Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3630

An experimental analysis of optimization base motion planning

Alen Duraki

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. rer. nat. Marc Toussaint
Supervisor:	Ph.D. Nathan Ratliff
Commenced:	25. February 2014
Completed:	27. August 2014
CR-Classification:	1.2.8, 1.2.9

Abstract

Through the advance of technology, robots aren't just objects from science fiction novels, but they are becoming more and more part of our everyday life and are specially designed and constructed to support us. From little household helper that finish domestic chores to big industrial robots that assemble huge machines and automobiles, there are hundreds of different types of robots.
Especially in interaction with humans it is important for a robot to react to environmental changes. In case of upcoming difficulties the robots ought not to freeze in place and take a lot of time to calculate their possibilities on how to avoid obstacles that might be in their way. Rapidly-exploring random trees and other sampling-based methods are one way to solve this problem, where robots are able to find a viable solution and execute it. However, the resulting solutions were not efficient enough and post-processing consisting of optimization methods had to be applied to smooth out the trajectories. The necessity of this optimization lead to a new generation of planning methods that were based on optimization alone. In many domains, these optimization-based motion planners constitute the state-of-the-art.

This thesis compares various optimization-based motion planners within an efficient common environmental representation with respect to the following criteria: speed, accuracy, and applicability. Through a variety of tests with arbitrary obstacles the methods will be compared and the results presented. The document concludes with an outlook on further work and possibilities.

Kurzfassung

Durch den technischen Fortschritt sind Roboter nichtmehr nur Objekte aus *Science-Fiction* Büchern sondern Teile unseres Lebens. Sie werden zu einem immer größer werdenden Teil unseres alltäglichen Lebens und werden speziell dazu entwickelt, uns bei vielen Aufgaben zu unterstützen. Es gibt bereits eine Vielzahl an Robotern: von Haushaltshelfern, die kleine Arbeiten verrichten, bis hin zu größeren, industriellen Robotern, die gewaltige Maschinen und Automobile zusammenbauen.
Besonders in der Interaktion mit Menschen ist es für einen Roboter wichtig auf Veränderungen in seinem Umfeld zu reagieren, zum Beispiel wenn ein Hindernis aufkommt, soll der Roboter im Stande sein, in kürzester Zeit eine neue Route zu berechnen, und so die Aussetzzeit so minimal wie möglich zu halten. Rapidly-exploring random trees und andere Abtastmethoden sind nur ein Paar mögliche Methoden, bei denen Roboter eine Lösung finden, sie umsetzen und so möglich entstandene Probleme beheben. Die dabei entstandenen Methoden sind jedoch nicht immer effizient und erfordern eine Nachbesserung durch Optimierungsverfahren, um die berechneten Bewegungen zu verbessern. Die Notwendigkeit solcher Optimierungen führt zu

einer neuen Generation von Bewegungsplanern, die auf diese Methoden basieren. In vielen Bereichen werden sie mittlerweile als Standard angesehen.

In dieser Arbeit werden verschiedene Bewegungsplaner, die auf Optimierungsverfahren aufbauen, in einer vordefinierten Umgebung verglichen. Dabei wird der Wert auf folgende Faktoren gelegt: Geschwindigkeit, Genauigkeit und Anwendbarkeit. Durch eine Vielzahl von Tests werden die Bewegungsplaner verglichen und die Resultate präsentiert. Abschließend zeigen wir weitere Möglichkeiten auf, die in näherer Zukunft vielleicht durchführbar sein werden.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

In robotics we try to develop artificial beings with human-like capabilities, which should be involved in our everyday life. To reach this goal we have to solve many problems about the entire concept of an autonomous machine, e.g. what and how many sensors are needed so the robot can scan its environment sufficiently, or how to recognize obstacles and react to them as quickly as possible. The faster technology evolves, robots become increasingly capable, able to move around, grab things and interact with machines and humans in its environment. A reason for that is the higher accuracy of newer sensors and overall faster hardware used to build such robots.
Besides the evolution of sensors the motion planning methods have evolved as well. Even though the sampling-based path planning algorithms, e.g. *Rapidly-exploring random tree*, *Probabilistic roadmaps*, etc., are a working solution to motion planning in general, they are not always the most efficient way to find a feasible path for certain problems [KF11], [Ran12]. Due to this problems, so called optimization-based motion planning algorithms were developed to provide a better way on finding solutions for existing problems. But how can we find out what the best method to plan a robot's movement is and what the definition of a good motion planner ought to have? The difficulty and complexity varies from task to task and it is not guaranteed for a planner to work the same way for each of the tasks with the same efficiency. Using optimization-based methods can bring huge advantages, however, depending on the applied optimizer, complications might occur which would negate all the obtained benefits. Depending on what optimizer we use the needed time to calculate a feasible path can be either pretty small or enormously large. An Optimizer that requires lots of time is not appropriate for robots to operate at interactive speed. There are boundaries for the mathematical optimizations as well. It is not guaranteed that the resulting solution for an optimization problem is the global (best) possible, but a local optimum which solves the problem sufficiently enough. On the other hand this kind of motion planner produce a lot less overhead data which saves up storage management. There are lots of data reviewing sampling-based methods about efficiency, accuracy, and time but the newer optimization-based methods have not been reviewed to be able to make a reasonable statement about their capability as general motion planner for robots.

In this thesis we will experiment with various implementations of motion planner using certain optimization techniques. We use an existing modeling framework to define a virtual robot and

implement an efficient distance field based environmental representation. Through adjustments we want to find out if the resulting trajectories solve the problem with a valuable outcome. To determine if a solution is feasible we will take a look at convergence conditions and how close to the given goal position we can get. Through obstacles, e.g. shelves, tables and walls, we want to observe how the behaviour of the robot is affected by the different methods and the different environmental conditions. Time efficiency is another aspect that is important to evaluate effective motion planner, that is why we want to avoid huge calculation times that would force our robot to freeze for a long time.

1.2. Outline

The diploma thesis is outlined as follows:

chapter 2 – Equipment and methodology describes what techniques and methods were used to set up the environment. It is divided in multiple subtopics. In the first part we will talk about optimization and give a brief overview on the general problem and take a closer look on some methods. After that we will talk about the framework we implemented, tested, and visualized our experiments with. A short introduction of the system with further technical information is given. At last we will talk about the environment in which the tests took place; how we executed them and collected the data for a further analysing purpose.

chapter 3 – Results and Discussions will show the final results of the experiments with mathematical analysis, e.g. calculating the median. After that a discussion including comparisons of results and pointing out significant or unusual results of certain methods is following immediately.

chapter 4 – Conclusions completes this thesis and describes conclusions made through the results and what future perspectives can be made. We will also describe possible adaptations and changes that could lead to better algorithms and improve planning methods even more.

1.3. Conventions

Names of authors or developers are written in capitals, e.g. NATHAN RATLIFF.

For *vectors* we will use lower case bold letters, e.g. $\mathbf{x}$. If we want to talk about the i-th component of a vector $\mathbf{x}$ we write x_i.

A scalar function f of a vector $\mathbf{x}$ is called *objective function* and it is the function we want to maximize or minimize. Objective functions for the optimization problem are differentiable functions.

A function is m-times differentiable if for any $\mathbf{x} \in \mathbb{R}^n$ the derivative $\nabla^m f(\mathbf{x})$ can be calculated.

We differentiate between *configuration space* and *workspace*:

- The *workspace* is the Euclidean 3-space $\mathbb{R}^3$ with $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \in \mathbb{R}^3$.
- The *configuration space* describes a set of possible transformations that could be applied to the robot [LaV06].

A *trajectory* is a sequence of configurations describing a motion of a robot. Trajectories have the following properties:

- $\xi = \begin{pmatrix} q_1 \\ q_2 \\ ... \\ q_n \end{pmatrix}$ is a trajectory of length T, where $q_i, i \in 1, 2, ..., T$ is a multidimensional vector describing a position of the robot.
- The resolution of trajectories is given by the number n. The higher n is, the finer the trajectory becomes which leads to higher optimization times.
- A trajectory has two important points: q_{init} and q_{goal}. The former is the initial position of the robot's limb which is fixed and therefore it is not affected by the optimization methods, while the latter one is given as the final position and used to determine if a solution is found and if so, how precise it is.
- A feasible solution is found if the difference between $\phi(q_T)$ and q_{goal} is lower than a certain threshold Δ:

  $|q_{goal} - \phi(q_T)| < \Delta$

To optimize methods we assume that the functions we are working with are twice differentiable and therefore the gradient vector and Hessian matrix are given as followed:

- The gradient $\mathbf{g}$ is defined as $\nabla f(\mathbf{x}) = \mathbf{g}(f) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$.

- The Hessian matrix $\mathbf{H}$ is defined as $\nabla^2 f(\mathbf{x}) = \mathbf{H}(f)(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$

The identity matrix of dimension $n \times n$ is called $\mathbf{I}^n$ and defined as $\mathbf{I}^n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix}$

2. Equipment and methodology

2.1. Optimization Methods

Since the optimization methods are the central component about this type of planners, we want to discuss what *mathematical optimization* is and how it concerns us with regard to robotics. In the following subchapter we describe the general idea behind optimization and take a closer look on some methods that we implemented and used for our experiments. It is based on the books [NW06] and [JS96].

2.1.1. General problem

In optimization we are trying to minimize or maximize a function subject to equality and inequality constraints [NW06]. If f is our objective function and $\mathbf{x}$ is a vector of variables then we can write the *constrained optimization problem* as follows:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \quad s.t. \quad \begin{array}{l} c_i(\mathbf{x}) = 0, \quad i \in \mathcal{E}, \\ c_i(\mathbf{x}) \leq 0, \quad i \in \mathcal{I}. \end{array} \tag{2.1}$$

In (2.1) $\mathcal{E}$ and $\mathcal{I}$ stand for a set of indices of equality and inequality constrains respectively. If these two constrain sets are empty, we talk about the *unconstrained optimization problem*.

In optimization we differentiate between two kind of solutions: *local* and *global*. In Figure 2.1 we can see a visualization of global and local solutions for a given one-dimensional function $f(x)$.

To determine a *global solution* for this problem, we have to find a vector $\mathbf{x}^*$ that satisfies the equation:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^n. \tag{2.2}$$

If there is an $\mathbf{x}^*$ that solves (2.2), then we call $\mathbf{x}^*$ a *global solution* for the optimization problem.

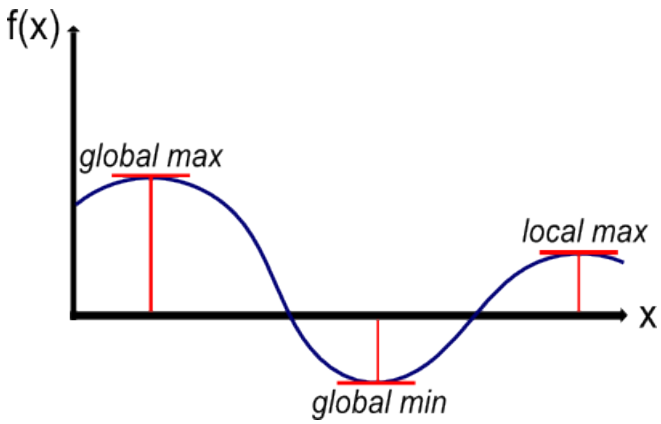

Figure 2.1.: Visual example of local and global optima of given function f

Now let $N \subset \mathbb{R}^n$ be a continuous set of vectors that are in the immediate neighbourhood of $\mathbf{x}^*$, then we call $\mathbf{x}^*$ a *local solution* if the condition

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \qquad \mathbf{x} \in N \tag{2.3}$$

is fulfilled. If one or more solution can be found for (2.3) then we talk about *local solutions* for the optimization problem.

However, in constrained optimization both the local and global solution have to satisfy the equality and inequality constrains from (2.1) to be considered as feasible solutions.

Now that we know what the general optimization problem is, let us talk about optimization methods. An *ideal optimization method* gets an arbitrary starting point $\mathbf{x} \in \mathbb{R}^n$ as input and converges to a local or global solution after a certain amount of iterations. If no solution can be found it has a termination condition which prevents the method from running indefinitely. The main task of those methods is to decide what direction to take and how big of a step size to take. Mathematically, it can be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{2.4}$$

where α_k is the *step size* and $\mathbf{p}_k$ is the *direction* at the point $\mathbf{x}_k$. Every method has its distinctive way of choosing a proper step size α_k and direction $\mathbf{p}_k$. To ensure that we have a decrease of f after every iteration, it is necessary for p_k to be a *descent direction*, which means that $\mathbf{p}_k^T \nabla f(\mathbf{x}_k) < 0$ has to be valid. We formulate the general descent direction with

$$\mathbf{p}_k = -\mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k), \tag{2.5}$$

where $\mathbf{B}_k$ is a symmetrical and nonsingular matrix. Combined with (2.5) we can give a general definition for descent direction through

$$\mathbf{p}_k^T \nabla f(\mathbf{x}_k) = -\nabla f(\mathbf{x}_k)^T \mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k) < 0. \tag{2.6}$$

The main difference between various optimization methods is the choice of $\mathbf{B}_k$.

The success and viability of an optimization method depends on its ability to find a suitable step size α_k, and a direction $\mathbf{p}_k$ that results in the biggest decrease of f after every iteration. Finding reasonable values for this two parameters might look easy at first, but it can take up a lot of time and is bound to high computational costs.

Nonetheless, there are some optimization methods that solve this problem sufficiently enough to be considered as possible methods for our motion planner. To decide if they are viable or not, we want to take a closer look and figure out what advantages and disadvantages they show.

2.1.2. Gradient Descent Method

The first method we want to take a closer look at is the *gradient descent method*, or sometimes called *steepest descent method*. This optimization method requires the objective function to be *once* differentiable because it is using the information of the derivative to make decisions about the direction we have to take in each iteration. It is one of the simplest methods and can be obtained from (2.5) by setting $\mathbf{B}_k$ to the identity matrix $\mathbf{I}^n$ which concludes to the following equation:

$$\mathbf{p}_k = -\mathbf{I}^n \nabla f(\mathbf{x}_k). \tag{2.7}$$

From (2.7) we can easily see that this method uses the gradient $-\nabla f(\mathbf{x}_k)$ as its direction for the next iteration. If we take the definition of the direction given in (2.7) and insert it in (2.4), we receive the iterative equation for the *gradient descent method* that can be formulated with

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k). \tag{2.8}$$

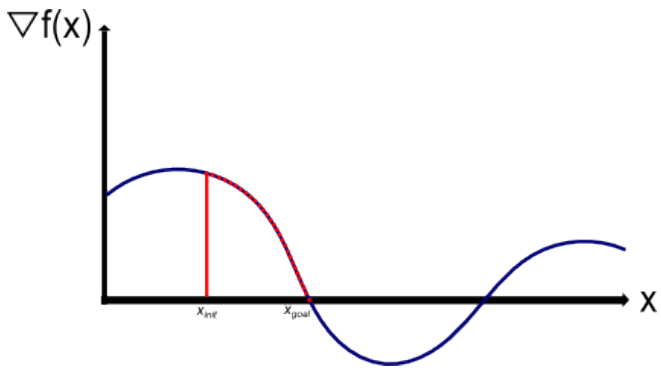

Figure 2.2.: Through the reduction of the gradient we reach the desired goal

Through Figure 2.2 we can see how the *gradient descent method* progresses towards a solution $\mathbf{x}_{goal}$ from an initial point $\mathbf{x}_{init}$.

Now that we defined the *gradient descent method*, let us talk about the advantages and disadvantages of it. Due to (2.8) we have a direction $\mathbf{p}_k$ that assures a decrease of the objective function f, which is quite possibly the only advantage we can get from this method. With this assurance we can say that if we can find a sufficient number of iterations k, we will eventually reach a local optimum at $\mathbf{x}_k = \mathbf{x}^*$. However, this leads us to the biggest disadvantage of this method, time. Even under ideal conditions the convergence rate is impractical and takes up a lot of time. To find a reasonable solution we need enormous numbers of iterations and lots of calculations to converge to a local optimum. Another big disadvantage is that it is only using information about the gradient, therefore it can not distinguish between an optima of a function and a saddle point. These are mainly the reasons why this method is not a viable solution for real time applications such as motion planner in robotics. But since it is one of the simplest methods, we still want to make use of it and set it as basis for other motion planner. Moreover we want to try out various adjustments to make this method applicable in terms of robotics.

2.1.3. Newton's Method

Unlike in the previous method, the objective function has to be *twice* differentiable for the *Newton's method* to be applied. This method uses information about the derivatives to determine what direction to take at every iteration step. To get *Newton's method* from the equation (2.5) we simply set $\mathbf{B}_k$ to the Hessian matrix $\mathbf{H}(f)(\mathbf{x}_k)$. Because the objective function f is clear from the context, we use the following short notation for the Hessian matrix at point $\mathbf{x}_k$: $\mathbf{H}(f)(\mathbf{x}_k) = \mathbf{H}(\mathbf{x}_k) = \mathbf{H}_k$. We can add the newly defined $\mathbf{B}_k$ to (2.5) and we get

$$\mathbf{p}_k = -\mathbf{H}_k^{-1}\nabla f(\mathbf{x}_k) = -\nabla^2 f(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k) \tag{2.9}$$

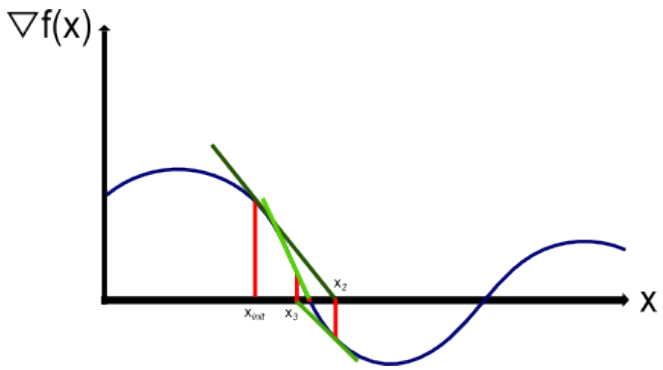

Figure 2.3.: Visualization of iterative search for optima with *Newton's method*.

as the direction for *Newton's method*. With this direction for the iteration, we combine (2.4) with (2.9) and we have the iteration rule mathematically written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k). \tag{2.10}$$

Figure 2.3 shows the iterative progression of *Newton's method* from the initial point $\mathbf{x}_{init}$.

Since it is not guaranteed that the Hessian matrix $\mathbf{H}(f)$ is positive definite, direction $\mathbf{p}_k$ is not always a descent direction. That means, unlike the *gradient descent method*, we can not assure that we have a decrease of the objective function f after every iteration. Therefore, we adjusted this method by adding $\lambda \mathbf{I}^n$ to the Hessian matrix $\mathbf{H}$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k (\mathbf{H} + \lambda \mathbf{I}^n)(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k) \tag{2.11}$$

to make sure that it is positive definite. This trick is called a *Levenberg Marquardt update*.

With this method we get better convergence than *gradient descent's*, and therefore finding a solution for the optimization problem is a lot faster. *Gradient descent* had problems with saddle points and got stuck at them, because it could not distinct between optima and saddle points, however, *Newton's method* can distinguish between the two, but it is not able to necessarily avoid them in any nice way.
Due to its fast convergence and low calculation cost, *Newton's method* is a good fit as proceeding for a motion planner used in robotics. Like with the *gradient descent*, we want to achieve improvements by adjusting certain parameters on this method.

2.1.4. Optimization in robotics

The biggest question for now is: *How to apply optimization to robotics?*

In robotics we want to generate the *shortest trajectory* ξ from a given *initial configuration* q_{init} to a *goal configuration* q_{goal} that is *collision-free* and *smooth*. This sentence can be formulated mathematical as an optimization problem as:

$$\min_{\mathbf{q}\in\mathbb{R}^n} f_q(\mathbf{q}) \quad s.t. \quad \begin{array}{ll} c_i(\mathbf{x}) = 0, & i \in \mathcal{E}, \\ c_i(\mathbf{x}) \leq 0, & i \in \mathcal{I}, \end{array} \tag{2.12}$$

where f_ξ is the trajectory objective function, $\mathbf{q} \in \mathbb{R}^n$ is a n-dimensional vector containing all configurations of a trajectory, and c_i for $i = 1, 2, \cdots, n$ are the constraints given through the environment of the robot. Examples for constraints are obstacles that are on the robot's way to the goal configuration, or any constraint given through the physical appearance of the robot, e.g. height, mobility, degree of freedom. Like in the general optimization problem, we want to have an iterative function that takes an arbitrary trajectory $\xi_{start} \in \mathbb{R}^n$ and returns a trajectory $\xi^* \in \mathbb{R}^n$ that defines a smooth and collision-free motion for the robot.

Being able to construct a motion planner that immediately designs a reasonable solution for the prior mentioned problem would be ideal, since it would not only reduce planning and calculation time but it also would be a huge step into the right direction for the future of robotics. We would be able to develop and build robot systems capable of interacting in real time with their environment and especially with humans. Artificial intelligence might not be only part of science fiction novels but a possible future for the state of our current robotics.

2.1.5. Examples for optimization-based motion planner

We want to give a brief introduction to an optimization-based motion planner: *Covariant Hamiltonian Optimization for Motion Planning* (CHOMP) [NRS09]. Unlike prior motion planner, CHOMP was designed to directly generate trajectories that optimize a given problem in respect to various criteria. As defined in 2.1.4, CHOMP uses information about derivatives to generate a smooth and collision-free trajectory between a start configuration and a desired goal configuration. It calculates derivatives via finite-differencing and uses them to solve the problem. By defining obstacle cost function and using them to penetrate the resulting solutions, CHOMP has a strict decision making process for finding the optimal solution. It uses information about velocity and acceleration to distinctively decide when the robot is getting too close to an obstacle. Through a distance field it can determine when certain constrains are being violated and how the penalization has to be taken care of.

Another approach to motion planning was taken through the following planning method: *Stochastic trajectory optimization for motion planning* (STOMP) [KCETS11]. Even though the name is some kind of similar to CHOMP, they differ in important ways. STOMP takes an initial trajectory that might be *noisy* and uses iterative optimization to generate a smooth and collision-free trajectory. With every iteration, STOMP takes the given trajectory and explores the space around it to create a new trajectory and combines them to find a new updated trajectory with overall lower cost function. Just like in CHOMP, the main optimization is done to cost functions considering the obstacle and smoothness criteria. However, STOMP strictly uses stochastic gradient approximations and does not require explicit cost derivative calculations.
This are just two motion planner that use optimization as part of their planning process. Both of them have their advantages and disadvantages respectively, and therefore either of them can be applied to different problems and be better or worse than the other.

2.2. ROS - Robot Operating System

The main framework we used in this thesis to implement the packages for our experiments is called *Robot Operating System* [Ros]. It is an open source system for personal robots and provides all the needed services for developing. Various tools are included to ease up the working process and speed up the development of robots. Other advantages of ROS are that there is a big community behind it with a large number of free libraries that were specifically developed for ROS and can be easily integrated. This operating system consists of different distributions and for this thesis we used the *ROS groovy* distribution with *Ubuntu 12.04 LTS*.

2.2.1. Optimization framework

Over the past six moths, NATHAN RATLIFF has developed an optimization framework for Autonomous Motion Department of the Max Planck Institute for Intelligent Systems (short: AMD framework) that was conceptualized to implement and test optimization-based motion planner. It consists of packages that are not ROS dependent, e.g. `amd_motion`, `amd_optimization`, etc., and packages that use ROS functionality.

Within this framework the user has access to already existing implementations of optimization methods or can make use of the interfaces to implement more methods. After implementing his needed functionality, he can use various ROS tools to run, test, and debug his implementations. Due to the handy use of the AMD framework, we used it to implement our testing environment and execute the experiments under several different conditions. The optimization framework includes common constrained and unconstrained method implementations such as *gradient descent*, *Newton's method with Levenberg Marquardt updates*, and *Augmented Lagrangian*. To set up a motion planner, all we have to do is define a trajectory objective function f consisting of

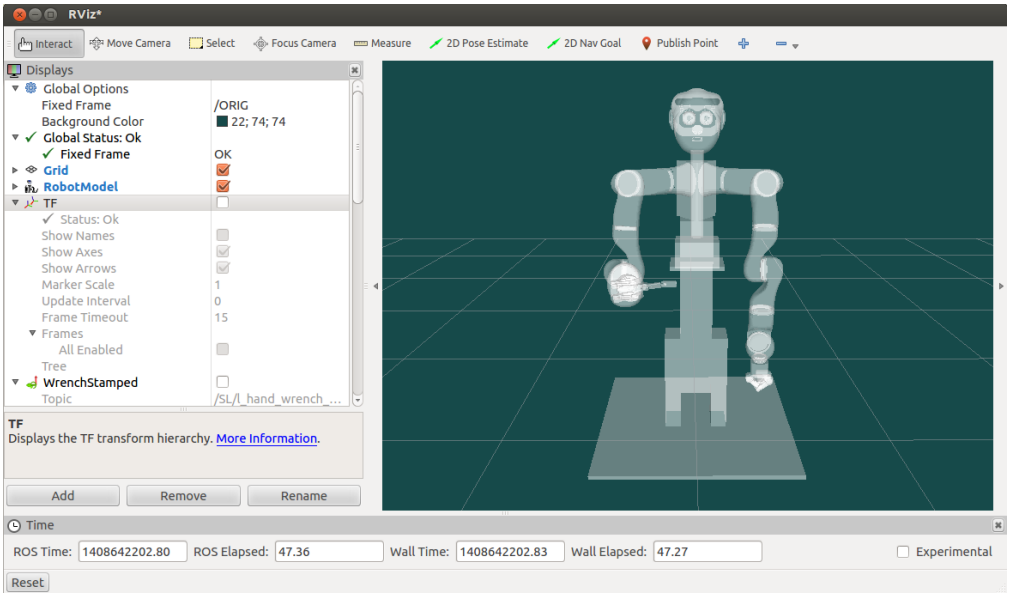

Figure 2.4.: The rviz ROS tool used to visualize our robot model

differentiable maps used to transform from *workspace* to *configuration space* and vice versa, set the default configuration and a goal that has to be reached, and various parameters that handle the penalization process while optimizing. Because AMD consists of ROS packages as well as default C++ libraries, the user has the benefits of using all the existing ROS tools to integrate his finished planner into an existing robotic system or test it with animated robots to find out more about possible problems and bugs.

While working with this framework, we used an animated robot model called *Apollo robot* that we have visualized with the ROS tool *ROS visualization (rviz)*. We fed our planner initial data and predefined parameters and it calculated an optimal trajectory solution for the existing problem. Through messaging, we fed Apollo the trajectory data and it animated the resulting motion within a given timeframe. Moreover, the visual feedback was used to compare the different solutions of various planner settings, that we obtained through parameter optimization.

2.3. Experiments

For the experiments to be convincing, we had to design a general environment and use it for all methods under the same conditions. Therefore the following criterion has to be met for the trajectory objective function:

- Robot's definition has to be given (i.e. *kinematics map*, *key points map*, etc.),

- metric term to determine distances within environment,
- various other intermediate terms,
- and the parameter for these terms functions.

Due to the existing model of *Apollo*, the definition of the robot was already given. Within this model we have a *kinematics map* and *key points map* that were used to perform the transformation from a given point $\mathbf{q}$ in configuration space to a cartesian position $\mathbf{x}$ in workspace:

$$\phi(\mathbf{q}) = \mathbf{x}. \tag{2.13}$$

Another important point are the intermediate terms, that are used to penalize motions that are hurting our definition of viable solutions. Our methods used the following five intermediate terms to calculate smooth and collision-free trajectories:

1. `DerivativeNormTerm`: Basic dynamics term in configuration space
2. `PosVelRegularizer`: Dynamics term in workspace
3. `QuadraticJointPotential`: Potential penalizing proximity to joint limits
4. `DistanceFieldFunction`: Obstacle cost term
5. `SquaredPotentialInX`: Biases movement to some default configuration to resolve redundancy

Within his AMD framework, NATHEN RATLIFF has already implemented most of these needed terms. However, we had to provide our own cost term to calculate distances to existing obstacle in our surrounding. For this purpose we wanted to use *Euclidean-distance* [DD09] and implemented a *distance field* (see: subsection 2.3.1 – Distance Field). All this cost terms implemented the `NDimDiffFunction` interface and provided methods to calculate the *objective function value*, *gradient value*, and the *Hessian value* at a given point $\mathbf{x}$. We wanted to try out different variants of this terms, so we created a wrapper `IntermediateCostTermWrapper` that omits the calculated Hessian matrix. That will show us whether such changes can improve the resulting trajectories or not.
There are a couple of bash scripts created mainly for the purpose to run our experiments fast with different configurations. These scripts can provide our planners the parameters without making changes to the code itself. If no parameters are set within a script, a collection of default parameters are used instead.

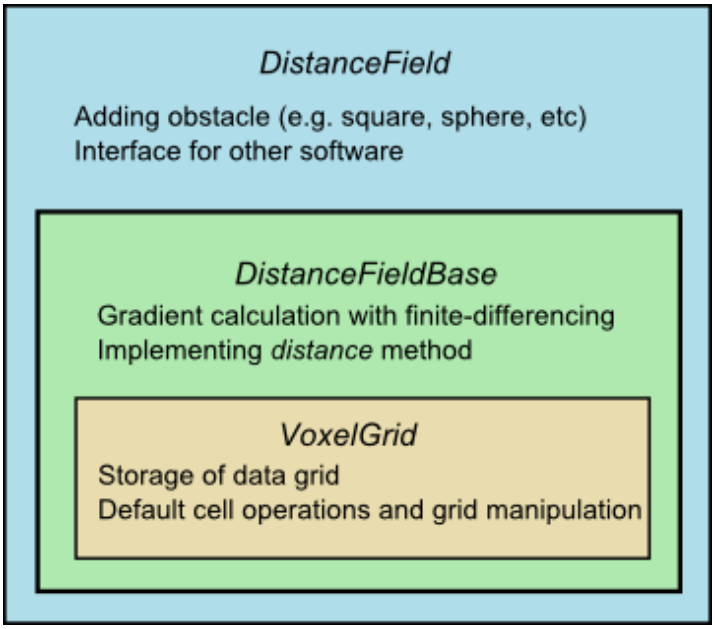

Figure 2.5.: Main structure of the distance field package

2.3.1. Distance Field

Providing an accurate and efficient distance transform or distance field is mandatory for a motion planner. A lot of time might be lost calculating proximity to the nearest obstacle. MRINAL KALAKRISHNAN developed based on the paper from FELZENSZWALB & HUTTENLOCHER [FH04] a distance field using the described *distance transform* algorithm.
Even though MRINAL KALAKRISHNAN's work was a good basis for the distance field we needed, some adaption had to be made. The main problem was that his distance field [Kal] had no functionality to add obstacles to it. The motion planner had to be tested under real conditions with obstacles in the robot's environment. In regarding to this requirement we had to generate an API for the user. It should be possible for him to add various obstacles (e.g. rectangles, spheres, etc.) to the distance field. MRINAL KALAKRISHNAN's distance field was *unsigned*. For our planner to work properly, however, a *signed distance field* was needed. To reach this goal we had to change up the base class and add functionality to assure correctness of the field.

Once the main changes have been applied to the distance field code, it could be added to the AMD framework and it provided the following API for the user:

- `getSignedDistance`: Gets signed distance value from a given position $\mathbf{x} \in \mathbb{R}^3$
- `getSignedDistanceGradient`: Gets signed distance gradient value from a given position $\mathbf{x} \in \mathbb{R}^3$

- `addSquareObstacleToField`: Adds a squared obstacle to the field and updates the distance field
- `addSphereObstacleToField`: Adds a sphere obstacle to the field and updates the distance field
- `addCylindricObstacleToField`: Adds a cylindric obstacle to the field and updates the distance field
- `addShelfToField`: Adds a shelf obstacle to the field and updates the distance field
- `addTableToField`: Adds a table obstacle to the field and updates the distance field

The `getSignedDistanceGradient` uses $\mathbf{x} \in \mathbb{R}^3$ to calculate the corresponding cell within the grid and uses *finite differencing* to get the gradient with

$$\nabla f \begin{pmatrix} x \\ y \\ z \end{pmatrix} = (f \begin{pmatrix} x+1 \\ y+1 \\ z+1 \end{pmatrix} - f \begin{pmatrix} x-1 \\ y-1 \\ z-1 \end{pmatrix}) \cdot \texttt{inv\_twice\_resolution} \tag{2.14}$$

where $x, y, z \in \mathbb{N}$ are the coordinates of the cells within the grid, and `inv_twice_resolution` is a constant to resolve errors concurring through the resolution.
In Figure 2.6 we can see a table obstacle that was added with the `addTableToField` function. The red squares symbolize the distance value which means. The further away we are from the table, the lower the opacity of the squares is, and therefore the robot would be penalized less at these points. The blue squares represent the inside of the obstacle with a *negative* distance field value.

This distance field was implemented as `NDimDiffFunction` and given to the motion planner. Unlike other functions, it can only generate the gradient vector $\mathbf{g}$ and returns a *zero matrix* as the Hessian $\mathbf{H}$.

2.3.2. Set-up of experiments

After all these preparations have been made, we were ready to create our motion planner for the desired experiments. We created different scenarios in which the robot has to determine the best possible trajectory or motion. The scenarios are divided as follows:

- **Scenario 1**: Robot stands in front of a wall and has to touch his nose. The wall is $1.0m$ in front of the robot, and its measurements are $2.0m \times 1.0m \times 2.0m$.

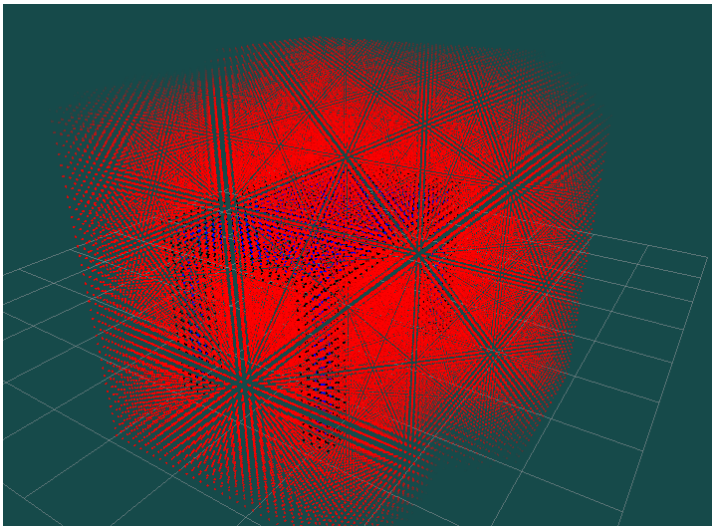

Figure 2.6.: Table obstacle within distance field visualized with rviz.

- **Scenario 2**: Robot stands in front of a table with its hand under the table. It has to move the hand to a position above the table. The table is $0.5m$ in front of the robot, and its measurements are $1.0m \times 1.0m \times 1.0m$.
- **Scenario 1**: Robot stands in front of a shelf with its hand in the bottom part. It has to move his hand to a position in the top part of the shelf. The shelf is $0.5m$ in front of the robot, and its measurements are $1.0m \times 1.0m \times 2.0m$. It has a two symmetrical shelfs, a bottom and a top shelf.

Our main goals is to compare various implementations of the *Gradient descent method* and *Newton's method*. We distinguish between *method adaptations* and *objective function adaptations*.

As *method adaptations* we understand:

- Initial step size: Using a variety of starting step sizes
- Step size update: Adjust update rule for step sizes
- Iterations: Maximal number of iterations a method has to find a feasible solution
- Waypoints adjustment: Changing the amount of trajectory waypoints

For the *objective function adaptations* we generated three test cases that use different intermediate terms to determine an ideal trajectory. In section 2.3 we presented a list of various terms and talked about a wrapper for them. In our tests we systematically wrapped some of this terms and omitted the Hessian matrices. Therefore, following *trajectory objective functions* were used for the optimization experiments:

- Default *Newton's method*: We maintain all five cost terms
- Full dynamic Hessian approximation: We keep only the Hessian of the first and second term.
- Configuration space dynamic Hessian approximation: We remove the Hessian of all terms except the first.

To get a convincing improvement for the optimization-base motion planner is our main goal here. Through the several test cases and set ups we want to find out what might be the best solution and which set-ups lead to worse solutions.

The motion planner had to work under the same conditions, and therefore we prepared a virtual machine. This computer had Ubuntu 12.04 LTS installed and we had 8 GB RAM for our calculations.

Set-up for *Gradient descent method*

- **Initial step size**: Started at $ss = 0.000125$ and went up to $ss = 0.000175$ in 0.000025 steps, which lead to three different variants for the initial step size.
- **Step size update**: Due to existing problems with adjustable step sizes, we used only fixed step sizes for *Gradient descent method*.
- **Iterations**: Set to fixed $i = 30000$ because of the bad convergence rate of this method.
- **Waypoints adjustment**: $n = 30$, $n = 100$, and $n = 200$ were chosen for the amount of waypoints.

After running some initial tests with the *gradient descent method* it was easy to see that this was no reasonable and feasible motion planner to be used in robotics. Therefore, this motion planner was used only on **Scenario 3** as a viable comparison to the results of *Newton's method*. Only three tests were executed with this method with the different amount of waypoints.

Set-up for *Newton's method*

- **Initial step size**: Started at $ss = 0.05$ and went up to $ss = 0.95$ in 0.05 steps, which lead to 19 different variants for the initial step size.
- **Step size update**: Unlike *Gradient descent method* the *Newton's method* had a problem with fixed step sizes which forced us to use only adjustable step sizes for this method.
- **Iterations**: Set to fixed $i = 100$. Within this range it was quite possible for the *Newton's method* to find a feasible solution.

- **Waypoints adjustment**: $n = 30$, $n = 100$, and $n = 200$ were chosen for the amount of waypoints.

For the *Newton's method* we were able to run all the planned versions with the various combinations. So with this planner we executed nine tests resulting from the combination of the **scenarios** with the differing amount of waypoints n.

3. Results and Discussions

In this chapter we want to present the results from the test we ran with our motion planner and discuss the meaning of them. To give a reasonable representation of the the data we want to use some mathematical definitions to describe the quality of a given test. We are using the *median*, *average* (3.1), and *standard deviation* (3.2) to display the quality of the resulting data.

Let N be the number of existing test results, then the *average* is calculated with

$$\frac{1}{N}\sum_{i=1}^{N}(x_i), \tag{3.1}$$

where x_i is the i-th result. Under the same assumption we get the formula for the *standard deviation*:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - x_{avg})}{(N-1)}}. \tag{3.2}$$

3.1. Experimental results

For the presentation of the results we are referencing to the tables in the Appendix A. We are taking the median, average, and standard deviation from the tables and take a closer look on certain values. To determine if a feasible solution was found, we are setting $\Delta = 0.01$ in the equation of 1.3. This means, that the final configuration $\mathbf{q_T}$ of the generated trajectory ξ has to be lower than Δ for us to have a reasonable solution. If the final configuration $\mathbf{q_T}$ violates this condition then ξ is not solving the given problem sufficiently and it is not accepted.

3.2. Examination

We are going through each **scenario** from 2.3.2 and present median (*M*), average (*A*), and standard deviation (*S*). Additionally, we will examine data that is standing out in either a

positive, or in a negative way. Moreover, as mentioned in the same chapter, we are applying the various adjustments on the intermediate cost terms for each of the *scenarios*.
As mentioned in 2.3.2, the **scenarios 1** and **2** were omitted for the gradient descent because of the lacking performance of the *gradient descent method*. But to have some comparable data, we executed an experimental optimization on **Scenario 3** using the *gradient descent method*.

Scenario 1: Wall experiment – 30 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	0,476315789	0,010575479	0,476315789	0,010383555	0,472105263	0,023514458
M	0,48	0,0100648	0,48	0,0100646	0,47	0,0100662
S	0,009551339	0,002023507	0,008950808	0,001425991	0,007873265	0,039934892

Table 3.1.: First scenario with 30 waypoints

Through this table we have all the needed information to inspect this test case. For each of the three versions of the motion planner, the required time in average is around $0.47s$ which is quite reasonable for our cause. The average distance for the first two versions is close to our convergence value $\Delta = 0.01$ but does not fall below it, which means we have not found a feasible solution. The third version gives us a median close to $\Delta = 0.01$ but it does not fall below it either. Additionally its average distance is way over $0.023m$ which means we are approximately $2.3cm$ from our goal point away. With a standard derivation around $3.9cm$ it is the worst planner with this set-up.
If we take a look at the entire table A.1, we can see why the third version is giving us those bad results. If we use an initial step size that is close to $ss = 1.0$, the final robot position is $\approx 13.7cm$ away from the goal position which is an undesirable result. However, it is fascinating to know that if we use $ss = 0.1$ and $ss = 0.85$ as initial step sizes, the third version of the motion planner generates a solution where the robot is less than $1.0cm$ away from the desired position.

Scenario 1: Wall experiment – 100 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	1,553157895	0,007941839	1,558421053	0,007807503	1,560526316	0,030632445
M	1,54	0,00784141	1,54	0,0076171	1,54	0,00806611
S	0,025397426	0,001007865	0,037603658	0,000756215	0,033578049	0,044706032

Table 3.2.: First scenario with 100 waypoints

At the cost of some calculation time we reached already improvement for the optimizations. The first two versions of the motion planner reach an average distance from $\approx 0.7cm$ to our destined goal. The standard deviations are with values around and below $0.1cm$ ideal for our purposes. We want to point out that the second version of the motion planner solves the problem in average faster with higher precision.
Like in the first test with 30 waypoints, the third version is giving unsatisfying results. The average and standard deviation are over $3.0cm$ even though the median is at $0.8cm$. By taking a look at the whole table A.2 we see that step size $ss > 0.75$ leads to results that are more than $\approx 10.5cm$ away from our main goal. If we keep this in mind and use only step sizes $ss \leq 0.75$ then the third version is giving us convincing results that satisfy our conditions. With the summarized data we can make some assumption about the collected data but we need to inspect it closer to decide if a motion planner is viable or not.

Scenario 1: Wall experiment – 200 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	3,24	0,008157897	3,221052632	0,007915012	3,231052632	0,032473457
M	3,23	0,00808522	3,22	0,00780167	3,21	0,00834295
S	0,035276684	0,001042863	0,040674144	0,000904518	0,051628452	0,041480754

Table 3.3.: First scenario with 200 waypoints

For all these results the required time is with more than $3.22s$ too high. The accuracy got worse compared to the case with 100 waypoints. Investing twice the time for the calculations and losing accuracy is a bad trade-off for this case. Remarkable is the fact, that for all the motion planner applied on the various test cases, the initial step size $ss = 0.05$ leads always to a non-optimal solution.
Through table A.3, we see clearly that the third version is still the least reliable. If we are

using an optimization with more waypoints, a smaller step size ss is mandatory to solve the problem with a feasible solution. The motion planner can not generate a usable trajectory with $ss \geq 0.75$.

Scenario 2: Table experiment – 30 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	0,491578947	0,046372944	0,471052632	0,061000942	0,475789474	0,010800905
M	0,48	0,00601621	0,47	0,00577956	0,48	0,00469523
S	0,040724434	0,06291398	0,008093026	0,075467118	0,033385923	0,026843648

Table 3.4.: Second scenario with 30 waypoints

Change of environment has no effect on the speed of the motion planner. With $\approx 0.48s$ it has generated a feasible trajectory. However, if we take a look at the standard deviation for the distance, we can see that we had a fall off there. The values for the first version are fluctuate with $\approx 6.3cm$ around the average, but its median is with $0.6cm$ the best result we got so far. This means that there are certain solutions that are standing out negatively. Our second version of the motion planner is generating even worse results. Surprisingly this time the third versions was the most successful. With a median of $\approx 0.5cm$, and therefore it is closest to our desired position.
By inspecting the entire table, we see that our motion planners have certain step size values generating almost perfect trajectories. With a step size $ss = 0.6$, the second version is only $0.2cm$ off, while using $ss = 0.55$ leads to a solution that is $\approx 21.3cm$ off from the required position. The third version got an ideal solution for every step size except for $ss = 0.8$, and therefore it is the most viable one for this scenario.

Scenario 2: Table experiment – 100 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	1,561578947	0,006572154	1,552105263	0,004043997	1,557368421	0,006401402
M	1,56	0,00370381	1,55	0,00370833	1,56	0,00402259
S	0,02167004	0,011636142	0,019882697	0,000873756	0,026841532	0,009800045

Table 3.5.: Second scenario with 100 waypoints

At $ss = 0.95$ the first motion planner has a $5.4cm$ offset to the goal and that is the only solution that has not converged for this test case. All other step sizes lead to feasible solutions for all three planner. With calculation times around $1.55s$ it is debatable if it is reasonable to use so many waypoints.

Scenario 2: Table experiment – 200 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	3,225789474	0,009713417	3,260526316	0,008047844	3,227894737	0,005909818
M	3,22	0,00566804	3,25	0,00558847	3,23	0,00574177
S	0,036562851	0,018906997	0,05264912	0,01131687	0,028003759	0,000377053

Table 3.6.: Second scenario with 200 waypoints

Like in the previous scenario, all motion planner solved the problem and generated trajectories reaching the goal position. The average of the first two planners is just below $1.0cm$ and therefore they converged properly. Using 200 waypoints for a trajectory leads always to a calculation time over $3.0s$. Freezing time of $3s$ is significantly high for robotics' purposes. Our robot would not be fully interactive if he stops for several seconds to determine his next motion or movement.
Our first version of the motion planner has a value that is falling off the grid for $ss = 0.7$ and the second planner has one at $ss = 0.75$. Aside from that we have reached ideal convergence for every step size.

Scenario 3: Shelf experiment – 30 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	0,413157895	0,010218312	0,430526316	0,006787773	0,464736842	0,006677118
M	0,47	0,00662503	0,47	0,00662263	0,47	0,00662083
S	0,08698995	0,014660317	0,068513882	0,000439549	0,026534761	0,000277742

Table 3.7.: Third scenario with 30 waypoints

As we can see, the difference between table and shelf is not recognizable. The standard deviation for the first method is around $1.0cm$ due to a spiking value at $ss = 0.9$. The average for this method is above our threshold value $\Delta = 0.01$, but has no greater meaning when you

take a look at A.7.
The two other versions have standard deviations that lie at $\approx 0.03 - 0.04cm$, which means that all results are close to each other. In average the distance to our goal position is $0.6cm$ and therefore we have convergent solutions. To optimize a trajectory with 30 waypoints our motion planner require less than $1s$, which would be more than ideal for a viable solution in robotics.

Scenario 3: Shelf experiment – 100 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	1,565789474	0,017076666	1,551578947	0,028872566	1,542631579	0,011382859
M	1,55	0,00628406	1,55	0,00604228	1,53	0,00695569
S	0,045131387	0,046453594	0,02733804	0,067590153	0,029597692	0,019445054

Table 3.8.: Third scenario with 100 waypoints

As we have seen in the previous results, the averages and standard deviations of all the motion planners are not convincing. To determine if a planner is viable or not, we have to take a closer look at all the results generated during an experiment. You can see in A.8 that solutions are not converging and getting stuck at some points that are more than $20.0cm$ away from our goal position. If the step sizes leading to this bad results were omitted, our motion planners would be ideal for robotics' purposes. Without the spiking values, planners generate trajectories with final configuration that are closer to the desired position.

Scenario 3: Shelf experiment – 200 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	3,304210526	0,045546636	3,332631579	0,041123423	3,323684211	0,029157883
M	3,3	0,00877765	3,32	0,00861207	3,31	0,00860566
S	0,018653504	0,044659174	0,031418697	0,042495214	0,031659741	0,089719925

Table 3.9.: Third scenario with 200 waypoints

Using 200 waypoints for this scenario leads to the worst outcome. The averages for all three planners are higher than $2.9cm$ while the standard deviations are between $4.2cm$ and $8.9cm$. The first version of the motion planner produces for step sizes $ss \leq 0.45$ feasible solutions.

Aside from that, all the solutions are off by at least $\approx 4.3cm$. For the second one the step size $ss = 0.5$ is the upper threshold. Step sizes above 0.5 are not sufficient enough for us to use. By far the best results are provided by the third version of our planner. Until reaching a step size of 0.8 it is solving the optimization problem adequately. Its biggest fall-off is with $ss = 0.95$ where the calculated solution is $\approx 40.0cm$ off the main goal position. This value is not bearable.

Now that we covered the nine test case for the *Newton's method* let us take a look at the solutions generated through *gradient descent method*.

Scenario 3: Shelf experiment (gradient descent) – 30 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	124,3666667	0,0119849	124,7033333	0,0119849	124,8266667	0,0119849
M	124,38	0,0119849	124,58	0,0119849	124,94	0,0119849
S	0,1106044	0	0,26727015	0	0,287286152	0

Table 3.10.: Third scenario with 30 waypoints (gradient descent)

The first thing that we see is the time required for this motion planner. With over $120s$ they needed $250\times$ more calculation time than the planners using *Newton's method*. And even with 30.000 iterations, they have not converged to a feasible solution. However, the most remarkable part of these planners are the similar results. No matter what step size you choose, all the motion planner return a solution that is $\approx 1.2cm$ off from the goal position. Because gradient descent ignores information about the *Hessians*, the three versions are *identical* for this optimization method.

Scenario 3: Shelf experiment (gradient descent) – 100 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	421,0633333	0,0121689	421,39	0,0121689	422,7766667	0,0121689
M	421,39	0,0121689	421,4	0,0121689	422,74	0,0121689
S	1,496974727	0	1,065035211	0	0,875576001	0

Table 3.11.: Third scenario with 100 waypoints (gradient descent)

With 100 waypoints it needs $280\times$ more time than the *Newton's method* with the same number of waypoints. By increasing the number of waypoints we got a loss in accuracy. Aside from that, everything is the same as with 30 waypoints.

Scenario 3: Shelf experiment (gradient descent) – 200 Waypoints

	First version		Second version		Third Version	
	Time (s)	Distance (m)	Time (s)	Distance (m)	Time (s)	Distance (m)
A	856,74	0,0121663	858,3833333	0,0121663	859,5866667	0,0121663
M	855,68	0,0121663	857,92	0,0121663	859,24	0,0121663
S	2,691393691	0	0,811192538	0	1,314736983	0

Table 3.12.: Third scenario with 200 waypoints (gradient descent)

Like before, increasing the number of waypoints did not improve the outcome of the solutions. With $\approx 855s$ for the calculations, this method is not sustainable for our purposes. A robot can not freeze in place for more than $2s$ to calculate its motions.

4. Conclusions

Through the collected data we can say that our motion planner were mostly successful. Adapting the step sizes iteratively has revealed some unbearable occurrences leading to non-converging optimization methods. The three different versions of our motion planner displayed exactly what was expected: various tasks require a diversity of motion planners. While in the first scenario the results of the third motion planner were questionable, its solutions for other scenarios were astounding. The same goes for the other planners, which have their pros and cons themselves.

In regard to robotics, we can say that motion planner based on *gradient descent* are not a viable option at all. The time such a motion planner takes up to get a solutions is unbearable and undesirable. Even with the lowest number of waypoints these planners needed at least $120s$ to determine the *optimal* solution. Additionally, the given 30.000 iterations were not enough to make the solution converge.
However, the *Newton's method* based motion planners were a big success. Even for trajectories consisting of 200 waypoints, these planners did not need more than $3.5s$ in average. Aside from spiking divergent values, they were able to determine converging solutions to all of the three scenarios. The most important part about them though, is to pick the right initial step size. Our tests have shown that the best step sizes for one task might be the worst for another task.

4.1. Future perspective

Due to time issues, we were only able to run our tests with unconstrained motion planners. However, in robotics we have to take account of other constraints than just obstacles. The physical appearance of the robot forces joint constraints on us, that were neglected by our motion planners. To consider them properly, we have to implement a motion planner using a *constraint optimization method* (e.g. *Augmented Lagrangian method*). Such a motion planner would not violate joint constraints, and therefore could be applied to real robots.
Because our technology is advancing with enormous speed, this research field will get more attention in the near future. It is relatively new and has big potential to grow even more. Building robots that are able to interact with humans efficiently would ease our everyday lives a lot. It might even be, that someday we will go to a restaurant and a robot is bringing us our

meals using *motion-based planners* to avoid other customers' tables and chairs. Until then we have to invest more time on researching such methods to find more powerful planners.

A. Appendix

A.0.1. Test data

To assure completion of collected test data, we attach the collected data that was created during the tests with the motion planner.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	0,47	0,0189112	0,48	0,016271	0,46	0,011386
0,1	0,48	0,0106575	0,48	0,00994725	0,47	0,00959152
0,15	0,47	0,0102731	0,46	0,0100414	0,48	0,0100488
0,2	0,49	0,0101262	0,48	0,0100462	0,46	0,0100732
0,25	0,49	0,0100749	0,48	0,0100555	0,47	0,0100672
0,3	0,48	0,0100635	0,46	0,0100618	0,48	0,0100662
0,35	0,47	0,0100616	0,48	0,0100636	0,46	0,0100673
0,4	0,48	0,0100628	0,47	0,0100681	0,48	0,011349
0,45	0,48	0,0100626	0,48	0,0100614	0,47	0,0100649
0,5	0,49	0,010058	0,48	0,0100428	0,47	0,0100623
0,55	0,49	0,0100682	0,47	0,0100641	0,47	0,0100174
0,6	0,47	0,0100681	0,48	0,01007	0,48	0,0100691
0,65	0,47	0,0100667	0,48	0,0100646	0,47	0,0100657
0,7	0,48	0,0100691	0,47	0,0100692	0,48	0,0100641
0,75	0,46	0,0100596	0,48	0,0100733	0,47	0,0100632
0,8	0,47	0,0100639	0,47	0,010076	0,48	0,0100739
0,85	0,47	0,0100648	0,47	0,0100678	0,48	0,00998188
0,9	0,46	0,0100641	0,48	0,0100726	0,46	0,137022
0,95	0,48	0,0100582	0,5	0,0100709	0,48	0,136641
Average	0,476315789	0,010575479	0,476315789	0,010383555	0,472105263	0,023514458
Median	0,48	0,0100648	0,48	0,0100646	0,47	0,0100662
S. deviation	0,009551339	0,002023507	0,008950808	0,001425991	0,007873265	0,039934892

Figure A.1.: Robot touches his nose with a wall in front of him. Trajectory consisting of 30 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	1,56	0,0118806	1,57	0,0106394	1,59	0,00991183
0,1	1,54	0,0081277	1,54	0,00762281	1,54	0,00783408
0,15	1,57	0,00783261	1,57	0,00761206	1,52	0,00774957
0,2	1,58	0,00777038	1,53	0,0075898	1,54	0,00770248
0,25	1,53	0,00771643	1,54	0,00755702	1,54	0,00766302
0,3	1,56	0,00770253	1,54	0,0076171	1,54	0,00766675
0,35	1,57	0,0077038	1,55	0,00761299	1,62	0,00763528
0,4	1,54	0,00784141	1,53	0,00781902	1,54	0,00797945
0,45	1,53	0,0079057	1,54	0,00767664	1,53	0,00798911
0,5	1,54	0,00791806	1,57	0,00726061	1,56	0,00800083
0,55	1,54	0,00796235	1,62	0,00734494	1,61	0,00806611
0,6	1,53	0,0080302	1,55	0,00804394	1,55	0,00817191
0,65	1,58	0,0080489	1,65	0,0080621	1,54	0,00840271
0,7	1,53	0,00790182	1,54	0,00813057	1,59	0,00841436
0,75	1,63	0,00724818	1,54	0,00805447	1,57	0,00978397
0,8	1,56	0,0070809	1,64	0,00718591	1,54	0,11838
0,85	1,53	0,00790228	1,53	0,00810302	1,54	0,117917
0,9	1,55	0,00732147	1,53	0,00723133	1,64	0,105804
0,95	1,54	0,00699963	1,53	0,00717883	1,55	0,116944
Average	1,553157895	0,007941839	1,558421053	0,007807503	1,560526316	0,030632445
Median	1,54	0,00784141	1,54	0,0076171	1,54	0,00806611
S. deviation	0,025397426	0,001007865	0,037603658	0,000756215	0,033578049	0,044706032

Figure A.2.: Robot touches his nose with a wall in front of him. Trajectory consisting of 100 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	3,23	0,0121387	3,24	0,0112243	3,36	0,010074
0,1	3,24	0,00837103	3,22	0,00800442	3,26	0,00815338
0,15	3,23	0,00808522	3,16	0,0079693	3,19	0,00808669
0,2	3,21	0,008009	3,17	0,00789533	3,2	0,00799041
0,25	3,3	0,00794127	3,19	0,00780167	3,26	0,00791225
0,3	3,2	0,00788096	3,18	0,00778032	3,18	0,00784758
0,35	3,2	0,00783342	3,2	0,00774474	3,3	0,00774733
0,4	3,26	0,00802895	3,25	0,00802205	3,19	0,00823197
0,45	3,29	0,00809917	3,21	0,00791733	3,21	0,00816476
0,5	3,19	0,0081262	3,27	0,00756866	3,22	0,00822704
0,55	3,2	0,00816339	3,27	0,00760079	3,29	0,00834295
0,6	3,3	0,00820652	3,17	0,00828402	3,28	0,00849696
0,65	3,21	0,00831229	3,24	0,00835223	3,17	0,00874162
0,7	3,26	0,00784435	3,23	0,00837342	3,26	0,00884195
0,75	3,28	0,00728238	3,19	0,0071652	3,18	0,0997885
0,8	3,26	0,00719409	3,23	0,00737719	3,18	0,10031
0,85	3,22	0,00843968	3,28	0,00684959	3,2	0,0996402
0,9	3,23	0,00693111	3,3	0,00718255	3,21	0,0999741
0,95	3,25	0,00811231	3,2	0,00727211	3,25	0,100424
Average	3,24	0,008157897	3,221052632	0,007915012	3,231052632	0,032473457
Median	3,23	0,00808522	3,22	0,00780167	3,21	0,00834295
S. deviation	0,035276684	0,001042863	0,040674144	0,000904518	0,051628452	0,041480754

Figure A.3.: Robot touches his nose with a wall in front of him. Trajectory consisting of 200 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	0,48	0,0066207	0,46	0,0065949	0,47	0,00469523
0,1	0,48	0,00601621	0,46	0,00577956	0,48	0,00578056
0,15	0,47	0,00533426	0,47	0,00511464	0,47	0,00499597
0,2	0,46	0,00469981	0,47	0,00447197	0,49	0,00430806
0,25	0,56	0,00432677	0,47	0,00426847	0,49	0,00413877
0,3	0,47	0,00425888	0,48	0,00416351	0,47	0,00405881
0,35	0,47	0,00416054	0,47	0,00413749	0,49	0,00415274
0,4	0,47	0,00414205	0,48	0,00412046	0,56	0,00413054
0,45	0,61	0,00413216	0,46	0,00575044	0,47	0,00413243
0,5	0,48	0,00365721	0,48	0,00411019	0,46	0,00532224
0,55	0,47	0,0419604	0,47	0,213171	0,47	0,00553614
0,6	0,46	0,0349336	0,47	0,00214196	0,48	0,00473246
0,65	0,46	0,00403885	0,47	0,128702	0,48	0,00397766
0,7	0,53	0,136242	0,48	0,136565	0,48	0,00574192
0,75	0,48	0,0859767	0,48	0,0659358	0,49	0,00319797
0,8	0,48	0,0532668	0,46	0,0829655	0,49	0,121613
0,85	0,48	0,174833	0,48	0,17081	0,46	0,00527424
0,9	0,48	0,194655	0,46	0,198636	0,37	0,00528634
0,95	0,55	0,107831	0,48	0,111579	0,47	0,00414211
Average	0,491578947	0,046372944	0,471052632	0,061000942	0,475789474	0,010800905
Median	0,48	0,00601621	0,47	0,00577956	0,48	0,00469523
S. deviation	0,040724434	0,06291398	0,008093026	0,075467118	0,033385923	0,026843648

Figure A.4.: Robot moves arm from under the table to top of the table. Trajectory consisting of 30 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	1,57	0,00659244	1,57	0,00653982	1,53	0,00578198
0,1	1,61	0,00564408	1,55	0,00595382	1,54	0,0056552
0,15	1,59	0,0051465	1,56	0,00505329	1,56	0,00483729
0,2	1,56	0,00443786	1,55	0,00426972	1,62	0,00402259
0,25	1,53	0,00394664	1,55	0,00388516	1,6	0,00360914
0,3	1,56	0,00377144	1,58	0,00372801	1,55	0,00356645
0,35	1,56	0,00366407	1,54	0,00363367	1,55	0,00364976
0,4	1,58	0,00358797	1,54	0,00353234	1,56	0,00368179
0,45	1,57	0,00358004	1,54	0,00352573	1,56	0,00341853
0,5	1,54	0,00352941	1,55	0,00365958	1,56	0,00316161
0,55	1,56	0,00369366	1,55	0,00370833	1,58	0,00358896
0,6	1,53	0,0036464	1,57	0,00367138	1,59	0,004388
0,65	1,57	0,00372529	1,6	0,00365245	1,52	0,00479588
0,7	1,54	0,00372237	1,53	0,00370775	1,53	0,00400297
0,75	1,56	0,00323926	1,53	0,00424465	1,53	0,00387949
0,8	1,58	0,00357338	1,53	0,00371335	1,53	0,00427218
0,85	1,55	0,0012439	1,56	0,00371526	1,58	0,0467652
0,9	1,53	0,00370381	1,52	0,00335819	1,54	0,00415997
0,95	1,58	0,0544224	1,57	0,00328345	1,56	0,00438964
Average	1,561578947	0,006572154	1,552105263	0,004043997	1,557368421	0,006401402
Median	1,56	0,00370381	1,55	0,00370833	1,56	0,00402259
S. deviation	0,02167004	0,011636142	0,019882697	0,000873756	0,026841532	0,009800045

Figure A.5.: Robot moves arm from under the table to top of the table. Trajectory consisting of 100 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	3,22	0,00668646	3,28	0,00676355	3,19	0,00589557
0,1	3,25	0,00592657	3,26	0,00573169	3,22	0,0057346
0,15	3,25	0,00546804	3,25	0,00529161	3,24	0,00553267
0,2	3,21	0,00536017	3,25	0,00520343	3,2	0,00559538
0,25	3,32	0,00547038	3,33	0,00541913	3,23	0,00564249
0,3	3,19	0,00557052	3,21	0,00558847	3,27	0,0057678
0,35	3,21	0,00570431	3,28	0,00565662	3,22	0,00588392
0,4	3,22	0,0057089	3,26	0,00567819	3,21	0,00586177
0,45	3,2	0,00569318	3,23	0,00560181	3,21	0,00569208
0,5	3,17	0,00561577	3,23	0,00556739	3,26	0,00554632
0,55	3,22	0,00561469	3,22	0,00556814	3,21	0,00572433
0,6	3,21	0,00572544	3,23	0,00554087	3,2	0,00679353
0,65	3,21	0,00317354	3,24	0,00563151	3,23	0,00564558
0,7	3,23	0,0876872	3,43	0,00565825	3,31	0,00564083
0,75	3,22	0,00571883	3,26	0,054666	3,23	0,00574177
0,8	3,23	0,00561107	3,23	0,00573879	3,23	0,00622679
0,85	3,31	0,00576969	3,21	0,00557994	3,21	0,00652015
0,9	3,22	0,00566804	3,23	0,00244298	3,23	0,00629285
0,95	3,2	0,00238213	3,32	0,00558067	3,23	0,00654811
Average	3,225789474	0,009713417	3,260526316	0,008047844	3,227894737	0,005909818
Median	3,22	0,00566804	3,25	0,00558847	3,23	0,00574177
S. deviation	0,036562851	0,018906997	0,05264912	0,01131687	0,028003759	0,000377053

Figure A.6.: Robot moves arm from under the table to top of the table. Trajectory consisting of 200 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	0,48	0,00858674	0,48	0,00833613	0,48	0,00683878
0,1	0,49	0,00783564	0,47	0,00757114	0,46	0,0075814
0,15	0,47	0,00724094	0,47	0,00700342	0,48	0,00712617
0,2	0,47	0,00686887	0,47	0,00669975	0,48	0,00669651
0,25	0,47	0,00671328	0,47	0,00666817	0,46	0,00663167
0,3	0,48	0,00668753	0,47	0,00667785	0,47	0,00662251
0,35	0,48	0,00666014	0,48	0,00664064	0,46	0,00660005
0,4	0,48	0,00664353	0,46	0,00661909	0,47	0,00661637
0,45	0,49	0,00662503	0,3	0,00658292	0,36	0,00661727
0,5	0,34	0,00661749	0,41	0,00662263	0,46	0,00620967
0,55	0,41	0,00662133	0,33	0,00662275	0,48	0,00660513
0,6	0,36	0,00662118	0,3	0,00658139	0,48	0,00655835
0,65	0,42	0,00662195	0,48	0,00663677	0,47	0,00663836
0,7	0,33	0,00661618	0,41	0,00661555	0,46	0,00662083
0,75	0,21	0,00661618	0,48	0,00661904	0,48	0,00643333
0,8	0,37	0,00661628	0,47	0,00662121	0,47	0,00666541
0,85	0,23	0,00662167	0,46	0,00661648	0,47	0,00654539
0,9	0,48	0,0707203	0,3	0,00661557	0,47	0,00661528
0,95	0,39	0,00661367	0,47	0,00661718	0,47	0,00664277
Average	0,413157895	0,010218312	0,430526316	0,006787773	0,464736842	0,006677118
Median	0,47	0,00662503	0,47	0,00662263	0,47	0,00662083
S. deviation	0,08698995	0,014660317	0,068513882	0,000439549	0,026534761	0,000277742

Figure A.7.: Robot reaches from bottom shelf to top shelf. Trajectory consisting of 30 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	1,65	0,00872496	1,63	0,00850544	1,53	0,00760263
0,1	1,53	0,00771915	1,56	0,00764055	1,53	0,00739212
0,15	1,53	0,00680345	1,55	0,00672227	1,52	0,00668857
0,2	1,52	0,00649942	1,55	0,00642172	1,52	0,00654466
0,25	1,62	0,00636026	1,62	0,00635837	1,53	0,00625791
0,3	1,61	0,00633131	1,54	0,00619026	1,52	0,0063549
0,35	1,61	0,00632593	1,53	0,006125	1,54	0,00633137
0,4	1,54	0,00611909	1,54	0,00603215	1,6	0,00679456
0,45	1,56	0,00632989	1,54	0,00602254	1,52	0,00677872
0,5	1,54	0,00628406	1,55	0,00600501	1,52	0,00694371
0,55	1,58	0,00618127	1,53	0,00595552	1,58	0,00696847
0,6	1,55	0,00604115	1,54	0,00603504	1,55	0,00695569
0,65	1,65	0,00600212	1,55	0,00595452	1,52	0,00703663
0,7	1,52	0,00593954	1,54	0,206644	1,53	0,00690035
0,75	1,52	0,0055763	1,53	0,00604228	1,55	0,0916623
0,8	1,52	0,00601361	1,53	0,00602903	1,52	0,00707092
0,85	1,54	0,00618359	1,55	0,00603932	1,62	0,00708413
0,9	1,61	0,208884	1,55	0,233824	1,57	0,00794822
0,95	1,55	0,00613755	1,55	0,00603173	1,54	0,00695847
Average	1,565789474	0,017076666	1,551578947	0,028872566	1,542631579	0,011382859
Median	1,55	0,00628406	1,55	0,00604228	1,53	0,00695569
S. deviation	0,045131387	0,046453594	0,02733804	0,067590153	0,029597692	0,019445054

Figure A.8.: Robot reaches from bottom shelf to top shelf. Trajectory consisting of 100 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,05	3,31	0,00877765	3,36	0,00861207	3,31	0,00762306
0,1	3,3	0,00772349	3,29	0,00757812	3,31	0,00737272
0,15	3,29	0,00737093	3,29	0,00691608	3,36	0,00744362
0,2	3,28	0,00762408	3,33	0,00725421	3,34	0,00816527
0,25	3,32	0,00806367	3,32	0,00781658	3,3	0,00831187
0,3	3,29	0,00834496	3,3	0,00822697	3,32	0,00860566
0,35	3,3	0,00851412	3,32	0,00848035	3,36	0,00863699
0,4	3,3	0,00858841	3,39	0,00860718	3,41	0,00869954
0,45	3,3	0,00863489	3,31	0,00852695	3,3	0,00863713
0,5	3,31	0,0438068	3,39	0,00853317	3,34	0,00858645
0,55	3,31	0,0706042	3,36	0,0797251	3,29	0,00859155
0,6	3,31	0,0515243	3,37	0,0875162	3,31	0,00861668
0,65	3,3	0,00856159	3,32	0,0643891	3,31	0,00859357
0,7	3,37	0,0911449	3,35	0,0682229	3,3	0,00853491
0,75	3,3	0,0986779	3,32	0,0734994	3,3	0,00877172
0,8	3,3	0,107093	3,36	0,00870865	3,3	0,00877223
0,85	3,31	0,115237	3,32	0,0836358	3,3	0,010189
0,9	3,29	0,0775102	3,32	0,140579	3,37	0,0102048
0,95	3,29	0,127584	3,3	0,0945172	3,32	0,399643
Average	3,304210526	0,045546636	3,332631579	0,041123423	3,323684211	0,029157883
Median	3,3	0,00877765	3,32	0,00861207	3,31	0,00860566
S. deviation	0,018653504	0,044659174	0,031418697	0,042495214	0,031659741	0,089719925

Figure A.9.: Robot reaches from bottom shelf to top shelf. Trajectory consisting of 200 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,000125	124,25	0,0119849	125,01	0,0119849	124,94	0,0119849
0,00015	124,38	0,0119849	124,58	0,0119849	125,04	0,0119849
0,000175	124,47	0,0119849	124,52	0,0119849	124,5	0,0119849
Average	124,3666667	0,0119849	124,7033333	0,0119849	124,8266667	0,0119849
Median	124,38	0,0119849	124,58	0,0119849	124,94	0,0119849
S. deviation	0,1106044	0	0,26727015	0	0,287286152	0

Figure A.10.: Robot reaches from bottom shelf to top shelf. Trajectory consisting of 30 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,000125	422,37	0,0121689	422,45	0,0121689	423,67	0,0121689
0,00015	419,43	0,0121689	421,4	0,0121689	421,92	0,0121689
0,000175	421,39	0,0121689	420,32	0,0121689	422,74	0,0121689
Average	421,0633333	0,0121689	421,39	0,0121689	422,7766667	0,0121689
Median	421,39	0,0121689	421,4	0,0121689	422,74	0,0121689
S. deviation	1,496974727	0	1,065035211	0	0,875576001	0

Figure A.11.: Robot reaches from bottom shelf to top shelf. Trajectory consisting of 100 waypoints.

Step size	First version		Second version		Third version	
	Time	Distance	Time	Distance	Time	Distance
0,000125	854,74	0,0121663	859,32	0,0121663	859,24	0,0121663
0,00015	855,68	0,0121663	857,92	0,0121663	861,04	0,0121663
0,000175	859,8	0,0121663	857,91	0,0121663	858,48	0,0121663
Average	856,74	0,0121663	858,3833333	0,0121663	859,5866667	0,0121663
Median	855,68	0,0121663	857,92	0,0121663	859,24	0,0121663
S. deviation	2,691393691	0	0,811192538	0	1,314736983	0

Figure A.12.: Robot reaches from bottom shelf to top shelf. Trajectory consisting of 200 waypoints.

Bibliography

[DD09] E. Deza, M. M. Deza. *Encyclopedia of Distances*. Springer, 2009. (Cited on page 21)

[FH04] P. F. Felzenszwalb, D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, 2004. (Cited on page 22)

[JS96] J. E. D. Jr., R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. 1996. (Cited on page 13)

[Kal] Distance Field for ROS. URL http://wiki.ros.org/distance_field. (Cited on page 22)

[KCETS11] M. Kalakrishnan, S. Chitta, P. P. Evangelos Theodorou, S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*. 2011. (Cited on page 19)

[KF11] S. Karaman, E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. In *International Journal of Robotics Research*, volume 30, pp. 846–894. 2011. (Cited on page 9)

[LaV06] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/. (Cited on page 11)

[NRS09] J. A. B. Nathan Ratliff, Matt Zucker, S. Srinivasa. CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In *IEEE International Conference on Robotics and Automation (ICRA)*. 2009. (Cited on page 18)

[NW06] J. Nocedal, S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006. (Cited on page 13)

[Ran12] M. T. Rantanen. Robot Motion Planning with Probabilistic Roadmaps. Technical report, Computer Sciences, School of Information SciencesUniversity of Tampere, 2012. (Cited on page 9)

[Ros] ROS - The Robot Operating System. URL http://www.ros.org/. (Cited on page 19)

All links were last followed on August 17, 2014.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature