this thesis have been performed using the UR5 robot model. This leaves the question whether the results obtained from these tests are viable for other systems. It is likely that for most fully controllable robots the results will be similar. However there are two robot types which require further inspection are parallel robots and underactuated robots.

The problem with using the Unified Robot Description Format and comparable notations is that this notation only allows for tree like robots. This means that every robot is made up of links. It starts with a parent link to which zero or more child links are attached. This can then continue until a robot is fully described. There is, however, no method in which a cycle can be created using this description format, nor is it possible for two parents to have the same child link. Thus in order to plan motions for parallel robots a different type of notation needs to be used.

The problem of underactuated robots and specifically trivially underactuated systems is that they are significantly different from a completely controllable robot arm. The standard definition of a trivially underactuated robot is that they have a lower number of actuators than degrees of freedom. The steering functions described in this thesis might not be able to connect all combinations of random states for these underactuated systems. RRT itself has no problem connecting two states even for underactuated systems, since it is probabilistically complete. The difficulty for the motion planning problem is whether a motion can be found

in a short time frame. The duration of the calculation again increases for more complicated robots.

Something that has not been utilized for the UR5 robot model, but might be useful, is the cyclic nature of every joint. Due to this cyclic nature of the joints every position can generally be reached from two sides, both clockwise and counterclockwise. Motions with lower cost might be found if this cyclic nature is utilized. It is important to keep in mind, when implementing this cyclic nature, that the joints are not unbounded. Every joint from the UR5 robot can perform a rotation of $4\pi[rad]$ from the lower bound until reaching the upper bound. In the UR5 robot model which has been used in the tests this has been limited further to $2\pi[rad]$, in order to decrease the size of the planned for state.

6-2 Distance Function Test & Verification

In section 4-2 the different distance functions have been verified by comparing their estimation of the cost, to the actual cost of a motion in the state space. Here some distance functions showed a strong correlation to the actual cost, however in combination with RRT and a random controller these distance functions are not able to find a motion between two given states within a reasonable time frame.
This means either that the correlation test of the distance function is faulty or that a different controller is needed. First the test will be discussed and then the use of a different controller.

Since the distance function is defined as the cost to go from state to state, the correlation between the distance function and the time needed for an actual motion in the state space is a good measure. The problem might reside in the choice of the actual motions for the comparison. Determining these motions is not as dependent on the time as the actual motion planning problem. However, currently there is no way to exactly connect two states optimally or to determine whether a motion is optimal.

6-3 Collision Detection and Optimality

Due to time constraints both optimality and collision detection have not been actively incorporated for the motion planners created in this thesis. They should, however, still be mentioned if only briefly. The first step is to define both optimality and collision detection in the motion planing environment.

A motion planner is considered optimal if it returns the motion between two given states which has the lowest possible cost. Collision detection can be used in a motion planner to detect whether the robot has a collision with either itself or another object in the state space. If a program is found to perform collision detection it can easily be implemented in the modular motion planner, since it is also implemented for the configuration space version of the open motion planning library.
In path planning many different heuristics have been implemented in order to decrease the calculation time of optimal path planners with collision detection [7]. The motion planners created in this thesis can theoretically find the optimal solution and evade obstacles. This is the case due to the probabilistic completeness of the planners, meaning that any possible

motion between two states can thus be found if the motion planner is given enough time. The problem with this type of motion planning is again finding the right motion between two states within a reasonable time frame.

Chapter 7

Conclusion

Robots perform movements between states, these motions need to be planned. The main goal of this thesis is to design a planning infrastructure with minimal input, which can solve the motion planning problem for various robots. This all within a reasonable time frame and without preprocessing.

The created modular motion planner is made up of open source components. The specific motion planner algorithm are modeled after the Rapidly-exploring Random Tree (RRT) [1] and RRT-connect. This modular motion planner is probabilistically complete, meaning that it can find the solution to the motion planning problem in the state space if it is given enough time. This does not provide a guarantee that a solution motion is found within a reasonable time frame. In practice it is shown that for a complex robot a solution cannot be found even when the modular motion planner is run for extended durations.

The RRT is heavily dependent on the used distance function, which is used in order to determine the cost to travel from one state to another. In the literature a euclidean based metric is commonly used. This distance function has been tested, along with several other proposed simple distance functions. These tests show that the often used DF_{vel} has a low correlation with the actual cost of motions and should thus not be used as a distance function. The other distance functions have a strong correlation to the actual cost of motions in the state space. However, their behavior in the RRT environment did not result in the finding of a solution to the motion planning problem in a timely fashion. This also means that a strong correlation of a distance function to the actual cost of a motion does not ensure the usability of a distance function in the state space.

In order to find a connection between multiple states two steering functions have been used. Steering functions are able to connect two states exactly, however they use a crutch to do so. The two created steering functions are Time Optimal Path Parameterization (TOPP) [18] and PID-connect.
The crutch which TOPP uses is that it interpolates a path between two states, it then performs time-scaling to this path. This means that it cannot always find a solution motion, as the interpolated path might not be traversable by the robot.

The steering function PID-connect connects two states by moving from both start and goal states to a state with zero velocity. The crutch of PID-connect is less problematic if a start or goal state is chosen which already has zero velocity. The use of the steering functions greatly speeds up the finding of the motion planning problem solution in state space according to the tests done with the UR5 robot model.
This research successfully finds a solution to the motion planning problem in a reasonable time frame ($0.02 \pm 0.01[s], n = 1000$) using the combination of PID-connect and RRT.

Chapter 8

Recommendation

During the course of this research many observations have been made. Using these observations new directions for future research are recommended below.

- Complex Distance Functions
  During this thesis simple distance functions have been covered. Better results might be obtained if more complex distance functions would be utilized. Complex distance functions should consider the system as a whole, instead of considering the different joints separately. An example of a complex distance function is the minimum time distance function [17]. This distance function estimates the minimum time a robot needs in order to reach another state. It calculates this minimum time for every joint and the distance functions result is the maximum of the minimum time durations. A possible improvement to this algorithm can be to incorporate torque/force limits, as the current implementation of this distance function assumes constant accelerations.

- Alternative to PID-connect
  The steering function PID-connect uses the configuration space oriented PID-controller. There are many more controllers which are also able to connect a random state to a specified zero velocity state. Future research can go into finding these controllers and implementing them in the steering function. An example of an already stated controller which can take the place of the PID-controller is the LQR controller. The motions directly resulting from this type of steering function will not be time optimal. In order to bring these solutions closer to optimal TOPP can be used in order to improve the movement along the given path. Combining the steering functions PID-connect and TOPP should result in reliably obtaining fast motions.

- Velocity Parametrization
  During this thesis there was no method found which attempts to minimize the effort of a motion without increasing the duration. A description of a possible method which would be able to do this is as follows.
  This method essentially consists of changing the velocity vs time graph, while considering the boundaries of position, velocity and force/torque. The start and the goal state

are given, which means that the begin and endpoint of the graph are known. In order to reach the final state from the start state, the area of the graph, which is distance traveled, needs to be constant.
This method can be used if there exists a joint without maximum position, velocity or force/torque. At this point it increases the velocity of a section of the motion where no boundary is met, and decreases the velocity of the other states along the motion. This is done while maintaining a constant graph area and respecting the boundaries.
In doing so the total effort for the changed joint of the robot should decrease and the path along which the robot moves changes. This can be done for every joint.
With a slight adaptation this method should be able to define a motion between two states, if it exists within the given time frame. If this method is combined with time scaling faster motions can be achieved as well.

- Offline Motion Planning
  All motion planning methods are slow to find the needed motion. A possible solution which shifts the problem elsewhere, is provided by the Probabilistic Road Map (PRM) [6] for the the configuration space. Instead of calculating the motions needed to travel between states when they are needed, PRM calculates these motions beforehand. This means that this method creates a road map which connects many different states together. This lowers the difficulty of the motion planning problem as the start and goal states only need to be connected to the road map, which can be done using a distance function and a steering function. The creation of the road map itself does take a considerable amount of time.

- State Space Demarcation
  A next step can also be to attempt to find the optimal motion between two states. At the moment the entire state space is searched in order to find the desired motion, but once an initial motion has been found the searched state space can be limited. This initial motion has a duration and only faster motions between the start and goal state are interesting. The limitation of the state space can be performed by defining the states which are not reachable from both start and goal states within the current maximum duration and removing these from the searched space. A more advanced method has been created in the configuration space called Batch Informed Trees (BIT*) [25]. The method used in BIT* defines the configurations which are reachable from the start configuration and from which the goal configuration is still reachable within the current maximum duration. All configurations which are not reachable, do not need to be searched. Currently there is no comparable method in the state space.

Bibliography

[1] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," *Computer Science Dept., Iowa State University*, pp. 98–11, 1998.

[2] J. Reif, "Complexity of the mover's problem and generalizations," *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pp. 421 – 427, 1979.

[3] Ioan A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

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

[5] J. Barraquand and J. Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles," *IEEE International Conference on Robotics and Automation*, pp. 2328–2335, 1991.

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

[7] W. Spierenburg, "Motion Plannning Literature Survey," 2016.

[8] J. Kuffner, J.J. and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation*, vol. 2, no. ICRA, 2000.

[9] T. Kunz and M. Stilman, "Kinodynamic RRTs with Fixed Time Step and Best-Input Extension Are Not Probabilistically Complete," *Algorithmic Foundations of Robotics XI*, vol. 107, no. Springer Tracts in Advanced Robotics, pp. 233–244, 2015.

[10] M. Rickert, "Efficient Motion Planning for Intuitive Task Execution," Dissertation, Technische Universitat Munchen, 2011.

[11] K. Ahnert and M. Mulansky, "Odeint - Solving ordinary differential equations in C++," *CoRR*, vol. 1389, pp. 1586–1589, 2011.

[12] R. Diankov, "Automated Construction of Robotic Manipulation Programs," Ph.D. dissertation, Carnegie Mellon University, 2010.

[13] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots : a survey based on user feedback," *CoRR*, vol. abs/1402.7, pp. 1–15, 2014.

[14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS : an open-source Robot Operating System," *ICRA workshop on open source software*, vol. 3, no. 3.2, p. 5, 2009.

[15] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," *Proceedings 2009 ICRA. IEEE International Conference on Robotics and Automation*, pp. 2859–2865, 2009.

[16] L. Jaillet, J. Hoffman, J. Van Den Berg, P. Abbeel, J. M. Porta, and K. Goldberg, "EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2646–2652, 2011.

[17] T. Kunz and M. Stilman, "Probabilistically Complete Kinodynamic Planning for Robot Manipulators with Acceleration Limits," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[18] Q.-C. Pham, "A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm," *CoRR*, vol. abs/1312.6, 2013.

[19] J. Johnson and K. Hauser, "Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path," *IEEE International Conference on Robotics and Automation*, no. 1987, pp. 2035 – 2041, 2012.

[20] D. Wang and W. Zhang, "Trajectory optimization and tracking control of mobile robots based on DMOC," *Chinese Control Conference*, vol. 33, no. 61203064, pp. 2422–2427, 2014.

[21] S. Rakshit and S. Akella, "A trajectory optimization formulation for assistive robotic devices," *IEEE International Conference on Robotics and Automation*, pp. 2068–2074, 2016.

[22] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental Online Learning in High Dimensions," *Neural Computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[23] M. Araki, "PID Control," *Control Systems, Robotics, and Automation*, vol. II, pp. 1–23, 2002.

[24] J. B. Burl, *Linear Optimal Control: H(2) and H (Infinity) Methods*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[25] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "BIT *: Batch Informed Trees for Optimal Sampling-based Planning via Dynamic Programming on Implicit Random Geometric Graphs," *CoRR*, vol. abs/1405.5, 2015.

[26] W. L. Goffe, G. D. Ferrier, and J. Rogers, "Global optimization of statistical functions with simulated annealing," *Journal of Econometrics*, vol. 60, no. 1-2, pp. 65–99, 1994.

[27] F. Gao and L. Han, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 2012.

Appendix A

How-To State Space OMPL

During this thesis OMPL and various planners have been adapted for use in the state space. This was done to simplify the future attempts of creating new motion planners. In this appendix the installation guide for this infrastructure will be provided in combination with some remarks. For this installation ubuntu 12.04 was used.

```
OMPL-1.0.0
(Guide: http://moveit.ros.org/wiki/OMPL/Add_New_Planner)

Make sure ROS is installed (In this case Hydro).
Download the OMPL source here, OMPLapp is not required:
    http://ompl.kavrakilab.org/installation.html
Navigate to the ompl-1.0.0-source directory.
Create the folders: "build/Release".
Navigate to the Release folder.
Run this command:
   $ cmake -DCMAKE_INSTALL_PREFIX=/opt/ros/hydro ../..
or this command for use with debugging:
   $ cmake -DCMAKE_BUILD_TYPE=Debug -DCMAKE_INSTALL_PREFIX=/opt/ros/hydro ../..
Then install the code:
   $ sudo make install
```

,This is the default installation of OMPL. The following sections of the guide detail the needed additions for OMPL to be used in the state space. Interesting to note is that OMPL also uses the term "state space", but this refers to the configuration space. Utilities for control based planning are provided by OMPL, which have been used as the basis for the state space variant.

```
RBDL 2.4.0

Download the most recent stable version of RBDL from:
    https://bitbucket.org/rbdl/rbdl/get/default.zip
Extract the downloaded file and in the main folder open CMakeLists.txt.
Set "RBDL_BUILD_ADDON_URDFREADER" to "ON".
Navigate to the RBDL "src" folder.
Create the folders: "build/Release".
$ cmake -D CMAKE_BUILD_TYPE=Release ../..
$ sudo make install
```

RBDL has multiple tutorials available which have been used in order to discern its proper usage. By default the URDF loader is turned off, but being able to use URDF files as input greatly simplifies the use of the motion planner infrastructure.

```
Eigen3

Install Eigen3 (C++ library) through the Ubuntu Software center.
$ cd /usr/include
$ sudo ln -sf eigen3/Eigen Eigen
```

The Eigen library is used by RBDL as a linear algebra library and is by default installed in */usr/include/eigen3/Eigen*. However, it is searched for in */usr/include/Eigen*. Hence a link is created at the required location which points to the installation location. at

```
Odeint

Download the Odeint C++ libraries from:
    http://headmyshoulder.github.io/odeint-v2/downloads.html
In this case it is called "headmyshoulder-odeint-v2-b816e93"
$ sudo cp -R
    /Location/Of/The/File/headmyshoulder-odeint-v2-b816e93/include/boost/numeric
    /usr/include/boost/
```

The boost version Hydro used in this infrastructure does not contain a version of Odeint and the infrastructure can thus not compile if it is not added. The usage of Odeint was clarified via the various tutorials available on their site.

```
URDF

An example urdf model for the UR5 can be obtained as follows:
create a storage folder
$ cd /Location/Of/older/
$ git init
$ git clone -b "hydro" https://github.com/ros-industrial/universal_robot
    src/universal_robot
Add the "ur_description" folder located at "/src/universal_robot/ur_description" to
    the ompl-1.0.0-Source folder
```

The most important file from this folder is *ur_description/urdf/ur5_robot.urdf*. However, adding the other files allows RVIZ to display the proper robot model.

```
Additions to OMPL-1.0.0

The following folders and files need to be added to the OMPL folder
../ompl-1.0.0-Source/demos: launch
../ompl-1.0.0-Source/demos/CMakeLists.txt
../ompl-1.0.0-Source/demos/WWF.cpp
../ompl-1.0.0-Source/FindRBDL.cmake
../ompl-1.0.0-Source/src/ompl/control/planners: wwf
../ompl-1.0.0-Source/src/ompl/control/SpaceInformation.h
```

These files and folders have been included with the thesis. They contain the changes and adaptations performed to OMPL itself in order to make it usable in the state space.
The files in the "launch" folder are used in order to run the various demos.
In the *CMakeLists.txt* file the compiler was instructed to create the new demos.
The file *WWF.cpp* instantiates the state space environment.
FindRBDL.cmake has been added to search for RBDL in the default installation folders, so RBDL can be used with the various demos.
In the folder "wwf" the source and header files of the state space planners are stored.
In *SpaceInformation.h* a function is added so the PID-controller weights, created in the demo, can be accessed in the planner.

```
The most used files are located at:

ompl-1.0.0-Source/demos/WWF.cpp
ompl-1.0.0-Source/src/ompl/control/planners/wwf
```

The *WWF.cpp* creates the main environment of the planning infrastructure, from here the state propagator is instantiated using RBDL, Odeint and the URDF file. The basic parameters of the environment can be changed here, for example time duration of the simulation, bounds of joints and efforts, used planner, gravity, start and goal states, etc.. At this point it needs to be ensured that parameters are passed through to the different planners. It has to be verified that no duplicate channels have been created.

The "planners" folder contains files concerning the control planners. The for this thesis created state space planners are located in the folder "wwf". Three planners have been created based on RRT, RRT-connect and PRM.
The RRT planner consists of the files *WWF.h* and *WWF.cpp*. Difficulties with this adaptation consist of rewriting control based functions so they work in the state space and acquiring the environment parameters.
The RRT-connect planner consists of the files *WWF_Connect.h* and *WWF_Connect.cpp*. There are two major changes in consideration with the RRT planner. The first is the addition of a second tree, which moves backwards in time. The second is rewriting the goal oriented movement so that it is guided towards the closest node on the opposing tree as opposed to the seed of that tree.
The PRM planner consists of the files *WWF_PRM.h* and *WWF_PRM.cpp*. In this planner many trees can be created, which explore both forwards and backwards in time and attempt to connect to trees in the opposing direction. The main difficulty is storing all made connections, checking whether the goal is connected to the start and then determining the solution motion.

```
Navigate to ompl-1.0.0-Source/build/Release.
$ cmake -D CMAKE_BUILD_TYPE=Release ../..
$ sudo make install
```

After the first installation the cmake files only need to be updated when a new planner is added. In other cases the code only needs to be compiled.

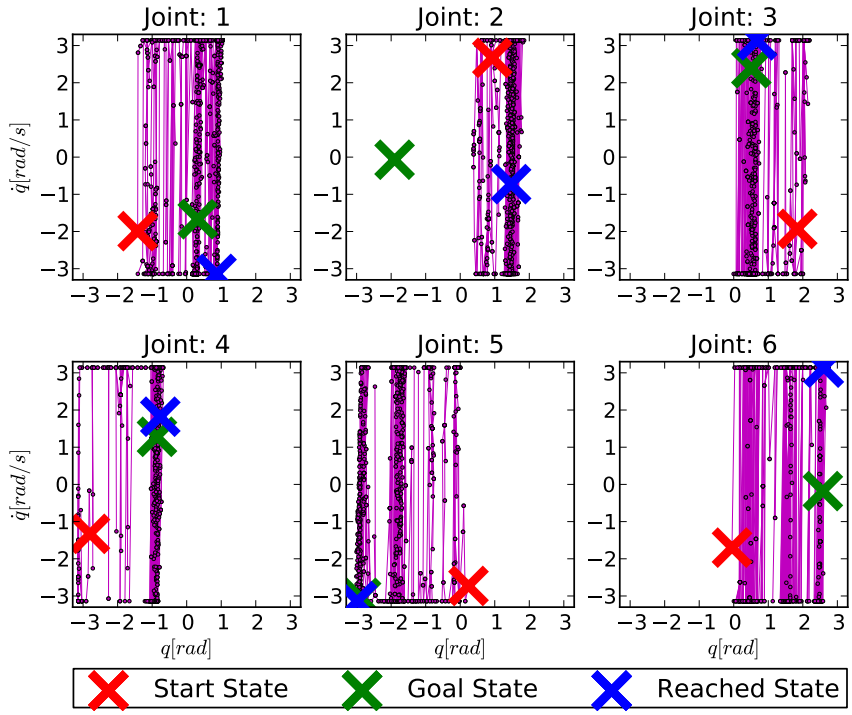
```
The planner can now be used by first adding the source:
$ source devel/setup.bash
followed by running the demo:
   $ roslaunch ompl demo_WWF.launch model:=/home/directory/of/the/urdf/model.urdf
for debugging run:
   $ roslaunch ompl demo_WWF_Debug.launch
       model:=/home/directory/of/the/urdf/model.urdf
for only the motion planner without RVIZ run:
   $ roslaunch ompl demo_demo.launch model:=/home/directory/of/the/urdf/model.urdf
```

In the demo a random start and goal state are chosen, for which the connecting motion is determined. When the *demo_WWF.launch* finds a motion it is shown using RVIZ. The *demo_WWF_Debug.launch* can be used if the code is changed in order to make the debugging process easier. When *demo_demo.launch* is run only the text based results are shown, without the RVIZ visualization.

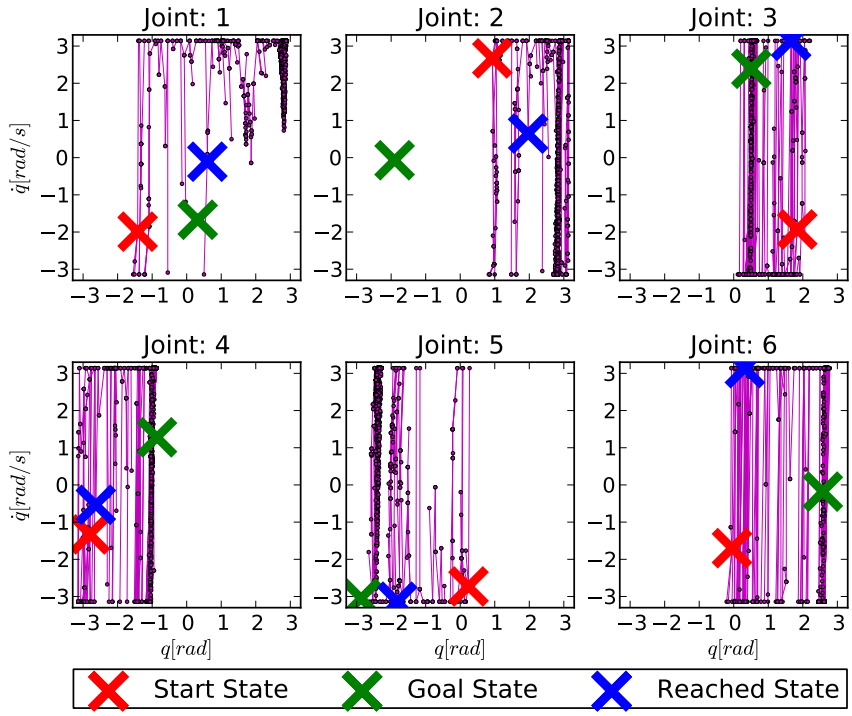
Appendix B

RRT Trees

In subsection 4-4-3 and subsection 4-4-2 the behavior of the different distance functions in combination with RRT is discussed. This has been done both with a goal bias of 1 as well as 0.05. Since some obtained trees looked similar it was unnecessary to show them all individually at that time. In order to be as complete as possible this appendix does show these trees in the following figures. The trees resulting from a combination of the distance functions and a greedy RRT are shown in Figure B-1, and in Figure B-2 the trees resulting from a more naive RRT.

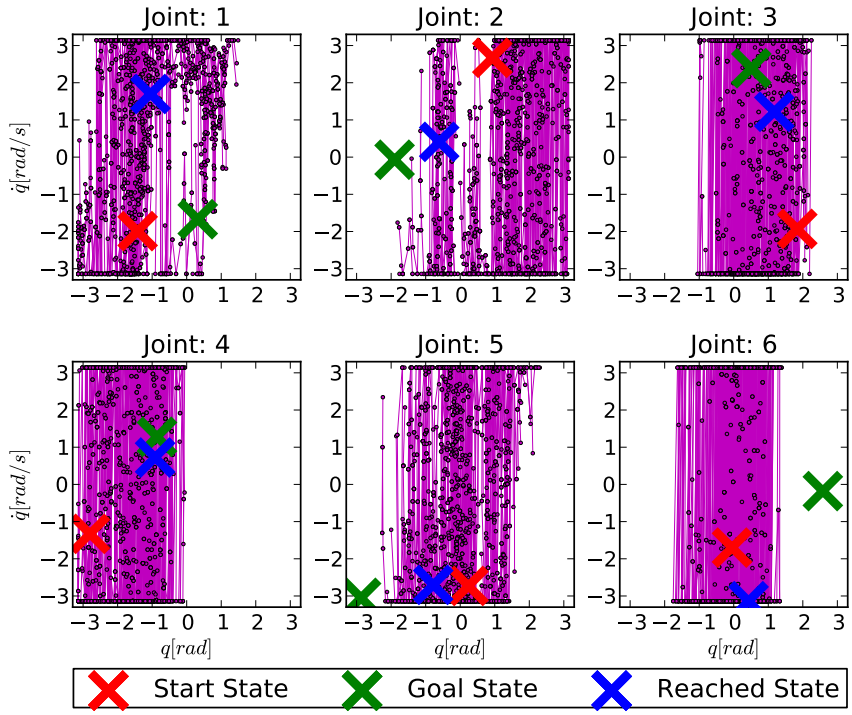


(a) using DF_{pos}

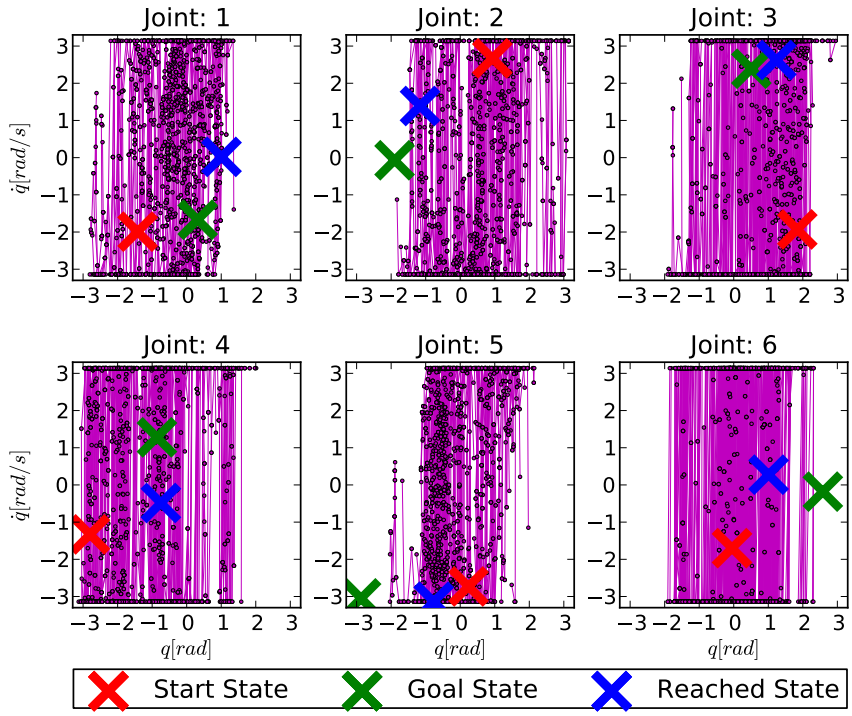


(b) using DF_{RK}

Figure B-1: A RRT-tree consisting of 1000 nodes using different distance function and a goal bias of 1. Nodes are shown as circles and the connecting lines depict motions.



(a) using DF_{pos}



(b) using DF_{RK}

Figure B-2: A RRT-tree consisting of 1000 nodes using different distance function and a goal bias of 0.05. Nodes are shown as circles and the connecting lines depict motions.

Appendix C

Control Input Selection

In chapter 5 a best of multiple control inputs method was discussed in order to improve the exploration speed of the motion planning algorithm. It was also said that this method is not probabilistically complete.
In order to mend this problem two algorithms have been investigated, namely Probable Selection and Simulated Annealing (SA) [26].
The method of Probable Selection does not always use the best of multiple control inputs, but also allows worse inputs to be chosen with a certain probability. SA differs in that a "temperature" regulates this chance. The temperature starts of high allowing many non optimal inputs and lowers over time, only allowing the optimal input. The SA algorithm is originally designed not to get stuck in local minimums and to be able to find the global minimum. This global minimum in motion planning is the movement connecting the start to the goal state.
The use of these methods increased the rate of exploration, however it did not succeed in creating an actual connection. Since these methods do not provide a connection no further effort was put in order to prove whether or not these methods are probabilistically complete.

Appendix D

LWPR

Steering functions are used to connect two states, this connection can be used to indicate the cost to go between these states. It would thus be interesting to see whether a steering function can be used as a distance function. A drawback for the use of TOPP in an actual planning environment is its long calculation duration. This means that TOPP itself cannot be used as an often used distance function. In order to reduce the calculation time at the moment of actual planning a type of lookup table can be created which removes the necessity of completely recalculating the output value using using the steering function. The method used to create this speed up for TOPP is Locally Weighted Projection Regression (LWPR) [22]. LWPR can be used to approximate non linear high dimensional functions. Its specific inner workings are complex and can be found in the reference.

In order to setup the lookup table using LWPR an input and an output set need to provided. The input is the start and the goal state, whereas the output consists of the time needed to traverse between these two states. Multiple start and goal state combinations have been used in order to obtain an eventual set of 10000 samples. For all of these start and goal state combinations the time needed has been calculated using TOPP. In order to improve the results of LWPR the input and output values have been scaled to the unit interval. Of this scaled set of 10000, 6000 samples have been used in order to create the model which is to be used as the lookup table. The remaining 4000 samples were used to verify the model. In order to verify the model the start and goal state where provided to the model and the time prediction was calculated. This time prediction was then compared to the actual calculated value by TOPP. This was done by calculating the mean squared error. The calculation method can be seen in Equation D-1. Here n_{tot} is the number of verification samples, n is the current sample. y_p is the needed time predicted by LWPR and y is the needed time calculated by TOPP. Lastly MSE is the mean squared error.

$$MSE = \frac{1}{n_{tot}} \sum_{n=0}^{n_{tot}} (y - y_p)^2 \tag{D-1}$$

Throughout the creation of the model it was compared to the test set multiple times. The result of these tests can be seen in Table D-1. Here it can be seen that a model with more

values becomes more accurate. The used model created with 6000 samples, which was tested on a dataset of 4000 samples has a MSE of $0.00234[s^2]$.

Model Samples	MSE $[s^2]$
10	1.24
100	0.0383
1000	0.0339
6000	0.0331

Table D-1: The MSE of an LWPR model created with varying sample sizes, compared to a verification set of 4000 samples.

Appendix E

PID-Controller Weights

PID-connect uses a PID-controller, the tuning of this controller is no trivial task. There are many different strategies which can be used to determine the necessary weights of the different elements of the controller. These methods will not be discussed in this thesis, however the final results of the tuning process might be interesting and are therefore given in Table E-1.

	P	I	D
Joint 1	150	0	25
Joint 2	1000	0	200
Joint 3	700	0	30
Joint 4	100	0	20
Joint 5	75	0	15
Joint 6	30	0	5

Table E-1: The weights used for the PID-controller in PID-connect.

The PID-controller is originally tuned for use in the RRT environment itself. This means that it does not constantly move towards the same goal, therefore the I-element is set to zero. This also means that the controller is strictly speaking a PD-controller. As was observed during the several test runs the inclusion of the I-element is not necessary for the correct functioning of the PID-controller for the UR5 robot arm.

Automated tuning of the PID controller has also been attempted, in order to minimize the needed input for the motion planning infrastructure. The automated tuning was performed using a numerical optimization method called Nelder-Mead [27]. However, this method could not be implemented properly as the convergence of the weights did not lead to the desired behavior of the system.