Master's Thesis

Development of a tool chain to integrate different robot manipulators in ROS

Clemens Mühlbacher

Graz, 2013

Institute for Software Technology
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa
Second reviewer: Ass.Prof. Dipl.-Ing. Dr.techn. Gerald Steinbauer

Abstract

In this thesis the arm navigation stack of the ROS framework will be evaluated for their applicability for different robot arms in particulate popular research robot arms. The arm navigation consists of several algorithms which are performed after each other in a tool chain. Will will shown in the empirical evaluation that some of the algorithms for specific sub-task such as path planning do not always find a solution to a given manipulation task. Thus the complete tool chain fails. We identified the major problems which cause the tool chain to fail. In particular the default inverse kinematics algorithm causes the tool chain often to fail. To overcome this problem an novel inverse kinematic algorithm based on a global optimization algorithm was developed. The empirical evaluation shows that the developed algorithm significantly improves the performance of the entire tool chain.

Kurz Zusammenfassung

In dieser Arbeit wird der Arm-Navigation-Stack welcher in ROS vorhanden ist anhand von verschieden Manipulatoren, welche vor allem beliebt im Forschungsbereich sind, evaluiert. Der Arm-Navigation-Stack besteht aus verschiedenen Algorithmen welche einem nach dem andern aufgerufen werden. Diese Kette an Algorithmen, welche jeweils eine Teilaufgabe lösen, formen eine Tool-Chain. Mithilfe einer empirischen Evaluierung wird gezeigt das einzelnen Algorithmen in Kombination mit verschieden Manipulatoren eine erfolgreiche Ausführung für eine gegebene Manipulationsaufgabe nicht möglich machen. Dadurch ist eine erfolgreiche Ausführung der Tool-Chain nicht möglich. In dieser Arbeit werden die schwerwiegendsten Probleme welche ein Scheitern der Tool-Chain auslösen aufgezeigt. Besonders zu beachten ist das der Algorithmus welcher verwendet wird um die Inverse Kinematik zu berechnen besonders problematisch ist da dieser nicht für jeden Manipulator eine Lösung findet. Um dieses Problem zu lösen wurde ein Invers Kinematik Algorithmus entwickelt, welcher einen globalen Optimierungs-Algorithmus verwendet um eine Lösung zu finden. Mithilfe der empirischen Evaluierung wird gezeigt das dieser Algorithmus eine deutlich bessere Performance aufweist und somit zu einer besseren Performance der gesamten Tool-Chain beiträgt.

Acknowledgement

I would like to thank all colleagues who supported me during the work on my master thesis with their fruitful discussions. Moreover I want to thank Steinbauer and Wotawa, who guided me through the whole thesis work and with all the upcoming questions about the different parts of the thesis. Additionally I want to thank the people of the IncubedIT[1] for there support. Last but not least I would like to thank my family, which supported my during my studies. This thesis was partly funded by the the Austrian Research Promotion Agency (FFG) with the ”Innovationsscheck” program.

[1] http://www.incubedit.com/team

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Place, Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Ort, Datum Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The introduction presents the main motivation for the need of extending the existing tool chain, which is used to perform manipulation tasks of the Robot Operating System (ROS) [1] [3]. Then we will state the problem we focus on within this thesis more formally. Also the challenges and goals which arises from the problem stated before will be discussed. We also will briefly discuss the contributions of this thesis and will outline the structure of the thesis at the end of this chapter.

1.1. Motivation

Mobile manipulation is a hot topic in robotic research. The basic task in mobile manipulation is to position some part of a manipulator on e specified position. To achieve this a simple approach is to use a sense-plan-act scheme to perform its task, which is in contrast to other used methods, like for example [4]. Even with this simple approach and all its limitations to the possible task, which could be performed, it is not always possible to perform the task successfully.
It would be promising if a manipulator could simply place one specified part at a certain desired position. If this task could be fulfilled with a manipulator it could be used to perform more complex tasks, like for example to pick up an object which has got a known grasping point with the manipulator. This task can be processed with few positioning steps. First, the manipulator is positioned to grasp the object with the gripper opened. Afterwards the gripper is closed and the object is lifted to a certain position. An example of such a task can be found in [5]. It is obvious that if the positioning of the manipulator can be solved, the goal to pick up an object can be archived.
So it would be promising if it is possible to perform this task with any manipulator. But at the moment it is quite an effort to achieve this simple task with any given manipulator using off-the-shelf compound open source. Some manipulators are supported by ROS. There are some other packages which tackle the problem for another manipulator and some manipulators are not supported at all.
To provide a general solution which can be used with different manipulator different algorithms are used to perform this positioning task. Not all of the algorithms currently used finish with a success and thus reduce the success to perform the positioning task. This problem especially arise at the calculation of the inverse kinematics.
To check how successful the algorithms are different evaluations have to be performed which cover all aspects of the positioning task. Unfortunately such a set of evaluations does not exists yet.

[1] http://www.ros.org/wiki/

1.2. Problem Description

As also discussed in Section 1.1 the problem we are facing is the usage of different manipulators to perform simple positioning tasks in a general way. So the main goal of this thesis is to make it easier to use a manipulator for a primitive manipulation task within ROS. The task is to position the end effector, which is the last link of the manipulator, or another part of the manipulator with a certain position and orientation within space, or more formally given a manipulator description and a desired pose of one part of the manipulator, which could be reached, move the manipulator in such a way that this pose is reached without any collision with the known environment. Another important problem which will be tackled within this thesis is a method to evaluate the performance to point out weaknesses as well as to open up the opportunity to show how an improvement impacts the overall performance.

1.3. Challenges and Goals

The main challenges to perform this simple positioning task is to handle all problems in a general way to use different manipulators. The goal is to use methods, which perform the positioning task in such a way that they are not specific to a particular manipulator. Another challenge is an evaluation, which considers a manipulator as well as a possible environment is a challenge. The goal which rises from this challenge is to evaluate the performance of the used algorithms in a general way. Another important challenge is the execution time, which is spent to generate a plan, which should be, as a goal, as short as possible. The last challenge which should be mentioned is that the execution should not damage the manipulator. Out of this challenge there are two goals to archive. First of all collisions with the environment should be avoided. The second goal is to consider the limits of the manipulator, which are position, velocity and acceleration limits of the joints.

1.4. Contributions

This thesis provides two major contributions. First an evaluation which evaluates each part independently, as well as an overall evaluation which is performed with various combinations of algorithms and manipulators. The second contribution is a general way to calculate the inverse kinematics with a high success rate, compared to other general methods. This general calculation of the inverse kinematics find a solution, even in cluttered environments, where other methods do not find a solution. The method is based on a global optimization technique. The developed algorithm was furthermore published in [6].

1.5. Organisation of the thesis

In Chapter 2 we will give a short introduction to the system we use to perform the task and explain the principles of manipulators and the different steps of the tool chain in a formal way. Afterwards we will give an overview of related research (Chapter 3) of the different steps of the tool chain as well as on solutions which consider the whole task. The next chapter explains how the positioning task was tackled within the system (Chapter 4). The evaluation of all parts of the tool chain will be discussed afterwards in Chapters 5. We will conclude the thesis with a briefly conclusion in Chapter 6 and possible future work in Chapter 7.

Chapter 2

Prerequests

Within this chapter we will give an overview of the base system (in Section 2.1) that is used to introduce the tool chain. After this short overview we will briefly explain how a manipulator is built and the terms which are used later on (in Section 2.2). We will also describe how a manipulator can formally be described (in Section 2.3). After this more general descriptions we will briefly explain how the tool chain is constructed (in Section 2.4) to solve the task and explain the problems, which are solved in each of the steps of the tool chain in separate sections.

2.1. ROS

In this thesis we will use ROS[1] and extend the existing tool chain to execute primitive manipulation tasks with different manipulators. To understand why it is easy to split up the task in several components we will briefly explain ROS and point out some other advantages of ROS.
ROS is a software framework for robotics applications. ROS uses so-called nodes[2] as building blocks for a robot software system. These nodes interact transparently with each other. This is done through a publish-subscriber principle, which is implemented through so called topics[3]. These topics are used to publish data and to subscribe data from such a topic (a simple example is shown at Figure 2.1). Thus it is simple to use different nodes to serve different tasks without side effects to other tasks which may occur if the system would be build as a monolithic block. ROS also provides a package system[4], which makes it possible to easily share and reuse work. This opens up the possibility to simply reuse the improvements within different systems. Last but not least a wide range of researchers use the system to implement their work and make it easily available through the package system.

2.2. Robot Manipulators

Now we briefly explain how a robot manipulator is built up and introduce some terms that will be often used later on. A more detailed explanation on how to describe a manipulator will be given later on in Section 2.3.
A robot manipulator consists of two basic building blocks which can be seen in Figure 2.2. The first one is the link which is a rigid body. In general a link is even element and has got a specified fixed length.

[1] http://www.ros.org/wiki/
[2] http://www.ros.org/wiki/ROS/Tutorials/UnderstandingNodes
[3] http://www.ros.org/wiki/ROS/Tutorials/UnderstandingTopics
[4] http://www.ros.org/wiki/ROS/Tutorials/NavigatingTheFilesystem

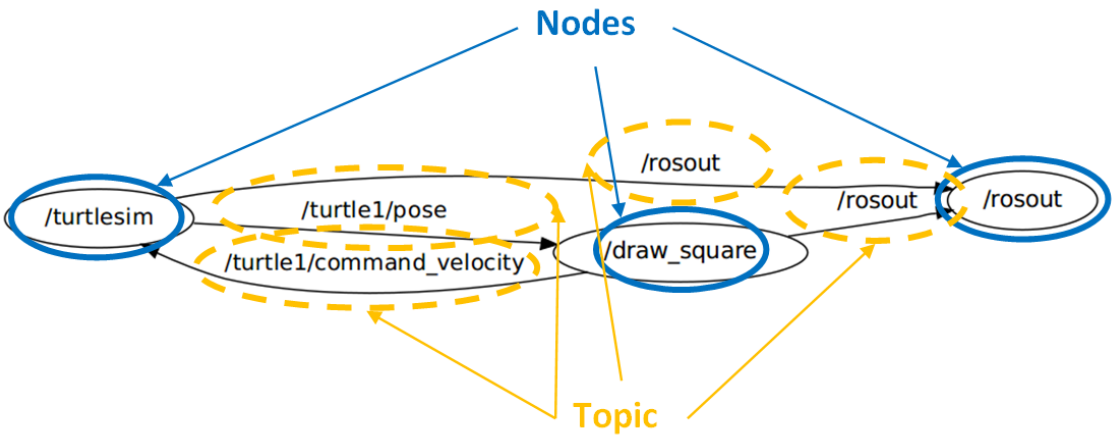

Figure 2.1.: rxGraph (`http://www.ros.org/wiki/rxgraph`) of three ROS nodes with topics which are published and subscribed. (The arrow points from the publisher to the subscriber)

The second building block is the joint. The joint connects two links. Through the joint it is possible to move one link relative to another link. There are many different types of joints: revolute (see Figure 2.3), prismatic (see Figure 2.4), cylindrical (see Figure 2.5), spherical (see Figure 2.6) and universal joints (see Figure 2.7). We will focus on revolution joints since they are used very often to build a robot manipulator. Each of these joints can have a certain joint position, which is expressed with one or more values depending on the type of the joint. As mentioned above we will focus on the revolution joint which has one value, the rotation angle around the joint axis measured from an initial pose. Each of the joints may have some limitation for the range of values.
A simple example of is a part of the human arm. The links are the forearm and the upper arm. The joint is the elbow, which cannot rotate 360 degrees around its axis and thus have joint limits.
A term which we will use later on is the kinematics chain. It is defined as a set of links and joints which are connected to each other. The kinematics chain is called closed if it contains a circle (a path from on link to another is possible if they are connected with a joint) and open if it contains no circle. We will focus only on open kinematics chains since the manipulator in this theses are all of this type.
To specify a specific constellation of the manipulator the value for each joint is specified. This value specification of each joint is called a configuration.

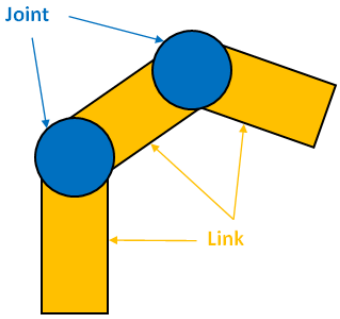

Figure 2.2.: Simple manipulator with three links and two revolution joints

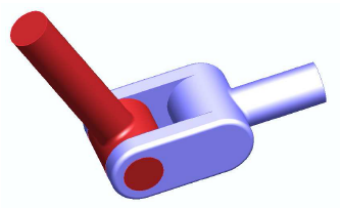

Figure 2.3.: Revolution joint (image from [1])

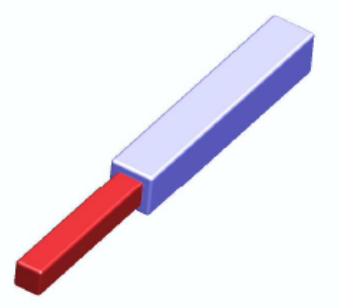

Figure 2.4.: Prismatic joint (image from [1])

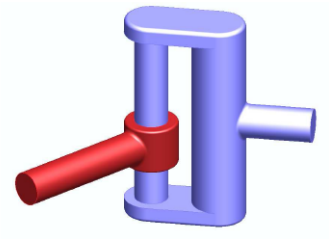

Figure 2.5.: Cylindrical joint (image from [1])

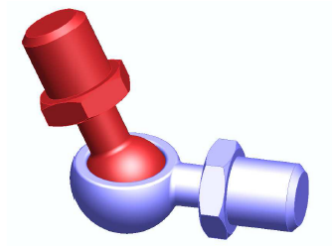

Figure 2.6.: Spherical joint (image from [1])

2.3. Manipulator description

Within this section we will explain how to formally describe a manipulator. We will focus only on the description of a simple manipulator with rigid body elements and an open kinematics chain.
As mentioned in the introduction we consider a manipulator to consist of links and joints. There are different methods to describe how these links and joints are connected to each other. We will discuss two popular widely used methods. The first one, which is called Denavit-Hartenberg parameters, is very common and can be used in general. The second method, which is called URDF (Unified Robot Description

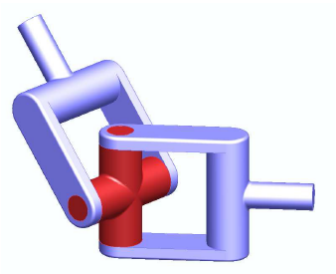

Figure 2.7.: Universal joint (image from [1])

Format) [5] is used to represent a robot (and also a manipulator) within ROS.
Other important facts are the limits for the joints, which must also be considered during the different planning steps and therefore have to be part of the description. This information can be stored within the description, which is possible in URDF, or separately, which has to be done if the Denavit-Hartenberg parameters are used and the parameters are not extended to store the information.

2.3.1. Denavit-Hartenberg parameters

The Denavit-Hartenberg parameters [7] can easily be used to describe manipulators with revolution or prismatic joints. The manipulators we are focusing on use revolution joints. Thus it is possible to use these parameter design. To make it more precise the parameters describe how the coordinate frames (the coordinate system associated with a specified link) are transformed into each other. To use the parameter there are some conditions that have to be fulfilled.

1. The Z-Axis of the current joint has to be within the rotation axis of the joint, except if the joint is a prismatic one.
2. The X-Axis of the current joint has to be the cross product of the Z-Axis of the current joint and the Z-Axis of the previous joint.
3. The Y-Axis of the current joint is chosen in such a way that the three axis build a right hand coordinate system.

This requirements can be easily fulfilled if the frame coordinate system is placed within the joint corresponding to the frame (the link with the number i corresponds to the joint of the number i, if both of them are numbered from the same base link).
The parameter itself describes four transformations, which can be performed one after the other to transform one coordinate system (called the previous) into another one (called the current).

1. Translate the frame along the previous Z-Axis to reach the position where the previous Z-Axis intersects the current X-Axis. This parameter is called d and specifies the distance of the joints (see Figure 2.8).
2. Rotate around the previous Z-Axis to align the previous to the current X-Axis. This parameter is called θ and specifies the fixed joint rotation from the previous joint to the current joint around the axis of the previous joint (see Figure 2.9).
3. Translation along the rotated X-Axis until the origin of the current coordinate system is reached. This parameter is called a and specifies the length of the rotation radius (see Figure 2.10).
4. Rotate around the transformed X-Axis to get the Z-Axis. This parameter is called α and specifies the twist around the current X-Axis (see Figure 2.11).

[5] http://www.ros.org/wiki/urdf

It can be seen that a transformation from one frame to another frame can be achieved easily if all the parameters are given. An example transformation can be seen in Figure 2.12. It is also possible to convert these transformations into other transformations for example a matrix representation if such a calculation could be performed faster.
It should also be mentioned that the parameters are not uniquely defined for a manipulator. For example if the Z-Axis is parallel there can be an infinite number of d values.
An important aspect to mention is that these parameters do not specify any joint limits.

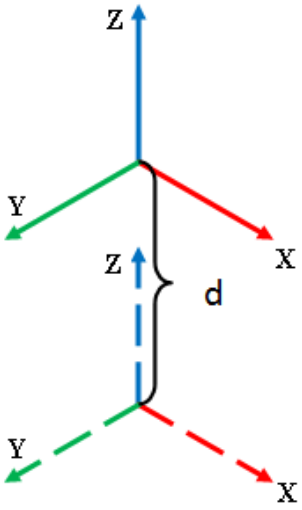

Figure 2.8.: First transformation with the help of Denavit-Hartenberg parameter d. The dashed coordinate axes are associated with the previous coordinate system

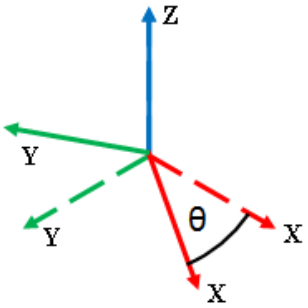

Figure 2.9.: Second transformation with the help of Denavit-Hartenberg parameter θ. The dashed coordinate axes are associated with the previous coordinate system

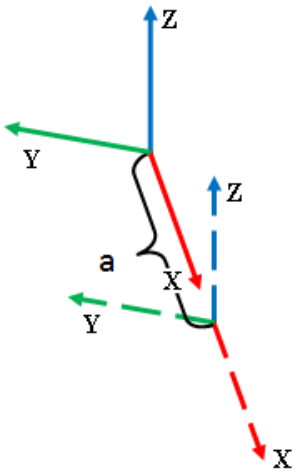

Figure 2.10.: Third transformation with the help of Denavit-Hartenberg parameter α. The dashed coordinate axes are associated with the previous coordinate system

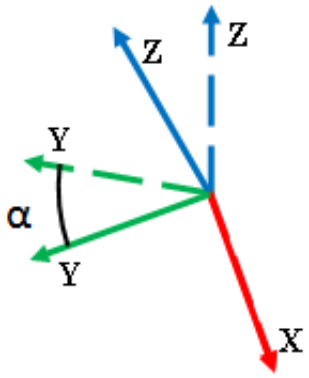

Figure 2.11.: Fourth transformation with the help of Denavit-Hartenberg parameter *a*. The dashed coordinate axes are associated with the previous coordinate system

2.3.2. URDF

Even if the Denavit-Hartenberg parameters are very common within the field of robotics we use another description to specify a manipulator. The description is called URDF [6] (Unified Robot Description Format) and is coded in a XML format. The URDF contains much more information which can be used within the field of robotics. We will only focus on the parts we use for solving the subtasks. To describe a manipulator we use the model description within URDF. The description contains two element types: the link elements and the joint elements.
The links specify many properties to handle the visualization, collision, and mass distribution. The collision information is used to check if a posture (specified for the manipulator through its structure and the joint values) is in collision with itself or the environment [7]. The mass distribution can be used to calculate the forces for the manipulator (e.g. Simulation).
The second element of the URDF model specification is the joint specification. It contains the name of the links, which are connected through the joint. One is called the parent link, the other one is called the

[6] `http://www.ros.org/wiki/urdf`
[7] For more information how collision information are used and how they are processed within ROS the webpage `http://www.ros.org/wiki/planning_environment` is a good starting point

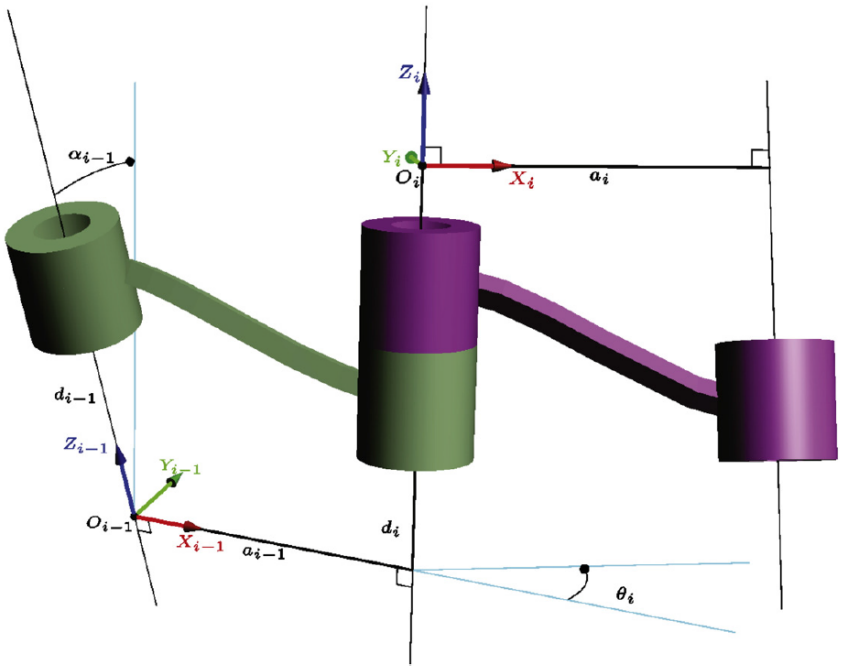

Figure 2.12.: Complete coordinate system transformation with Denavit-Hartenberg parameter. Image taken from [2]

child link. This terminology comes from the fact that an open kinematics chain can be represented in a tree structure starting from the base link (the base of the manipulator is fixed) down to the end effector link. The specification of the joint also contains a limit for the joint, limits for the joint velocities, as well as limits for the effort (how much effort can the controller command). There is also a parameter which specifies the correspondence between joint limits and joint velocity as well as joint velocity and joint effort [8]. Another set of parameters contains information about dynamics of the joint. The last important information which is also contained within the joint specification is the transformation between the two links which are connected through the joint. The transformation is given through a translation along the different axes and three rotations which are performed one after the other. The first rotation is a roll around x followed by a pitch around y and afterwards a yaw around z. Through the translations and rotations a complete transformation of the frame is possible. Finally the joint contains information about its type.

URDF example

After we explained a URDF in general we will look at an example. We will discuss each of the different parts of the URDF file, which were discussed above.

- The "*robot*" XML-tag specifies the robot, which is described within the URDF file. Moreover the name within the XML-tag is used to identify the robot, which is used later on to identify the robot.
  - The "*link*" XML-tag is used to identify a link. The name within the XML-tag is used to identify the link within the system.
    - The "*inertial*" XML-tag contains the mass of the link as well as the mass position and orientation, relative to the link origin.

[8] For more information about correspondence between joint limits and joint velocity as well as joint velocity and joint effort see `http://www.ros.org/wiki/pr2_controller_manager/safety_limits`

    - The "*visual*" XML-tag contains the information of the visual representation of the link as well as the relative position to the link origin. Also the material is specified to show the visualization as well as the geometry of the visualization. The geometry can either be a box, cylinder, sphere or a mesh. The mesh can be of arbitrary geometry since it is a simple mesh representation of the object.
    - The "*collision*" XML-tag specifies the origin as well as the representation of the collision model. The collision model is represented by a geometry information. This geometry information can contain the same elements as the geometry information within the visualization. This model is often a simplified version of the Visualization model due to the expensiveness of the collision checks.

  – The "*joint*" XML-tag contains the name of the joint, the type as well as the joint specification below.
    - The "*parent*" XML-tag contains the name of the parent link which is connected to the joint.
    - The "*child*" XML-tag contains the name of the child link which is connected through this joint.
    - The "*origin*" XML-tag of the joint specifies the origin of the joint which is relative to the parent origin. This origin is also the origin of the child link.
    - The "*axis*" XML-tag specifies the axis of the joint. In case of the revolution joint the axis specifies the rotation axis of the joint.
    - The "*limit*" XML-tag specifies the joint limits of the joint as well as the maximum joint velocity.

The resulting model is shown in Figure 2.13.

```
<?xml version="1.0"?>
<robot name="plate_with_box">
  <link name="the_plate">
    <inertial>
      <mass value="1.2" />
      <origin xyz="0.2 0.2 0.03" rpy="0 0 0" />
      <inertia ixx="0.001" ixy="0.0" ixz="0.0" iyy="0.001" iyz="0.0" izz="0.001" />
    </inertial>

    <visual>
      <origin xyz=" 0 0 0 " rpy="0 0 0" />
      <geometry>
    <box size="0.4 0.39 0.03"/>
      </geometry>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>

    <collision>
      <origin xyz=" 0 0 0 " rpy="0 0 0" />
      <geometry>
    <box size="0.4 0.39 0.03"/>
      </geometry>
    </collision>

  </link>
  <!-- "joint" to back box -->
  <joint name="the_plate_box_joint" type="revolute">
    <parent link="the_plate"/>
    <child link="the_box"/>
    <origin xyz="0.17 0 0.045" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
```

```
    <limit effort="1000.0" lower="-0.5" upper="0.5" velocity="0.5"/>
  </joint>

  <link name="the_box">
     <visual>
      <origin xyz="0 0 0 " rpy="0 0 0" />
      <geometry>
    <box size="0.06 0.38 0.10"/>
      </geometry>
      <material name="grey">
        <color rgba="1 1 1 0.5"/>
      </material>
    </visual>
  </link>
</robot>
```

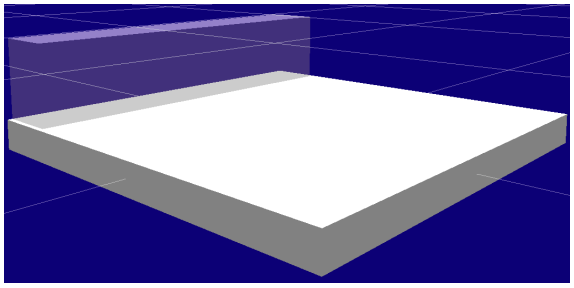

Figure 2.13.: Result of the example URDF viewed in rvize (`http://www.ros.org/wiki/rviz`).

2.4. Brief explanation of the tool chain

We will now briefly explain how we perform the position task. A very simple approach to fulfill the task of positioning the end effector of a manipulator, is to split up the problem into subproblems as described later on in Chapter 4. There are approaches which treat the task as one big problem, one example is [8]. But we will divide the problem into smaller subproblems. This approach is very useful to focus on some parts of the problem since some parts of the task depend more on the manipulator specific properties than others. We split up the problem into four subproblems, which can be seen in Figure 2.14, to solve the task. Within the next sections we will explain the problem descriptions of the first three steps. The last step of the tool chain is a simple execution of the generated plan. As we do not focus on control it will not be explained within this thesis.

2.5. The inverse kinematics problem

The first of four mentioned subproblems is the inverse kinematics problem. To get a better understanding of the inverse kinematics problem we will first describe the simpler forward kinematics problem. The two problems are strongly related to each other as it can be easily seen in Figure 2.15. The solution of the forward kinematics problem is also used to verify the results of the inverse kinematics problem and we will explain how it is used within the evaluation of the tool chain in Chapter 5. Both problems are mappings between the control space of the manipulator to the Cartesian space. The different joint span a vector space which is the so called control space of the manipulator (for more information we revere the reader to [9]). On the other side the Cartesian space consists of an translation part $\mathbb{R}^3$ and an rotational part $SO(3)$. To specify one constellation in the Cartesian space we use a pose which specify the translation part as well as

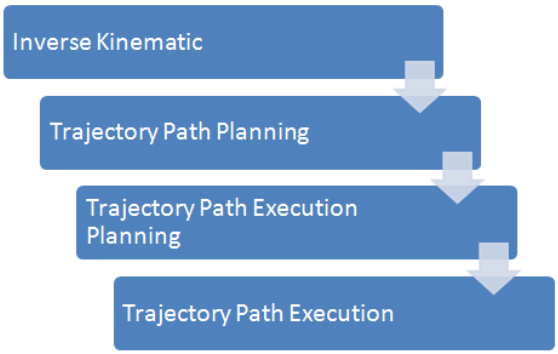

Figure 2.14.: The different steps of the tool chain

the rotational part. Within the Cartesian space it is also important to mention that this coordinates are all relative to a specified coordinate system. As mention above a coordinate system, which is associated with a link is a so called frame.

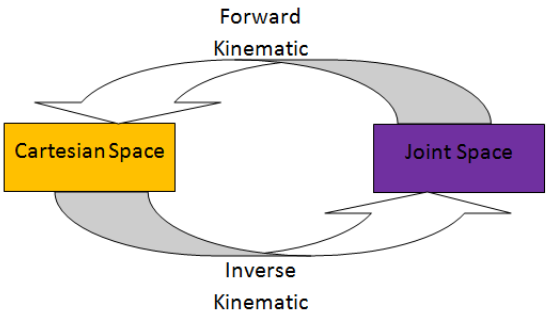

Figure 2.15.: Transformation between the joint and the Cartesian space

2.5.1. Forward kinematics problem

The forward kinematics problem is defined as follows: given the manipulator description and the joint values, determine the end effector pose (see Figure 2.16). The forward kinematics problem can be solved very easily through the given transformations of the robot description (it is important to mention that we only consider open kinematics chains). The joint values are used for determining the transformation from one frame to another frame. Each of these transformations can be represented through a matrix multiplication which uses homogeneous coordinates. The result is a number of matrix multiplications to transform one frame into another. These matrices can be concatenated to specify the full transformation from the base frame to the end effector. For example if we consider the manipulator in Figure 2.2 we get 5 matrices. The first is a transformation for the first link to the first joint A_1 the second is the transformation which is performed through the first joint B_1. Afterwards we transform along the next link with A_2, rotate with the last joint B_2 and translate along the last link with A_3. The different A_i matrices are all translation matrices along the links and the B_i are rotational matrices around the joints. The resulting transformation$T$ from the first link to the last link is given through the equation $T = A_3 * B_2 * A_2 * B_1 * A_1$.
The forward kinematics always has a unique solution if we consider open kinematics chains. All these properties make the problem easy and fast to compute. There are also other methods to calculate the forward kinematics problem, for example in a recursive way [10]. All methods have in common that the they are very fast in comparison to the inverse problem and there exist methods which solve the forward kinematics problem in a stable and fast way for any type of manipulator.

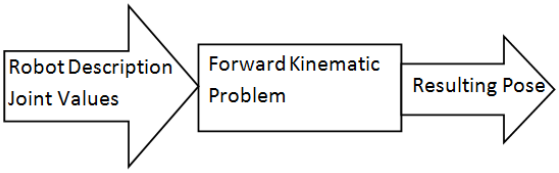

Figure 2.16.: Transformation from the joint into the Cartesian space

2.5.2. Inverse kinematics problem

The inverse kinematics problem is defined as follows: given the manipulator description and a desired pose of the end effector, determine the joint values which result in a manipulator pose with the end effector at the desired pose (see Figure 2.17). We will not focus on more complex versions of the problem with multiple goals of the manipulator as it is proposed in [11]. In contrast to the forward kinematics problem there is no closed form solution for every manipulator possible. It has been shown that there exist closed form solutions for manipulators with four [12] and five [13] degrees of freedom. There are also closed form solutions for some geometries for manipulators with six degrees of freedom ([14] and [15]) as well as for some geometries with seven degrees of freedom [16]. There are also closed form solutions known if one of the joint values is known for example [17], [18]. All these methods show the fact that they can solve the inverse kinematics problem only for a special type of manipulator and are often not as easy to implement in a generic way as it is possible for the forward kinematics problem. It is possible to derive closed form solutions for some manipulators by hand but this would contradict our motivation for generic tool chain.

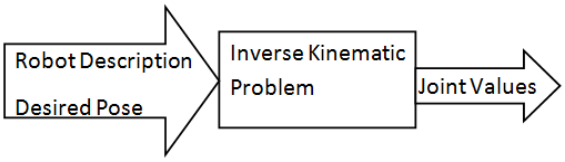

Figure 2.17.: Transformation from the Cartesian into the joint space

2.6. The trajectory path planning problem

After we discussed the inverse kinematics problem we will focus on the next step within the tool chain. The problem is how to plan a possible path to move the manipulator to its end position. This problem consists of two parts, namely the trajectory path planning problem and the trajectory path execution planning problem. We will divide the planning of the trajectory into these two steps as it was proposed in [19] or [20]. Another possible solution to tackle the trajectory planning would be to solve this problem as a whole. This type of planning is known as kinodynamic motion planning. This problem can be solved for example with the help of the proposed algorithms in [21] or [22]. We will discuss the decision why we divide this problem in such a way in Chapter 4.
From the previous step we have the joint values from the end position. The current position is also known from the robot. This leads to the problem to find a path from the current joint position to the joint positions of the end effector without violating the joint limits. It is also important that the environment is checked to avoid collisions with obstacles in the environment (this also includes the manipulator itself). This problem is known as the path planning problem. To define the problem to so called configuration space is used. This configuration space consists of all possible configuration a manipulator can have. Thus this space is the cross product of all possible joint values (for a more detailed specification see [9]). The problem is defined as to find the shortest possible path within a configuration space (see Figure 2.18). The path is

only possible if there are no collisions or other violations of constraints (in the case of a manipulator these constraints are the joint limits) on the path.

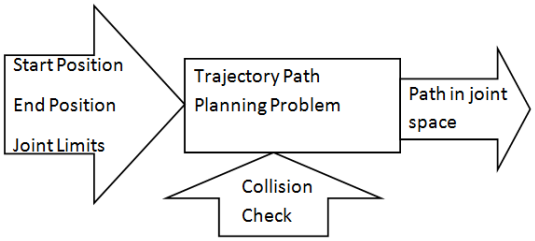

Figure 2.18.: Find a path from start to end position within joint space

This problem is also known as the the general movers problem, which is known to be PSPACE-complete [23]. This means that in the worst case the best algorithm uses a polynomial amount of space. It is makes also no difference between a deterministic and a non-deterministic algorithm, both will have at least a polynomial space consumption. It is important to notice that there is no efficient algorithm (i.e. an algorithm which solves the problem in polynomial time) known for PSPACE-Complete problems. This is also the reason why the problem is split up in two parts. Thus the problem instance become smaller and can be solved faster. Otherwise the problem state space does not only consists of the joint values. Instead it would also add an additional dimension which would represent the time of the position. Another impact of the time complexity of the motion planning problem yields to different algorithms which use randomized methods to overcome the complexity of this problem. Randomized algorithms are often used to deal with complex problems, with the known issue that these algorithms possible do not find a solution. Some examples of such algorithms are [24], [25]. One important aspect of this part of the tool chain is that it does not handle moving objects in this case. To be more precise we expect that the objects in the environment do not move during the execution of the movement of the manipulator. This is important because otherwise it would not be possible to divide the trajectory planing into two parts. There are problems with moving objects which can be addressed with this division of the trajectory planning (for example see [19]) but in general moving objects cannot be addressed. One justification why we do not consider the movement of objects is because many tasks with manipulators take place within environments which satisfy this constraint. For example if the robot is used within a home environment it can be assumed that the objects themselves will not move during an action with the manipulator. This assumption seams valid since the most objects cannot move by themselves. Another example would be an industrial environment with a fixed working space or a box which moves with the assembly line. The movement of the assembly line and the objects at the line do not mater since the manipulator also moves in the same way and there is no relative movement. Additionally it would dramatically increase the execution time of the tool chain, if moving objects would be considered.

2.7. The trajectory path execution planning

The last problem we will consider within this chapter is the trajectory path execution planning. Out of the previous step we have generated a path within the joint space. This path starts at the current joint configuration and ends at the joint configuration which was calculated through the inverse kinematics. Along this path several joint configurations are specified which have some maximal distance to each other. To specify it more precisely the output of the previous step is a list of joint configurations which have to be performed one after the other by the manipulator.
The problem we are facing in this section is how to find a path execution which follows the path given from the previous step as closely as possible and with a minimum amount of time (see Figure 2.19). It is also possible to reduce other types of costs but we will not consider such optimizations. It is also important to mention that there are several constraints, which have to be considered during the planning

phase. One constraint set consists of the joint limits, which also have to be considered at the previous steps. Other constraints are the velocity, acceleration limits, the maximum torque and the jerk of the joint. All these limits have to be specified within the robot description for each joint. It is important to note that this problem has another condition which has to be considered. The time to execute this planning step and the execution time of the manipulator movement itself should be as short as possible. This also means that it makes no sense to find a possible optimal solution, which means the solution with the lowest possible execution time, and discard suboptimal solutions, with a slightly higher execution time, if the time difference of the calculation, which yields the optimal solution and the calculation, which yields the suboptimal solution, is bigger than the time difference of the optimal and the suboptimal solution. Another important aspect we want to point out is that we consider all these different limits to avoid problems, which occur if the theoretical trajectory cannot be executed correctly via the manipulator. For instance the trajectory calculated within this step uses a velocity which cannot be achieved, this trajectory would result in a wrong final joint position. This wrong joint position leads to a wrong position of the manipulator. It is also important to note that we do not deal with moving objects as we discussed in previous sections. This would be possible and was already discussed, for example in [19].

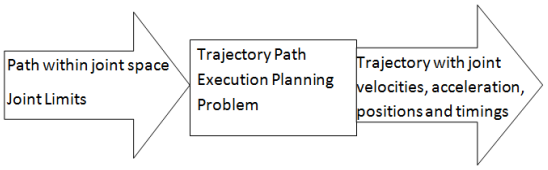

Figure 2.19.: Find joint velocities, accelerations, positions and timings to follow the given path

Chapter 3

Related Research

Within this chapter we will discuss related tools which offers similar functionality as well as tools and methods which tackle one of the problems introduced in the previous chapter.

3.1. Tool chain to handle a complete positioning task

There are some tools which support the whole task to position a part of the manipulator. We will review three of them and point out their drawbacks.
First of all it is important to note that ROS supports a tool chain for manipulators [1] (which is also described in [5]) but as stated above there are problems with the general use of this tool chain. The impact of the change on the tool chain, which were performed, can be found within the evaluation chapter 5, which also shows that without these changes the usage of the tool chain is limited.
Another tool chain for manipulation is provided through OpenRave [2] which is a powerful system, which supports the simple position task as well as more complex tasks. Within OpenRave the task can be specified and executed on a robot. To execute a task the system uses different components. On component is a a database, with many information about the robot. This database was generated during a "setup phase". Another component, which is used within OpenRave, holds the current sensor data. For motion planing two components are used. The first one is an inverse kinematics computation which will be discussed within Section 2.5 in more detail. The second component is a modified RRT (Rapidly-exploring Random Tree)[26] path planning algorithm which grows two random expanding trees to each other and tries to find a feasible path. The modified RRT is explained in more detail in [27]. A more detailed description about multiple similar motion planning algorithms will be given in Section 2.6. OpenRave incorporates also a possibility to interact with ROS, which will be used within the evaluation and will be discussed in 4.2.2. Unfortunately, this interaction does not use the current tool chain and so the two systems are not fitting well together. Another important aspect to mention is that there are some problems with the combination of OpenRave and some manipulators. We will discuss later on the implementation of the inverse kinematics in Section 4.2 and the evaluation Chapter 5.
Another system which deals with different manipulators but with focus on an industrial setting, is described in [28]. The task is specified via the industrial robot language and afterwards interpreted within the system. The system performs an inverse kinematics as well as an inverse dynamic calculation. Both calculations use a "setup phase" to produce equations, which are later used during the execution of a task. The calculations are performed in an analytically way to get solutions, which can be calculated fast during task the execution. During the execution of the program the systems try to perform non-jerky motions to avoid problems with

[1] http://www.ros.org/wiki/arm_navigation/Tutorials/tools/Planning%20Description%20Configuration%20Wizard
[2] http://openrave.org/

positioning errors. This system does not handle any type of environment during the motion. This is an valid assumption within an industrial setting with specified poses, because there are no uncertainties or changes in the environment and the positions could be calculated in such a way assuming collisions will not happen. This have the drawback that a general usage is limited and such systems does not perform for example well in a housekeeping tasks, because the environment changes between different planning steps within a household environment.

3.2. Solving the inverse kinematics problem

After we discussing the solutions, which tackle the whole task we will focus on solvers, which solves the inverse kinematics problem. We will present a short overview of the solving methods. There are many possible solutions but most of them do not offer any kind of easy possibility to use them within a full robot system. Also it was not possible to evaluate all possible solvers, which would be of interest to benchmark more inverse kinematics solvers to find the best of them. The solvers which are used to solve the inverse kinematics problem are discussed later on in Section 4.2 in more detail. The overview we present in this section will be categorized according to the taxonomy presented in [29]. But we will extend this taxonomy with a third class of methods to deal with Artificial Intelligence methods.

3.2.1. Analytical methods

The first class of methods comprises the analytic methods. These methods find all possible solutions for a given problem. The different solutions are introduced through the redundancy of the kinematic chain. This opens the possibility to use different solutions for the problem to choose the “best” one that fits the problem. For instance some solutions are not possible due to collisions with the environment. These methods can be subdivided into two categories: closed form methods and methods based on algebraic elimination.

Closed form solutions

Some closed form solutions are mentioned above in Section 2.5. These methods express the inverse kinematics problem as a closed form equations. This method finds a solution very fast without problems within run time that occur if an optimization method is used. Examples for such methods are [30] and [31].

Algebraic elimination

Algebraic elimination methods express the solution as a solution of a system with multiple polynomial equations. Examples for such methods are [32], [28] and [33]. Another approach is to express the solution in a closed form with the help of a system, that depends only on one joint. Such a system can be solved numerically. Examples for such methods are [17] and [18]. There are also methods which use some numerical methods to reduce the problem or to solve a part of the problem. Examples for such methods are [34], [35] and [29].

3.2.2. Numerical methods

The second class of methods uses numerical methods to determine one solution for the problem. These methods can be subdivided into four categories.

Newton-Raphson

The Newton-Raphson method uses the inverse of the Jacobian to converge to a optimal solution. The inverse of the Jacobian matrix represent the velocity matrix of the manipulator. This algorithm yields a solution of the inverse kinematics problem if a proper starting value is chosen. If the algorithm fails the procedure has to be restarted with another starting value to yield to a solution. Another problem beside selection of the starting point is that the inverse of the Jacobian is ill-conditioned near a singularity. At such a singularity the manipulator "lose" at least one degree of freedom. Thus the manipulator is additionally constraint in movements along a subset of all possible directions. This can prevent the algorithm from converging to a solution. This problem makes the method not suitable for any manipulator with less than six degrees of freedom. Another problem can occur if the manipulators have more than six degrees of freedom. This leads to the problem that the Jacobian matrix is not a squared matrix and thus cannot be inverted easily. There are many variations of this approach to solve these problems for example with the help of pseudo inverse or transposed Jacobian matrices. Some methods are described in [22], [36], [37], [38], [39], [3] and [40].

Levenberg-Marquardt, Dampled least square

To overcome the problem of a singularity the Levenberg-Marquardt and the Dampled least square methods do not use directly the inverse of the Jacobian. Instead, these methods additionally use a term within the matrix elements to avoid numerically unstable behaviour and make it possible to avoid singularity regions. The problem with these terms is that if the terms are too large, the algorithm converge slow to a solution. There are also different methods which use variations of this approach like [41] and [42].

Control theory based

Another method is to state the inverse kinematics problem as a control problem with an error corresponding to the difference of the current and the desired pose. The resulting formula uses the velocity of a joint with respect to the Jacobian matrix, end position velocity and the positioning error. This approach is used for example in [43].

Optimization-based Approach

A very general way to approach the inverse kinematics problem is to define the problem as an optimization problem, which should minimize the distance between the result of the forward kinematics, using the joint values from a possible result of the inverse kinematics problem, and the desired pose. The first two approaches within the numerical methods can also be classified as such a method. There are many different methods, which can be categorized in, this category e.g. [44], [45], [46], [4]. We will discuss this field in more detail in Section 4.3.

3.2.3. Artifice Intelligence method

The third class of inverse kinematics solvers uses different learning techniques to solve the problem. This class is divided into two categories, which differ in what is learned and how the inverse kinematics can be solved with the help of the learned information.

[3] http://www.orocos.org/kdl
[4] http://ompl.kavrakilab.org/classompl_1_1geometric_1_1GAIK.html#GAIK

Resolved Motion Rate Control

Instead of learning the inverse kinematics, which means learning the joint values corresponding to the pose, the joint velocities are learned. A solution for the problem is easier to handle even if there are different solutions for the same problem instance, which would cause the learning algorithm to fail. Through integration over the solution the inverse kinematics can be calculated. This approach is used for example in [47] and [38].

Inverse kinematics

Another approach is to learn the inverse kinematics itself and to avoid additional steps for solving the inverse kinematics. This approach is used for example in [48], [49], [50] and [51].

3.3. Solving the trajectory path planning problem

The next problem we want to solve is the trajectory path planning problem. There are several methods to solve the problem. We will divide this method into five categories and will explain them separately. Please note that we will focus in particular on the last one because the algorithms used within the tool chain are algorithms of this type. It would also be possible to use different methods depending on the context to solve the trajectory path planning problem. For example such an method was proposed in [52] which choose a solving method depending on the distance from the start to the end pose.

3.3.1. Potential Field

The first method uses a potential field, which can be constructed out of multiple potential fields to search towards the gradient to find the motion form the start- to the goal-configuration. To compute the potential field the environment is used to avoid collisions with the known environment. Thus the potential field consists of a field for attraction which points towards to goal and repulsive parts around the obstacles within the environment. Therefore it could be expensive to calculate such a potential field. Another problem of this method is that it could end up in a local minima. To escape this local minima random motions are performed. An example which uses such a method is proposed in [53].

3.3.2. Sequential Framework

Another method tries to use a hierarchical search with backtracking. This hierarchical search starts with the base joint of the manipulator and continue with the next joint of the manipulator until valid a path is found. In each hierarchical step step a path is search for the current joint which is selected. The path is searched in such a way that the path which was found for the joint above in the hierarchy are considered. If no path can be found the search makes a backtracking step within the hierarchy. One example of such an algorithm, which uses a potential field to find a path within the search step is described in [54]. There are different problems with these kind of algorithms. First the algorithms are very slow. The other important problem is that the algorithm does not always find the shortest path for the whole manipulator. Therefore we do not use this kind of approach.

3.3.3. Graph-based methods

The next group of method uses a graph to solve the problem. The nodes of the graph correspond to a joint configuration. The edges are movements of one or more joints to reach the next node. The search for a feasible path consists of the search for a path in the graph from the start to the end node. To avoid

the problem that the graph gets to large to be stored completely it will be calculated lazy to save only the information necessary to solve the problem. Examples that uses such a methods are [55] and [56]. These methods are very similar to the sampling based methods and thus we will not concentrate on the graph search methods. Beside the similarity it is also important to notice that the performance of the graph-based methods have to be improved to make it possible to compete with the sampling based methods.

3.3.4. Optimization methods

The next category contains methods which use optimization technicians to get a feasible path. At the beginning the methods use at least one guess of a path within the joint space. This path consists of different configuration with a specified distance between each other. Afterwards the path is refined until a solution is found. If more than one path is used all the paths are refined to find a solution. The authors of [57] use a genetic algorithm to find a solution with a fitness function that indicates if the path is in collision and if the goal is reached. The approach used in [58] calculates a gradient corresponding to the environment and the smoothness of the path to refine the guess to find a path. [59] uses a stochastic method to find a possibility to improve the path and find a solution. All these methods have in common that they are not as fast as sampling based methods.

3.3.5. Sampling-based methods

The last category, which is also the category we use to find a solution for the trajectory path planning problem, contains sampling-based methods. These methods sample the configuration space in a certain manner and afterwards connect them to find a path. As already discussed above these methods are very similar to the graph based methods because the sampling points and their connections could also be viewed as a graph. There are many methods which are part of this category. We will briefly point out some works and later on focus on algorithms which are used (partly) later on within the evaluation. The first example is RRT which connects the samples to form a tree [26], [60]. A similar approach like RRT is proposed in [61]. It uses random sub-goals which should be connected with the help of a modified A* algorithm. The Ariadne's Clew Algorithm which was proposed in [62] try to avoid places already visited with the help of landmarks, and try to find a connection from this landmark to the goal place. Other methods use try to place point near or along the boarder [63],[64], [65], [66]. Or change the search direction through reflection on the boarder of an object if a collision is detected [67]. Also it is possible to use not every sampled point instead use only point with special properties. Such an example is the proposed in [68]. The points are only of interest they act as a guard for a region or connect such guarding points. Another method to use only a sub-set of the sampled points was proposed in [69]. Which use only points which have at least one neighbor which is in collision. Another way to reduce the collision is to choose the sampling point in a cleaver way as it was proposed in [70]. The idea is to use sample points and move them to medial axis to avoid collisions. The authors in [71] proposed a similar approach which try to use the medial axis in Cartesian space. Also how fine grade the collision checks are preformed along a direct connection can be used for improvement. The authors in [72] assign a propability to each edge deepening how fine grade the check of the edge performed. A path is only found if every connection along the path is completely check and thus has got a probability of 1. The methods have in common that they find very fast a solution. This is the reason why we focus only on the methods within this category. We will now have a closer look at three different methods. Each of them contain interesting parts, which are used within the methods within the tool chain.

PRM

PRM [24], [73], [74], [75], [76] samples the configuration space in a random way. The different sample points correspond to a configuration of the manipulator with the corresponding joint values. Each of these configurations can be checked for a collision with the environment. If a collision occurs the corresponding

sample point will be removed from the samples. The sampling could be executed every time a new query appears or only if the environment changes. It is also possible to cache these environments if for example there are only few different configurations of the environment. This can happen for example within an industrial environment. After the sampling step is finished different path planning requests can be processed. To process this request the samples are connected in such a way that a local planner could find a connection and the connection does not yield to a circle within the resulting graph. Within this graph a shortest path search is processed. If no path can be found it is possible to re-sample the space or to add sample points to the current samples.
One big problem of this algorithm is the huge amount of collision checks for the sampling points. The check becomes more costly if the environment is very cluttered. This results in a very high execution time of the collision checks and the algorithm itself. On the other hand without enough sampling points it is not possible to find a path within the graph.
The development leads to a lazy evaluation of the sample points. This lazy evaluation, which was discussed in [77], [78] and [79], processes the collision check of the different sampling points in a lazy manner. This means that only sample points, which occur within the shortest path are checked. If the sampling point is not valid (the configuration is in collision with the environment) it will be removed. All the connections to these sample points will be removed too and the graph can be rebuilt without these sample points. It is sufficient to check if sample points, which are in the neighborhood of the removed sampling points are now close enough to be connected. Afterwards a shortest path search is processed and so on.

EST

Another way to find a path within the configuration space is presented in [25]. The idea is that a path is only possible if the sample point along the path can “see” the next sample point along the path. Two sample points see each other if a straight line can be drawn between these two points. This also means that the line does not intersect with an object. This check is performed on the objects, which have to be transformed to the configuration space and therefore change their shape dramatically. Such a transformation would also be complicated to achieved. To check this intersection the line is sampled and each sampled point has to be checked if it is in collision with the environment.
To find a path the algorithm uses two growing trees. One tree with a root at the start sample point and one with the root at the end sample point. The start sampling point is the starting configuration and the end sample point is the goal configuration to reach. Within each iteration of the algorithm the two trees grow and are checked to see if they are able to “see” each other. The two trees “see” each other if a sample point of the first tree “sees” a sample point out of the second tree. This can be verified very simply by checking if two of the possible combinations “see” each other. It would also be possible to cache the results from one to another iteration of this step. To extend a tree one sample point of the tree is chosen. In the neighborhood of these sample points new uniform distributed sample points are generated. With a certain probability these sample points are dropped. If a new sample point is not dropped it is checked if it is a valid sample point. If the new sample point is not valid it will be dropped. Afterwards the new sample is checked to see if it “sees” the old sample point of the tree. If the sample points see each other the sample points will be connected.
One drawback of the algorithm is to check if two sample points ”see” each other. This can be improved if only sample points are considered which are close to each other before the check if performed.

KPIECE

The last path planning algorithm we want to discuss is the KPIECE algorithm [21]. The idea behind this algorithm is to avoid sampling regions, which are not of interest. Sample points are not of interest if they don’t lead to a solution. As discussed above, one drawback of the PRM algorithm is that there is a huge amount of sample points, which have to be evaluated. The algorithm was improved by checking the sample points lazy but it does not solve the drawback that the algorithm samples many points within the configuration space in regions, which are not interesting. EST has got a similar problem. If many iterations

have to be made within the algorithm both the tree and the number of sample points grow. If the trees does not grow towards each other many irrelevant sample points are considered. With EST the problem can be solved by introducing a weight for each node, which changes the probability to be chosen to be the next node to grow the tree. This forces the trees to grow towards each other.
KPIECE uses another idea to reduce the number of samples. The idea is to use a multi-level grid. Each grid cell (A), has got one cell which is one level above A (parent cell) and k grid cells one level below A (child cells). The algorithm samples the configuration space randomly but it prefers cells which have no neighbor cells. The algorithm also tries to expand the grid if it possible to add a neighbor cell. This enforces the algorithm to explore as much space as possible. If a cell is selected, the sample point within the cell is chosen similarly to the method before, with the difference that the child cells are used to find the position of the sample point. This also forces the algorithm to sample the space uniform. This is necessary to make it possible that the algorithm is probabilistically complete. The path itself is established through a growing tree, which connects the sampled nodes and tries to find a path. The main difference to EST is the way the sample points are generated.

3.4. Solving the path execution planning problem

There are several methods which provide solutions for the path execution planning problem. For example the method proposed in [28] uses different function (ramps or sin waves) to interpolated between the different configurations. Other methods use splines to interpolate the points and perform afterwards a optimization to find a solution. Such methods can be found in [80] and [81]. The Authors in [82] use polinomials instate of splines to interpolate between the points. Many methods use a projection into a space which is a plain containing the path in one direction and the velocity along the path in another direction. There are many proposals to solve the resulting planing problem some examples are shown in [83], [84], [85], [86].
Each of these methods uses a set of constraints which are specified. The set contains constraints on the possible values for a joint, the velocity of the joints as well as the maximum and minimum acceleration of the joint. We will now focus on some of these methods, which contains the more interesting parts. These parts are later used in the algorithms, used in the tool chain. We only explain the idea behind each of them and provide not a detailed explanation of the different equations which are used within each of the algorithms.

3.4.1. Splines and polynomials

One method to find a trajectory execution is to use spline functions as proposed in [80] or [81]. The splines use the given joint configurations and additionally the start and the end velocity is set to zero. The splines try to interpolate the given joint configurations. The splines are continuous and differentiable. This is an important property because if the function would not be differentiable the velocity at this position would not be continuous and would yield to a violation of the velocity constraints. An important issue is that the splines need to know the time step between the joint configurations. To calculate this time step in such a way that the result is nearly time optimal the problem is formulated as an optimization problem. The optimization problem is to find the time steps of the different joint configurations in such a way that the overall time is minimal and the splines can be applied without violating any constraints. The methods differ in the calculation of the splines and in solving the optimization problem. It is also possible to use the spline interpolation not in correspondence to the joint values themselves but for example to correspond to joint velocities or accelerations. Another possibility is to use polynomial-functions instead of splines to interpolate the function. It is important that the resulting trajectory uses a function, which is differentiable as we stated above. An example which uses a polynomial instead of a spline is given in [82].

3.4.2. Linear segments with parabolic blend functions

Another method is to use a projected representation of the problem as it was proposed in [87], [87], [88]. The projection of the problem only works if the trajectory path is given in such a way that it can be differentiated. In general the path which is produced through a trajectory path planing algorithm does not fulfill the requirement that the path can be differentiated. It only offers joint configurations, which can be interpolated in a linear way. To overcome this problem it is possible to blend the different line segments with parabolic functions. This results in a differentiable function, which can be used later for the projection. Such a method was already proposed in [89]. The projection itself projects the problem on a different problem representation to find the velocities along the given path. The different limits have to be transformed into constraints within this projection. To find a solution to this problem it is possible to use the algorithm proposed in [90] and [91]. The algorithm uses a representation of this problem within a two dimensional space. The space represents the way and the corresponding velocity. It is important to note that the velocity corresponds to the joint velocities but is not the same as the joint velocities. Within this space the task is to move from the start pose (the position at the beginning of the projected path and with a velocity equal to zero) to the end pose (the position at the end of the projected path and with zero velocity). It is also necessary to move as fast as possible, which is achieved if the maximum possible velocity is used. The algorithm itself use a switching points. The switching point is a point, were the maximum velocity of a joint changes from one joint to another one. It is important to note that the manipulator should move as fast as possible, which results in the observation that at least one joint should move with maximum velocity if it is possible. The algorithm works as follows:

1. Integrate with the maximum possible acceleration until a constraint is violated.
2. If the violation of a constraint is caused by a velocity constraint move along the constraint until an integration is possible or another constraint is hit.
3. If the violation of a constraint is caused by an acceleration constraint calculate the switching point and integrate backwards with the minimum acceleration until the previously calculated movement is hit. The path now consists of the path before this intersection point and the path taken from the backward integration. Continue with the forward integration above.

The Algorithm will try to move along the resulting path using the maximum possible increase in velocity and the slowest brake action. This results in a path which uses a very high velocity and therefore is very fast. It is important to note that the integration, which is performed numerically can lead to some problems, which can be resolved through some additional checks.

Chapter 4

Solution concept

In the previous sections we have defined the problems which have to solved in each part of the tool chain as well as discussed some methods to solve this problems. Within this chapter we will discuss in more detail how the ROS Arm Navigation Stack [92] is used implement the tool chain. Moreover we will provide a full explanation of each step within the tool chain.

4.1. The tool chain

In Section 2.4 we briefly imposed the structure of the tool chain. Each of the steps within the tool chain and their definitions are given in Chapter 2. We will now have a closer look at the implementation of the tool chain. The concept of the tool chain is shown in Figure 4.1. Each subproblem receives the input from the previous subproblem. By solving the subproblem the necessary input for the the next subproblem is generated. This concatenation is also a reason why it is important that each of these steps is performed fast as well as with a high success rate in order to prevent the entire tool chain to fail.

1. The first subproblem is the inverse kinematics problem (for the definition see Section 2.5). The problem is to determine which joint values correspond to the resulting pose of the manipulator. The result also needs to fulfill the joint limits and must not be in collision with the environment. After solving this subproblem we get goal values for the joints, which should be reached from the current joint values to reach the resulting pose. It is important to notice that in general the resulting pose maybe could not be reachable from the current position. Also it is not always possible to use a straight movement (within the joint space) of the manipulator to reach the goal. Thus we need the next step to reach the calculated joint values from the current joint values.
2. The second subproblem is the trajectory path planning (for the definition see Section 2.6). The problem is to find a possible path within the configuration space from the current joint configuration to the desired joint configuration, which was calculated in the previous step. The resulting path is constrained to be minimal. It considers the change of the joint values and also does not violate the joint limits. The path also have to avoid collisions with the environment. The result of this subproblem is a list of joint values which should be reached one after the other to reach the final joint position. We only consider an environment which is stationary during planning and execution time. It would be possible to avoid some collisions with moving objects after the path was calculated as it was proposed in [19]. This was also discussed above. But we will not consider this possibility within this thesis. The result is a movement with the manipulator from the current configuration to a configuration, which represents the desired pose. The problem we have to deal with next is to get the joint velocities to pass one joint configuration after the other to get a fast and accurate movement with the manipulator.

3. The third subproblem is to follow the path determined in the previous subproblem. This is called trajectory path execution problem (for the definition see Section 2.7). The problem is to find a path execution which is as fast as possible for a given path. The different mechanical limits of the manipulator have to be considered in order to be able to execute the trajectory without problems. Also all different limits of the joint such that the limits of the joint velocity and acceleration has to be considered. The result of this subproblem is a list of time slots and the corresponding joint velocities for all joint. This list could be used to move the manipulator from the current configuration to a final configuration.

4. The last subproblem is to execute the given trajectory. The problem is not a planning problem. It is just a transformation of the output from the previous subproblem in order to make an execution possible. This part strongly depends on the manipulator and its controller. Thus we will not explain how to solve this problem, because this problem is solved via a driver, which is manipulator dependent and is usually provided by the manufacturer of the manipulator.

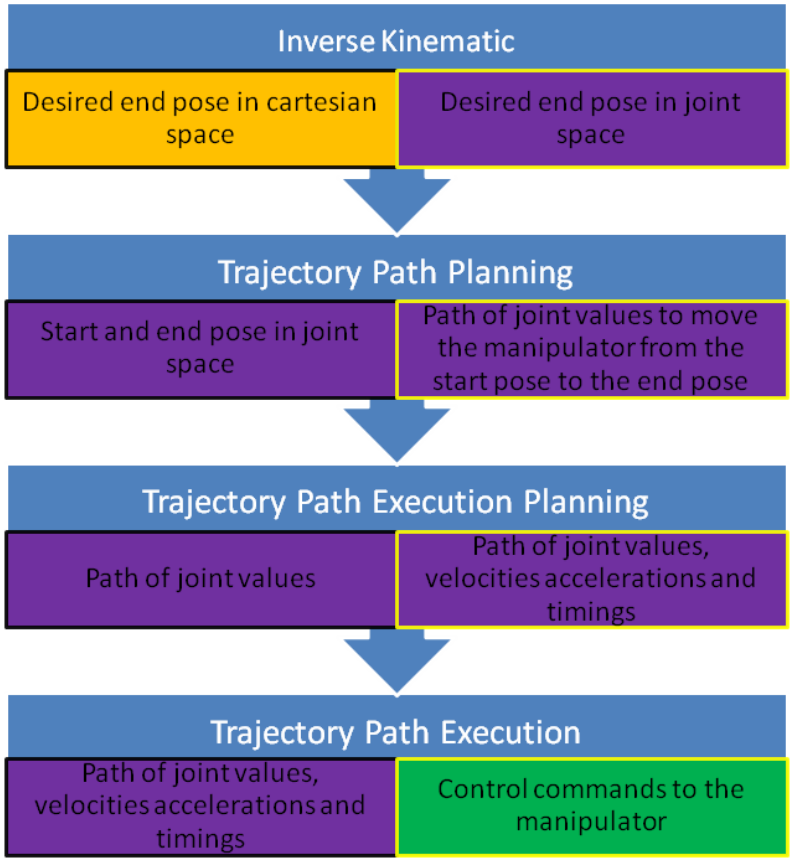

Figure 4.1.: The different steps of the tool chain. The black framed parts represent the input for each of the steps. The yellow framed parts represent the output for each of the steps. Within the orange box the values are specified in Cartesian space. Within the purple boxes the values are specified in joint space. The green box contain values which are specified in a manipulator depending representation.

It should be noted that the second and the third subproblem can be addressed in an integrated way as a kinodynamic motion planning problem, which can be solved for example with [21] or [22]. We treat each of the subproblems separately as it was proposed in [19]. This decomposition has the huge advantage that the problem becomes easier to handle and the third subproblem can be performed very fast, even for robots with many degrees of freedom as it was for example proposed in [82].

4.2. Solving the inverse kinematics problem

We briefly explained which different methods can be used to solve the inverse kinematics problem in Section 3.2. We will explain the methods which are used within the tool chain in more detail in this chapter. We use different methods to tackle the problem of the inverse kinematics. The simple reason for using different methods is that not every manipulator can use every method or the usage of a method results in drawbacks (for example a long execution time). There are also other tools which use multiple methods to solve the inverse kinematics problem, as for example [28]. The inverse kinematic solver are tried one after each other to find a solution, if the previous one does not find a solution.

4.2.1. KDL

KDL [1] is one part of the Open Robot Control Software (OROCOS) project [93]. The OROCOS project offers system independent libraries to build up robotic systems. KDL is a library that offers data types and solvers for problems. We focus on two solvers which are used to solve the inverse kinematics problem and the forward kinematics problem.

Forward kinematics problem To solve the problem it is very suggests itself to use the forward kinematics chain recursive solver. It is sufficient to apply one transformation after the other along the way from the base to the end effector of the manipulator. To get the transformations in a correct order and to avoid problems, which may occur if the manipulator is not only one straight kinematics chain (a robot with multiple manipulators could yield to such a case) the transformation is build recursively starting from the end effector. The outline of the algorithm is shown in Listing 4.1

```
forwardkinematicsChainRecursive( kinematicsTransformation, currentLink, baseLink)
  if(currentLink == baseLink)
    return
  parentLink = currentLink.getParent()
  kinematicsTransformation.addToFront( TransformationsFromTo( parentLink, currentLink))
  return forwardkinematicsChainRecursive( kinematicsTransformation, parenLink, baseLink)
```

Listing 4.1: Outline of the KDL forward kinematics chain recursive solver

As stated above, it is possible that the manipulator is represented in a tree structure. This makes clear that the chain from the end effector to the base is uniquely defined. The complete transformation consists of sub-transformations. Each sub-transformation is a transformation of adjoining links. Thus the complete transformation is feasible. At the end of the algorithm a transformation, which starts at the base and ends at the end effector and is feasible, is produced. This transformation can be used to calculate the forward kinematics (a more detailed description is given in [94]).

Inverse kinematics problem To solve the inverse kinematics problem we use a inverse kinematics solver with Newton-Raphson iterations. The algorithm needs two other algorithms to solve problems, which is used during the iterations at the algorithm. The first algorithm is a forward kinematics solver, as is described above. The second one is an algorithm to calculate the inverse Jacobian matrix. The algorithm also needs initial joint values. As discussed above, in Section 2.5 this starting point has an impact on the convergence of the algorithm. To calculate the inverse of the Jacobian matrix it is possible to use a functionality implemented in KDL, which calculates the pseudo inverse of the Jacobian with the help of a singular value decomposition. It is also possible to use a damped least square method to calculate the inverse of the Jacobian. This has the same advantages to handle singular values but also the drawbacks of a slower convergence. The outline of the algorithm for calculating the inverse kinematics is shown in Listing 4.2

[1] http://www.orocos.org/kdl

```
inversekinematicsSolverNewtonRaphson( numberOfTotalRounds, initalJointValues,
    endPosition)
currentJointValues = initalJointValues
while(numberOfRounds < numberOfTotalRounds)
  currentPosition = forwardKinmaticSolver( currentJointValues)
  currentVelocity = calculateVelocity( endPosition, currentPosition)
  if( currentVelocity < epsilon )
    return currentJointValues
  jacobianInverse = getJacobianInverse()
  inverseJointVelocity = calcuateInverseJointVelocity( jacobianInvere, currentVelocity)
  currentJointValues = currentJointValues + inverseJointVelocity
  checkAndHandleJointLimits(currentVelocity)
```

Listing 4.2: Outline of the KDL inverse kinematics solver with Newton Raphson iteration

We will now go into detail and explain the different steps of the algorithm. The algorithm iterates until a solution is found or a maximum number of rounds is reached. First, we calculate the forward kinematics to get the current pose, which is determined by the current joint positions. Then the current velocity is computed. For this the algorithm calculate the difference of the current and the desired pose. This distance is afterwards converted into a velocity be amusing that the distance can be traveled in one time step. This position difference is represented by a six dimension vector (three differences within the coordinates and three rotations differences). If the current velocity is below a specified limit, a solution is found and the iterations end. Otherwise the algorithm calculate the inverse of the Jacobian (with the help of the given function). Then the algorithm calculate the inverse joint velocity by multiplying the current velocity and the inverse of the Jacobian matrix. Now the current joint values are updated. After checking the limits the algorithm continue with another iteration. A more detailed description is given for example in [94]. As stated above we do not find a solution if we have a bad initial guess or the inverse Jacobian is not well defined. This does not happen very often if we use a damped least square method. The impact of the inverse Jacobian is caused by the update step. Please note that the algorithm is a simple numerical optimization algorithm with the Newton-Raphson iteration.

4.2.2. OpenRave inverse kinematics

OpenRave [2] is a framework to handle manipulation tasks (e.g. pick and place tasks). We will focus here on the inverse kinematics module [3]. This module provides a generator for inverse kinematics solvers, which is called Ikfast [33]. The solver also provides a forward kinematics solution. We will describe both in the following paragraph.

Forward kinematics problem One big difference to the forward kinematics solver of KDL is the time it takes to get the equation for calculating the forward kinematics. This equation is calculated within KDL at run time. It uses the current manipulator, the given base and end effector frame to generate the equations. Ikfast on the other hand uses the robot description and generates a file with the forward kinematics equation. This step is can be called “setup phase” and is performed for a new manipulator. The advantage is that after the files are generated the forward kinematics has only to perform the mathematical operations (trigonometric functions, multiplications and additions) to get a forward kinematics solution. This does not lead to a big time difference but it shows that the philosophy behind Ikfast and KDL is completely different.

Inverse kinematics problem The inverse kinematics problem can be solved via a solver, which is generated using analytic methods. The solver is calculated during the “setup phase” and stored in a file. After the setup the solver solves the inverse kinematics problem using the equations provided by the “setup phase”.

[2] http://openrave.org/
[3] http://openrave.org/docs/latest_stable/openravepy/databases.inversekinematicss/

This results in a very high performing inverse kinematics solver. Another important aspect is that the solver has no problem with singularities and can provide multiple possible solutions within one solving call. Unfortunately, some solutions cannot be found by the solver (we will have a closer look at the performance of the solver in Chapter 5). To generate an inverse kinematics solver with Ikfast the solver calculates constraints out of the kinematics equations. A kinematics equation consists of two kinds of transformation matrices. The first kind of matrices are static transformation matrices, which are the transformation along the links. The second kind of matrices are transformation matrices which change with there values according to the specified joint values. These equations can be rewritten by inversion of matrices. This yields to several equations, which can be reformulated as polynomial equations with a trigonometric and a general side condition. The result is a system with $2n$ unknown variables for a manipulator with n revolution joints. The system is solved using these equations. This is achieved by a search for the simplest and most unambiguous solution. To get this solution and solve the system it is important to notice that the system can be solved in different ways. But all these possible solutions have drawbacks, which have to be considered. All these requirements for a simple solution, which deals with the drawbacks, lead to a rating for a solution. This rating consists of a so called *solution complexity* and a *numerical complexity* [33]. The solution complexity measures the amount of introduced solutions due to these solving steps (e.g. an inverse of a sinus function results in two solutions). This measurement is used to avoid solutions, which introduce solutions that cause a conflict with the kinematics. To use solving steps, which offer different solutions it is sometimes necessary to get all solutions and thus it will not be avoided due to a bad ranking. The numerical complexity is used to differentiate between the equations, which have got the same solution complexity. The measurement consists of two parts. First the measurement penalizes solving steps, which use a division. This is used because a division by zero leads to a degenerate solution. The second part of the measurement is the number of operations. This should enforce the usage of solving steps, which are faster than other solving steps. Using these two measurements a variable is chosen. The solver chooses an equation using the knowledge of variables, which are already solved. In order to get a solution for the variable by sorting them. The following order is used to sort the variables:

1. Linear Independent equations. A low degree of the polynomial is preferred
2. Equations of the type $a * cos(x) + b * sin(x) = c$
3. Equations of the type $(cos(x))^2 = a$
4. Closed form solutions

The solver chooses the equation which is at the beginning of the order. If the solver finds an equation with a degenerate case (e.g. divided by zero) the solver creates a new branch for a solution, which penalizes the equation which causes the degenerate case. If the equation with a degenerate case can be calculated within the "setup" phase (for example $cos(x) = 0$) it results in the different branches, which correspond to the above degenerate case (in the example above $x = -\pi/2, x = \pi/2$). If such a degenerate case cannot be evaluate at the "setup" phase the equation is penalized more. There are also other tools available. But unfortunately not integrated in ROS, which also produces analytic solutions with methods like the described one within this section. For example [28] describe such a method.
To use the solver which is generated from ikFast within the tool chain, a wrapper was created which transforms the input into the correct input to pass it to the solver and to extract the right information from the solver. The wrapper also tries to find a solution which is near the desired pose. This increases the runtime of the wrapper as well as the success rate.

4.3. Stochastic inverse kinematics

With the above mention solver we run into the problem that not always a solution was found. KDL for example has problems with singularities and therefore can not always find a solution of the inverse kinematics problem. With OpenRave we have a very fast and exact inverse kinematics solver. The problem with this solver is that for some manipulators ikFast is not able to find a solution for reachable poses. For both solver (KDL and OpenRave) a solution can not be found and thus the complete tool chain fail. We

will have a closer look at the performance of the different solvers in Chapter 5. To make it possible to find an inverse kinematics solution for any kind of manipulator independent of the geometry we created the stochastic inverse kinematics solver. We are also able to deal with any degree of freedom in contrast to OpenRave which can only deal with with manipulators with up to six degrees of freedom.
This type of solver doesn't offer an opportunity to solve the forward kinematics problem. But it is important that there exists a forward kinematics solver which can be used in the inverse kinematics solver. To solve the forward kinematics problem the forward kinematic solver from KDL is used. It would also be possible to use other forward kinematics solvers, like the solver from OpenRave.

Inverse kinematics problem First we motivate the main idea behind the stochastic inverse kinematics. The idea is rather simple. We treat the problem of the inverse kinematics as a general optimization problem. This is similar to KDL which uses the Newton-Raphson iteration. But this is a local optimization algorithm and thus can get stuck in a local minima. The algorithm will minimize the distance from a possible pose to the desired pose. In case of the inverse kinematics solver the distance along the coordinates in Cartesian space and the orientation difference have to be minimized. It is important to notice that the distance within the orientation is determined by using quaternions [95]. An advantage of using quaternions instead of Euler angles is that the quaternions do not suffer from the so called Gimbal lock (the Gimbal lock causes the loss of one degree of freedom if for example the pitch and the yaw become aligned). With Equation 4.1 we calculate the position difference. p_1 and p_2 are both positions in $\mathbb{R}^3$. And with Equation 4.2 we calculate the orientation difference. q_1 and q_2 are both orientations in $SO(3)$[4]. With the help of the forward kinematic function $FK(v_{j_1},\ldots,v_{j_m})$ which takes a value for each joint (v_{j_i}) the objective function can be specified with Equation 4.3. For a more formally definition we refer the reader to [6].

$$posDiff(p_1,p_2) = \|p_1 - p_2\|^2 \tag{4.1}$$

$$orDiff(q_1,q_2) = |1-(q_1 \cdot q_2)^2| \tag{4.2}$$

$$F(v_{j_1},\ldots,v_{j_{i-1}}) = \alpha * posDiff(FK(v_{j_1},\ldots,v_{j_{i-1}}),\ P) + \beta * orDiff(FK(v_{j_1},\ldots,v_{j_{i-1}}),\ O) \tag{4.3}$$

Beside the minimization of the distance the algorithm also have to consider different constraints. The constraints in the case of the inverse kinematics are the joint limits. It would also be also be possible to add other constraints in order to avoid particular combinations of joint values or to find the closest feasible position to the desired pose.
The optimization problem stated above can be solved with many methods. There are different examples to solve the inverse kinematics with a stochastic optimization algorithm [45]. We will focus on a global optimization method to avoid the problem of local minima. To find such a global minimum we use the *Differential Evolution algorithm* [96]. This optimization technique converges fast to a global optimum. The idea behind *Differential Evolution* is to find the optimum via combination of individual solutions, which is similar to any genetic or evolutionary optimization algorithm.
We will now have a closer look at the proposed algorithm. First we want to explain one important term: the individual. An individual is one possible solution for the problem which could be combined with other individuals and/or mutated to find the global minimum. To represent such an individual within the algorithm it consists of two parts. The first part consists of the values to be used within the forward problem. In the case of the inverse kinematics the values are the joint values. The second part is the function value which is calculated using the objective function. In the case of the inverse kinematics calculation the value of the objective function is the distance to the desired pose. To use multiple possible solutions a population is used which is a set of individuals.
The algorithm combines individuals from the population to generate better individuals. The outline of the algorithm is shown in Listing 1

At the beginning the algorithm generates an initial population. There are several methods to generate such a population. A common method is to uniform distribute of the individuals within the whole space of

[4] $SO(3)$ is the space of all rotations in a three dimensional world

Algorithm 1: *findMinimum*

input : F ... Objective function,
C ... set of constraint,
maxIt ... maximum iterations of the algorithm,
CR ... cross over ratio,
N ... size of the *population*
output: a set of possible solutions CP

```
1  CP = generateValidPositions(N, C)
2  evaluate(CP, F)
3  i = 0
4  while (i < maxIt) ∧ ¬convergenceCriterialsMet(CP) do
5  |   donors = mutate(CP)
6  |   trials = recombine(CP, donors, CR)
7  |   trials = clamp(trials, C)
8  |   evaluate(trials, F)
9  |   CP = select(CP, trials)
10 |   i = i + 1
11 end
12 return CP
```

possible solutions. Then the value of the objective function is evaluated for each individual. If the maximum number of iterations (*maxIt*) is reached or a convergence criteria is met (*convergenceCriterialsMet*) the algorithm terminates. or if returns true. This can be achieved if a specified percentage (within this implementation 10 percent) of individual are very close (have a objective value of less than $10^{-}6$ to the solution (similar to the stopping criteria within KDL). Another possibility is to stop if the population begins to stagnate. This means for example that the values of the objective function of the best k individuals does not make any change significantly. Within the loop there are four steps, which are processed one after the another. First the current population is mutated to get a new population. There are different mutation strategies which can be used to generate a new population. The strategy we use chooses two random individuals and the best one and combines them. this operation is shown Listing 2. *f1* and *f2* are weights, which are parameters of this strategy. The combination of individuals iterate until a new population with the given size N is created.

Algorithm 2: *mutate*

input : P ... Current *population*,
N ... size of the *population*,
f1 ... first combinatoric weight,
f2 ... second combinatoric weight
output: a new *population donors*

```
1 for i = 1 : N do
2 |   r1 = drawRandomIndividual(CP)
3 |   r2 = drawRandomIndividual(CP)
4 |   best = getBestIndividual(CP)
5 |   newIndividual = f1 ∗ (r1 − r2) + f2 ∗ (best − r1)
6 |   donors = donors ∪ newIndividual
7 end
8 return donors
```

To make a simple example lets assume the population P consists of three individuals in a two dimensional world. The objective values as well as the position of the individuals can be seen in Table 4.1. We use now

the algorithm which is shown in 2 to calculate two new individuals ($N = 2$) with $f1 = 0.2$ and $f2 = 0.8$. For the first new individual the first random individual r_1 is individual 1 and the second random individual r_2 is individual 2. For the second new individual the first random individual r_1 is individual 2 and the second random individual r_2 is individual 1. The resulting individuals which are calculated are shown in Table 4.2. As it can be seen the distance from the new individuals to the best individual is smaller than the distance from individual 1 and individual 2.

Individual	Objective function	position 1	position 2	distance to best individual
1	0.5	1.5	-1.6	2.1932
2	0.3	1.0	2.0	2.0224
3	0.01	0.0	0.0	0.0

Table 4.1.: Example Population

Individual	position 1	position 2	distance to best individual
1	-0.96	0.56	1.1114
2	-0.48	-0.88	1.0024

Table 4.2.: Resulting individual

The existence of the possibility to adjust the weight during the algorithm is also noteworthy. But we will not use these possibility within the implementation.
After a new population is created the current and the new population are recombined. This operation is depicted in Listing 3. The algorithm iterates over the current population and chose the individual of the donor population at the same index. These two individuals are combined through of the cross over operation. The cross over operation first randomly selects on index which will be changed to a new value. This guaranties that at least one value changes. Afterwards for every joint value a decision is taken to use the value of the donor or the value of the current individual. If the joint value is at the specified index the value of the donor is chosen. Otherwise a random value between 0 and 1 is drawn and if this value is smaller or equal than the cross over ration the value is also taken form the donor. In any other case the value is taken from the current individual.
Using this algorithm at least one value of an individual changes, which for the inverse kinematics means at least one joint value changes. The population, which is generated through this procedure is the new population, which may have better individuals.

For an example we use the current population in Table 4.1, and as a donor population we use Table 4.2. In our example is $k = 2$, $N = 2$ and $CR = 0.8$. For the first new individual we draw the index 1. Thus the first joint value of the first new individual we use the first joint value of the first donor individual. For the second joint value we draw the number 0.5 and thus we use the second joint value of the first donor individual. For the second new individual we draw the index 1. Thus the first joint value of the second new individual we use the first joint value of the second donor individual. For the second joint value we draw the number 0.9 and thus we use the second joint value of the second current individual. The resulting population can be seen in Table 4.3.

Individual	position 1	position 2
1	-0.96	0.56
2	-0.48	1.0024

Table 4.3.: Resulting individual

Algorithm 3: *recombine*

input : P ... Current *population*,
donors ... donor *population*,
CR ... Cross over ratio defining how individuals are combined,
k ... number of degrees of freedom,
N ... size of the *population*
output: a new *population trials*

```
for i = 1 : N do
    index = drawUniformFrom([1 ... k])
    for j = 1 : k do
        if (j = index) ∨ (drawUniformFrom([0 ... 1]) ≤ CR) then
            individualToCombine = donors[i]
        else
            individualToCombine = P[i]
        end
        newIndividual[j] = individualToCombine[j]
    end
    trials = trials ∪ newIndividual
end
return trials
```

To check if new individuals are better than the old one the objective function is evaluated for every individual within the population. Then, the selection inside the new population and the current population is processed. There are different ways to perform the selection process. We use a very simple one which is depicted in Listing 4 The selection operation checks every individual from one population at position

Algorithm 4: *select*

input : P ... Current *population*,
trials ... trial *population*,
N ... size of the *population*
output: a new *population* P'

```
for i = 1 : N do
    if (objective(P[i]) < objective(trials[i]) then
        newIndividual = P[i]
    else
        newIndividual = trials[i]
    end
    P' = P' ∪ newIndividual
end
return P'
```

i with the individual from the other population at position i. The individual that has a lower objective function value is used within the next population.
The optimization algorithm presented in Listing 1 is very simple and can be improved by applying changes to the different used strategies. For instance using other strategies for mutation, selection, initialization. However, we will not discuss all these possible improvements since we use this very simple algorithm to solve the inverse kinematics, because it already shows a high success rate.
There is one last thing which worth mentioning. The optimization algorithm is embodied in algorithm 5. This is done to deal with found solution which are in collision. If the optimization algorithm find a solution which is in collision the optimization algorithm will be called again. After some iterations of recalling the

optimization algorithm with the initial number of individuals and iterations the number of individuals or the number of iterations is increased by a certain factor to increase the chance that a solution is found. This comes with the price that the run time of the algorithm is increased.

Algorithm 5: *NaiveStoachsticInverseKinematicSolver*

input : FK ... forward kinematic function,
TOT ... time out time,
$maxItI$ ... increment of maximum iterations,
NI ... increment of the *population* size,
τ ... threshold to start increment,
$maxIt$ ... maximum iterations of the algorithm,
CR ... cross over ration,
N ... size of the *population*
output: a set of possible solutions CP

```
1  sC = 0
2  while (currentTime() < TOT) do
3  |  PS = findMinimum(FK, C1, maxIt, CR, N)
4  |  foreach S ∈ PS do
5  |  |  if (checkConstraint(S, C2)) then
6  |  |  |  return S
7  |  |  end
8  |  end
9  |  if (sC > τ) then
10 |  |  if (sC%2) then
11 |  |  |  maxIt = maxIt * maxItI
12 |  |  else
13 |  |  |  N = N * NI
14 |  |  end
15 |  end
16 |  sC = sC + 1
17 end
```

4.4. Solving the path planning problem

After we discussed in general different algorithms to solve the path trajectory planning problem in Section 3.3 we want to introduce in more detail the solvers which are used at this step of the tool chain. All of these methods are part of the Open Motion Planning Library (OMPL) [92]. OMPL [5] is a library, which contains several sampling-based motion planning algorithms. The library does not contain any method to check for collisions with the environment itself. Such a check has to be provided by applying a different algorithm. This is very useful since this makes it possible to use one part of the program to perform collision checks. This part can be used to perform different collision checks within different algorithms. For example the collision check can be used within the trajectory planning but also within the inverse kinematics.
We will now explain how the two different methods work. These methods use parts of the previous discussed algorithms and combine them in different ways. There are a lot more possibilities within the tool chain. However we will focus on these two methods, which will also be used within the evaluation in Chapter 5.

[5] http://ompl.kavrakilab.org/

4.4.1. SBL

Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking (SBL) [6] is the first algorithm, which is used. The algorithm uses two growing trees to discover a path. The strategy for the trees expand is similar to the EST and uses a grid, similar as to the grid in KPICE, to enforce the exploration of regions, which are sampled sparsely. This should avoid the problem of samples, which are less interesting for the solution. Another method to speed up the process is the usage of a lazy evaluation schema. The combination of these methods may result in a decrease of the amount of sampling points, which are evaluated and are not used later within the solution.

4.4.2. LBKPIECE

The other algorithm, which is used is LBKPICE [7]. The algorithm uses a modified version of the KPICE algorithm. It uses two growing trees within the grid to explore the space. The algorithm does not use multiple levels as described in KPICE. This results in an exploration of the trees into regions, which are at the border of the explored space. Another important aspect of the algorithm is that it evaluates the samples in a lazy way. This results in a small number of samples, which have to be evaluated. This lead to a fast algorithm to solve the path planning problem.

4.5. Solving the path execution planning problem

The tool chain uses the so called ROS-Trajectory-Filters to perform the path execution planning. The filters can be applied one after another. For instance first a filter is used which calculated the timings without considering all limits. The next step is to apply another filter which considers these limits. But this would take more time if there would be no pre-processing step. This chain of filters can also be used if the controller of the manipulator has to use a certain trajectory time interval or other additional constraints, which have to be applied. For instance this is used within the evaluations with the Katana manipulator. The reason is that the maximum number of trajectory points which can be processed through the driver is limited to 16. Thus the number trajectory points have to be reduced in advance.

4.5.1. Splines

One of the trajectory planners uses cubic splines to generate the trajectory. The velocities between the way points is generated by constraining a continuity of the joint acceleration. The start- and end-velocity is set to zero. Then the spline is calculated by a linear system of equations. For each of the trajectory points a time stamp must be assigned. To calculated this time stamps the algorithm chooses the time stamps as fast as possible after each other. If some joint limit (the velocity or acceleration) is violated between two time stamps the interval is stretched.

4.5.2. Linear segments with parabolic blend functions

It is also possible to use the algorithm discussed in [90] and [91], which was already explained within Section 2.7. This is done through the provided open source implementation of the algorithm, which is available through the web page `http://www.golems.org/node/1570`.

[6] `http://ompl.kavrakilab.org/classompl_1_1geometric_1_1SBL.html#afba6f0f7e5db8e5537c56b00c54ba778`
[7] `http://ompl.kavrakilab.org/classompl_1_1geometric_1_1LBKPIECE1.html#gLBKPIECE1`

Chapter 5

Empirical evaluation

This chapter we will discuss the empirical evaluation for all described algorithms within the tool chain. There are different methods to evaluate the performance of the algorithms. The evaluations are performed in the following order

1. Evaluation of the performance of the different inverse kinematics algorithms within an empty space.
2. Evaluation of the performance of the different inverse kinematics algorithms within cluttered space.
3. Evaluation of the performance of the different trajectory planning algorithms within an empty space.
4. Evaluation of the performance of the different trajectory planning algorithms within a cluttered space.
5. Evaluation of the different trajectory path execution planning algorithms
6. Evaluation of the overall performance of the tool chain within an empty simulated environment.
7. Evaluation of the overall performance of the tool chain within a cluttered simulated environment.
8. Evaluation of the overall perform of the tool chain with real hardware.

We will describe the evaluation schema, the corresponding results and also give a brief discussion of the results for each evaluation in the corresponding sections. All of the evaluations beside the last one use 4 different manipulators which are popular in research areas and can are listed below

1. Neuronics Katana 300 6m180
2. Neuronics Katana 400 6m180
3. Kuka Youbot
4. Schunk Powerball

The different manipulators are visualized in Appendix A and the workspaces of the manipulators are visualized in the Appendix B.
The evaluations were performed on on a Intel Core i7 Q 820, 1.73 GHZ with 8 GB of RAM. The operating system was a Ubuntu 12.04 32-bit. The ROS version was Fuerte.

5.1. Evaluation of the different inverse kinematics algorithms

Within the previous Chapter 4.2 we propose three different methods to compute the inverse kinematics of a manipulator. To use this algorithm within a realistic scenario it is important to know about the different advantages and disadvantages of the algorithms.

5.1.1. Evaluation setting

The setting for evaluating these algorithms is very simple and focuses on two major criteria to select an appropriate algorithm. First of all the joint space is sampled in a random manner to find a feasible joint configuration. A feasible joint configuration is only possible if the joint limits are not violated and the robot is not in self-collision when using the joint values. The samples are checked to satisfy these condition. After a joint configuration is generated, the corresponding position of the end effector is calculated. This is done with the forward kinematic solver of KDL. The resulting position is feasible and thus should yield to a valid inverse kinematics solution. In this setting we do not consider any collision. Thus the collision check will always return that nothing is in collision. After the position is calculated, the inverse kinematics algorithm tries to find an acceptable solution. The time, to find such a solution, is measured. Also the positioning and the orientation error is calculated after the inverse kinematics algorithm finds a result. Moreover it is reported if the algorithm finds a solution at all.

5.1.2. Criteria of the evaluation

To perform the evaluation each algorithm is used with each of the manipulators. A result is accepted if the position difference of the each coordinate is lower or equal than 10 cm and the orientation error corresponding to the different axis are lower or equal to 180 degrees. This limits are very tolerant and used to make it easy for the algorithms to find a solution. This is done because the algorithm of OpenRave is wrapped in such a way that it tries to find a solution near the desired position if in another way a solution would not be found. The position error was calculated Equation 5.1

$$\delta_p = (x_{target} - x_{solution})^2 + (y_{target} - y_{solution})^2 + (z_{target} - z_{solution})^2 \tag{5.1}$$

Each of the differences is measured in meter. The orientation error was calculated with the help of the normalized quaternions through equation 5.2

$$\delta_o = abs(1 - dotProduct(o_{solution}, o_{target})^2) \tag{5.2}$$

The orientation error is 1 if the two quaternions a rotated around 180 degrees and 0 if they have the same orientation.
450 sample poses were tested with each combination to produce a good coverage of the euclidean space. The inverse kinematics algorithm is called 10 times for each pose. The result for each call for the same pose is stored in one block. The recalling is used to avoid problems caused by a random initializations, as it is used in the stochastic inverse kinematics and within KDL. To calculate how successful a algorithm performs its calculate we calculate the success rate through equation 5.3. Where N_s is the number of successful found solutions and N_t is the number of tests.

$$Success\ rate = \frac{N_s}{N_t} \tag{5.3}$$

The success is also calculated according to one block. This is done through the block success rate which is calculated through equation 5.4. Where N_s^b is the number of successful found solutions within this block and N_t^b is the number of tests within this block. Thus the block success rate specifies if the how often the algorithm find a solution for a particular pose. Within the evaluation we will use the mean of this block success rates.

$$Success\ rate = \frac{N_s^b}{N_t^b} \tag{5.4}$$

5.1.3. Result of the evaluation

Success rates:

In the following Table 5.1 the success rates as well as the block success rates of the different manipulators are shown. The blocks contains 10 retries. In case of KDL the block succcess rate changes with the block size. This change can be seen in Figure 5.1.

Manipulator	Inverse kinematics algorithm	Success rate	Blocks success rate
Katana 300 6m180	KDL	0.8524	0.9889
Katana 300 6m180	OpenRave	0.9031	0.9061
Katana 300 6m180	Stochastic	1	1
Katana 400 6m180	KDL	0.8700	0.9844
Katana 400 6m180	OpenRave	0.9259	0.9286
Katana 400 6m180	Stochastic	1	1
Youbot	KDL	0.7942	0.9978
Youbot	OpenRave	0.5263	1
Youbot	Stochastic	1	1
Powerball	KDL	0.4922	0.9399
Powerball	OpenRave	0.0756	0.0769
Powerball	Stochastic	1	1

Table 5.1.: Success rate of the different inverse kinematics algorithms for different manipulators in empty space.

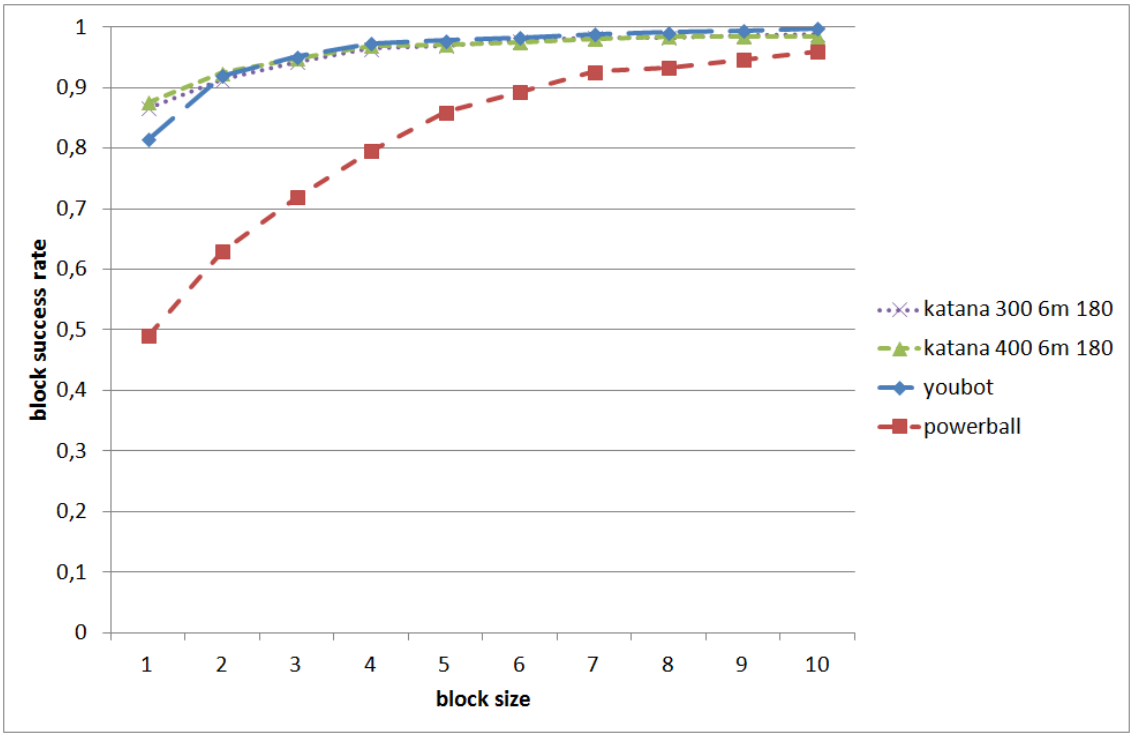

Figure 5.1.: Block success rate in respect to the block size for different manipulators using KDL.

Runtime:

The runtime of the different algorithms in combination with the different manipulators is depicted in Table 5.2 are given in seconds. The setup time of open rave does not influence the run time because it is not included in the tests.

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	0.0018	$2.0283*10^{-004}$
Katana 300 6m180	OpenRave	0.2879	0.15
Katana 300 6m180	Stochastic	0.1604	0.16
Katana 400 6m180	KDL	0.0019	$2.1902*10^{-004}$
Katana 400 6m180	OpenRave	0.3052	0.1520
Katana 400 6m180	Stochastic	0.5070	0.5460
Youbot	KDL	0.0078	$6.8048*10^{-004}$
Youbot	OpenRave	$2.2727*10^{-005}$	$1.5885**10^{-5}$
Youbot	Stochastic	0.2411	0.2850
Powerball	KDL	0.0094	0.0110
Powerball	OpenRave	$2.2854*10^{-005}$	$2.3426*10^{-005}$
Powerball	Stochastic	0.1719	0.1810

Table 5.2.: Timing of the different inverse kinematics algorithms for the different manipulators in empty space.

Position error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	$1.0888*10^{-011}$	$9.2871*10^{-014}$
Katana 300 6m180	OpenRave	$0.0014*10^{-004}$	$9.0000*10^{-024}$
Katana 300 6m180	Stochastic	$1.3859*10^{-004}$	$1.0150*10^{-015}$
Katana 400 6m180	KDL	$9.1352*10^{-012}$	$1.0702*10^{-013}$
Katana 400 6m180	OpenRave	$0.0013*10^{-004}$	$9.0000*10^{-024}$
Katana 400 6m180	Stochastic	$1.7991*10^{-004}$	$1.4941*10^{-016}$
Youbot	KDL	$1.0376*10^{-011}$	$4.3324*10^{-013}$
Youbot	OpenRave	0.0150	0.0150
Youbot	Stochastic	0.0037	$2.9205*10^{-015}$
Powerball	KDL	$6.7007*10^{-013}$	$1.7646*10^{-017}$
Powerball	OpenRave	0.0115	0.0109
Powerball	Stochastic	$9.6515*10^{-004}$	$7.1922*10^{-014}$

Table 5.3.: Position error of the different inverse kinematics algorithms for the different manipulators in empty space.

Orientation error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	$2.3711*10^{-013}$	$2.1684*10^{-019}$
Katana 300 6m180	OpenRave	0.5542	0.5417
Katana 300 6m180	Stochastic	0.0044	$6.4587*10^{-014}$
Katana 400 6m180	KDL	$1.1685*10^{-013}$	$2.1684*10^{-019}$
Katana 400 6m180	OpenRave	0.553	0.5702
Katana 400 6m180	Stochastic	0.0059	$9.2925*10^{-015}$
Youbot	KDL	$2.4471*10^{-013}$	$2.1684*10^{-019}$
Youbot	OpenRave	0.5293	0.5293
Youbot	Stochastic	0.1195	$1.7690*10^{-013}$
Powerball	KDL	$1.9218*10^{-012}$	$3.2032*10^{-015}$
Powerball	OpenRave	0.6770	0.9312
Powerball	Stochastic	0.0518	$7.4739*10^{-012}$

Table 5.4.: Orientation error of the different inverse kinematics algorithms for the different manipulators in empty space.

5.1.4. Discussion of the evaluation results

The above results provided a sound evaluation of the performance of the different approaches. The first result from this evaluation is that the stochastic inverse kinematics is significantly slower than the other algorithms. But always find a solution and show a small error. This observation shows that the proposed algorithm performs very well. It is to mention (as it was also discussed above) that the implementation is very simple and a faster implementation can make a run time decrease possible.

One interesting fact is that there is a problem with the resulting orientation of the solution if OpenRave is used. It seems that there is a problem with the used bridge between OpenRave and ROS. Such an error can occur through the implementation of the bridge which calls the analytic solver. If the analytic solver does not find any solution, the bridge tries to find a solution near the original solution. This is done through a slightly modification of the desired pose. This yields a better success rate but dramatically increases the orientation error.
Another very interesting result of the evaluation is that KDL is very precise and fast. This can be used to increase the success rate if the inverse kinematics algorithm is called multiple times if there is no solution found in the first try. This property comes from the fact that KDL uses a random initial guess and tries to find a solution with this guess. As discussed above in Section 4.2, a bad initial guess result in a failure of the algorithm. To overcome the problem of such a wrong initial guess a retry can be performed and can lead to better results. As the block success rate shows, this methodology is useful to overcome such a problem. Figure 5.1 show the number of retries and the corresponding success rate. The success rate increases with the number of retries, but the steepness depends on the manipulator and especially on the degree of freedom of the manipulator.
All these results yield to an simple conclusion. It can be a good idea to use multiple inverse kinematics solver to deal with the problem of a low success rate. For example KDL can be called multiple times because it is very fast. But if it does not find a solution the stochastic inverse kinematics can be used to get a result. This would lead to a fast inverse kinematics algorithm on average with a high success rate. Using such a cascading of solvers is a interesting topic for future projects. Also a caching strategy for the solutions can be of interest since a possible solution of the problem does not depend on the environment.

5.2. Evaluation of the different inverse kinematicss algorithms within a cluttered space

To evaluate the performance also in a cluttered space the inverse kinematics methods 4.2 are tested within a cluttered environment.

5.2.1. Evaluation setting

The setting to evaluate this algorithm is very simple and focuses on two major criteria to select an appropriate algorithm. First of all the joint space is sampled in a random manner to find a feasible joint configuration. A feasible joint configuration is only possible if the joint limits are not violated. After a feasible joint configuration is generated the corresponding position of the end effector is calculated. This yields to a position which is feasible and thus should result in a valid inverse kinematics solution. The setting utilize an artificial cluttered space. This is accomplished via a simple check if the solution is near the joint solution which was generated and used to generate the target pose. Thus the environment is a nearly fully cluttered environment. After the target pose is calculated the inverse kinematics algorithm aims to find an acceptable solution and the time to find such a solution is measured. Also the positioning error and the orientation error are calculated after the inverse kinematics algorithm finds a solution.

5.2.2. Criteria of the evaluation

Each algorithm was used with each manipulator to perform the evaluation. 450 poses were generated for each combination of manipulator and algorithm. The inverse kinematics algorithm is called 4 times for each pose. This is used to avoid problems through a random initializations, as it is used in the stochastic inverse kinematics and KDL. A result is accepted if the position difference of each coordinate is lower than 10 cm and the orientation error is corresponding to the different axis are lower than 180 Degrees. This limits are very tolerant and used to make it easy for the algorithms to find a solution. The position and orientation error were calculated through the same Equations (5.1, 5.2), which were also used for previous experiment.

5.2.3. Result of the evaluation

Success rates:

The success rate is calculated through Equation 5.3. The blocks success rate is calculated through Equation 5.4. In the following Table 5.5 shows the success rates as well as the block success rates of the different manipulators. The change of the block success rates are shown in Figures 5.2, 5.3.

Manipulator	Inverse kinematics algorithm	Success rate	Blocks success rate
Katana 300 6m180	KDL	0.6202	0.9688
Katana 300 6m180	OpenRave	0.5122	0.5140
Katana 300 6m180	Stochastic	0.8571	1
Katana 400 6m180	KDL	0.6222	0.9822
Katana 400 6m180	OpenRave	0.5341	0.5344
Katana 400 6m180	Stochastic	0.7692	1
Youbot	KDL	0.5440	0.9800
Youbot	OpenRave	0	0
Youbot	Stochastic	0.9029	1
Powerball	KDL	0.0284	0.1559
Powerball	OpenRave	0	0
Powerball	Stochastic	0.2667	1

Table 5.5.: Success rate of the different inverse kinematics algorithms for different manipulators in cluttered space.

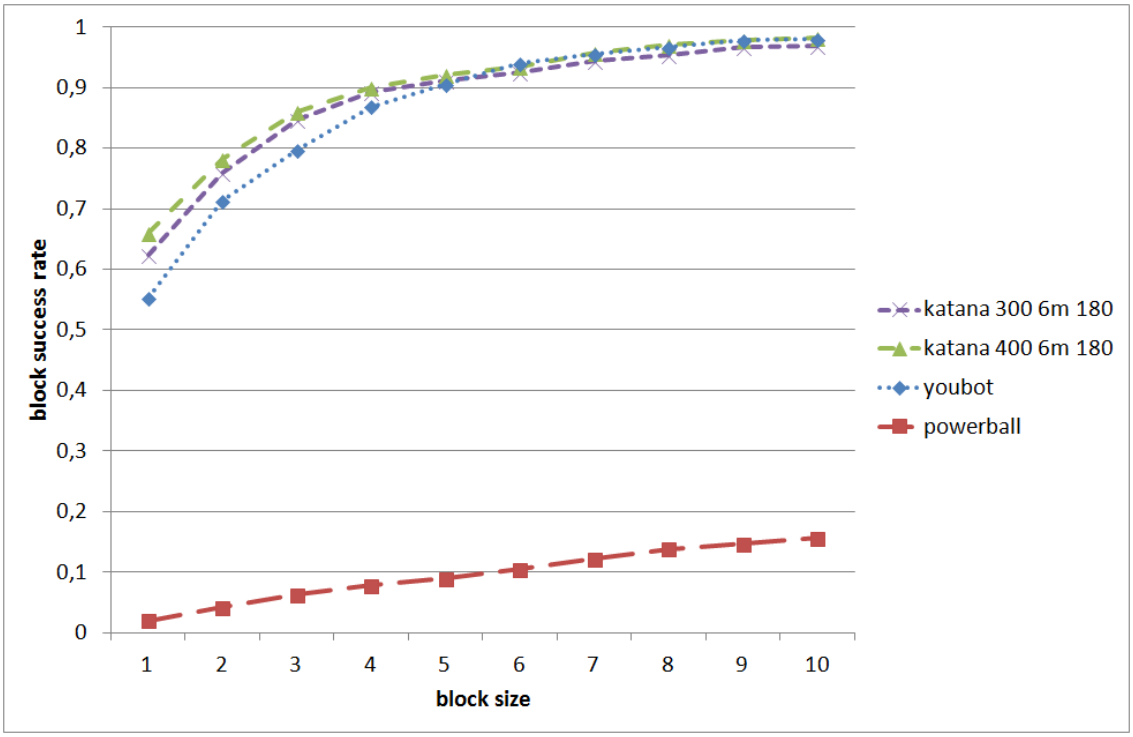

Figure 5.2.: Block success rate in respect to the block size for different manipulators using KDL.

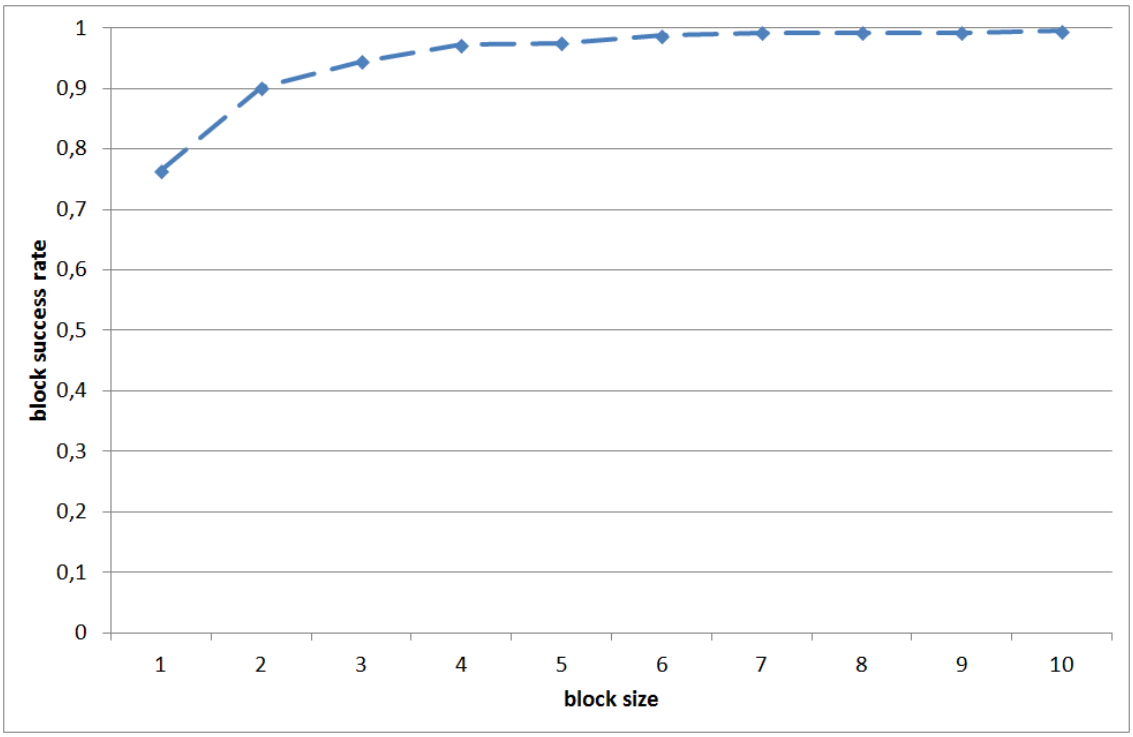

Figure 5.3.: Block success rate in respect to the block size the powerball and the stochstic inverse kinematic.

Runtime:

As above the runtime is given in seconds and the setup time of OpenRave is not counted.

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	0.0020	$2.6442*10^{-004}$
Katana 300 6m180	OpenRave	0.0586	0.0030
Katana 300 6m180	Stochastic	48.2635	34.3455
Katana 400 6m180	KDL	0.0024	$2.6134*10^{-004}$
Katana 400 6m180	OpenRave	0.0497	0.0030
Katana 400 6m180	Stochastic	3.4607	0.3225
Youbot	KDL	0.0078	$1.0000*10^{-003}$
Youbot	OpenRave	-	-
Youbot	Stochastic	7.3312	4.9986
Powerball	KDL	0.0081	$1.0000*10^{-003}$
Powerball	OpenRave	-	-
Powerball	Stochastic	67.5945	62.7940

Table 5.6.: Timing of the different inverse kinematics algorithms for different manipulators in cluttered space.

Position error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	$1.1336*10^{-011}$	$1.5391*10^{-013}$
Katana 300 6m180	OpenRave	$1.6066*10^{-024}$	$1.0967*10^{-024}$
Katana 300 6m180	Stochastic	$2.4369*10^{-011}$	$7.3884*10^{-013}$
Katana 400 6m180	KDL	$1.2019*10^{-011}$	$2.1663*10^{-013}$
Katana 400 6m180	OpenRave	$1.3564*10^{-022}$	$9.6112*10^{-025}$
Katana 400 6m180	Stochastic	$9.6527*10^{-010}$	$1.9142*10^{-015}$
Youbot	KDL	$9.5320*10^{-012}$	$8.2403*10^{-014}$
Youbot	OpenRave	-	-
Youbot	Stochastic	$3.4437*10^{-016}$	$2.1637*10^{-018}$
Powerball	KDL	$1.4572*10^{-012}$	$4.8351*10^{-017}$
Powerball	OpenRave	-	-
Powerball	Stochastic	$2.5434*10^{-004}$	$2.6815*10^{-005}$

Table 5.7.: Position error of the different inverse kinematics algorithms for different manipulators in cluttered space.

Orientation error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	$2.7615*10^{-013}$	$2.1684*10^{-019}$
Katana 300 6m180	OpenRave	$6.3327*10^{-005}$	$2.1684*10^{-019}$
Katana 300 6m180	Stochastic	$1.2336*10^{-009}$	$8.0723*10^{-011}$
Katana 400 6m180	KDL	$2.3913*10^{-013}$	$2.1684*10^{-019}$
Katana 400 6m180	OpenRave	$2.3261*10^{-004}$	$2.1684*10^{-019}$
Katana 400 6m180	Stochastic	$2.2628*10^{-007}$	$8.6620*10^{-014}$
Youbot	KDL	$1.4155*10^{-013}$	$2.1684*10^{-019}$
Youbot	OpenRave	-	-
Youbot	Stochastic	$2.0891*10^{-014}$	$1.5477*10^{-016}$
Powerball	KDL	$1.5968*10^{-012}$	$4.2065*10^{-014}$
Powerball	OpenRave	-	-
Powerball	Stochastic	0.0040	0.0011

Table 5.8.: Orientation error of the different inverse kinematics algorithms for different manipulators in cluttered space.

5.2.4. Discussion of the evaluation results

The results provide an evaluation of the performance of the different approaches. It can be easily seen that KDL is fast and has a small error, which is the same result as within the previous evaluation. If the same pose is retried multiple times the success rate increases as it can be seen in Figure 5.2. A difference is how fast the success rate increases. This is much slower compared to the empty environment. It is also important that in general the success rate is lower than in an empty environment.
Another result of the evaluation is that the stochastic inverse kinematics is significantly slower than KDL. But very often finds a solution and also shows a smaller error.
The last result from this evaluation is that if a rerun is used to increases the success rate, the success rate of the stochastic inverse kinematics increases faster than the success rate of KDL (as it could be seen in Figure 5.2 and5.3). Another difference in is that the stochastic inverse kinematics solver in contrast to KDL reaches a success rate of 1.
As discussed above a combination of the algorithm would further increase the performance. By combining the high success rate of the stochastic inverse kinematics and the fast solutions of KDL.

5.3. Evaluation of the different trajectory path planning algorithms within empty space

In this section we evaluate the performance of the two previous mentioned trajectory planning algorithms (see Section 4.4). The evaluation uses an empty space and thus results in the best possible performance concerning time and success rate. So the result of this evaluation is the upper bound for performance of the algorithm.

5.3.1. Evaluation setting

To evaluate the algorithms we sample a random joint configuration. This configuration is checked if it does not result in a self collision. This check is performed to remove a pose which would result in a self

collision of the manipulator. Then the algorithm is called and the time, position error and orientation error is recorded. To avoid any problems, the manipulator is initialized for every experiment in a position with each joint value at the half of its valid range.
This evaluation is performed for each of the manipulators combined with each of the algorithms. 300 poses are tested with each combination. Each position is 10 times retried to yield a result. To perform the evaluation the only the path planning algorithm is called.
It is very important to notice that each sampled position is feasible and a valid path can be found by applying the different algorithms. To make this more precise the following condition is stated.

Conditions for the feasibility of the path

Each joint can move continuously from one joint position to another one. Thus a joint can move from one position to another one, if there is no collision with the environment along the path of this movement. Another important fact is that each manipulator, we use within the evaluation, can reach a pose that the manipulator is in a straight way and the tip link is pointing into the sky. To describe the possibility that every path is feasible we will use the above information as well as general considerations of the path planning, which we will be explained briefly in the next paragraph.

General properties of path planning There are several simple considerations which can be stated independent of the manipulator.

1. If a manipulator can reach the configuration B (specifying each joint value) from a configuration A. The manipulator can also reach the configuration A from the configuration B. The path from B to A is simple the path from A to B in a reverse order. Thus a path from one configuration to another one is commutative in respect to its feasibility.
2. If a manipulator can reach the configuration B from a configuration A and it is possible to reach C from B, the manipulator can move from A to C by simply using the path from A to B and afterwards the path from B to C. Thus a path from one configuration to another one is transitive.
3. If a manipulator can reach a set of configurations M from a defined configuration I, the following holds: starting from any A $\in$ M it is possible to reach any B $\in$ M. To find a path from A to B, first reverse the path from A to I, which is possible due to the fist general property, and afterwards follow the path from I to B. Such a combination is possible due to the transitivity of a path.

Path conditions met by the Katana 300/400 6m180 and the Youbot The construction of the manipulators is used to construct a procedure to reach any configuration which is not in self collision. The manipulators can be seen in the Appendix in Figures A.1, A.2 and A.4. The first joint is at the bottom of the manipulator and the last joint is at the wrist of the manipulator. The joints are enumerated sequential from the bottom to the wrist. Every possible movement starts in the same starting configuration. This starting configuration is such that the manipulator is straight pointing upright along the z axis.
Through its construction it is easy to see that the first and the last joint can be rotated to their final position from the starting configuration. The ration along these two joints only cause a rotation around the Z axis. After the rotation around these two joints, the manipulator will end up in a configuration, which is similar to the starting configuration with a straight manipulator pointing upright along the z axis. This makes is possible to consider the change of the other joints only in a 2D way because all of these joints are orthogonal to the first and the last one and parallel to each other. This leads to a simple approach to reach the end position.
The first movement is performed at the second joint. The only possible collision is a collision between the base link and the first link. Since the resulting configuration would have the same constellation of the two links such a collision is not possible and the movement can be performed.
The next step is to move the fourth joint. The only possible collision, which can occur is between the last

two links and would also be a constellation which would be the same as within the final configuration. Such a collision could not occur due to the fact that the final configuration is not in collision. Thus this movement is possible.
The last step is to move the last missing joint. This is possible since there are only few possible pairs which can collide. The links at the moving joint cannot result in a collision with the same argument as above. The tip link and the link between the second and the last missing joint can result in a collision. This is only possible if the tip link looks into the direction of this link in the resulting configuration. Such a collision is not possible since this would also result in a collision within the final configuration. The same argument can also be used to explain a possible collision between the tip and the base link. Thus the resulting movement is possible and every configuration which is not in a collision with the manipulator itself can reach any other configuration which is not in collision with the manipulator itself. This follows from the third general consideration above in this section. Thus the conditions for a feasible path is meet for every possible configuration of the three manipulators if it is not in self collision.

Path considerations of the Powerball The construction of the manipulator is used to construct a procedure to reach any configuration which is not in self collision. The manipulator can be seen in the Appendix in Figure A.6. The first joint is at the bottom of the manipulator and the last joint is at the wrist of the manipulator. The joints are enumerated sequential from the bottom to the wrist. Every possible movement starts in the same starting configuration. This starting configuration is such that the manipulator is straight pointing upright along the z axis.
Through the construction of the manipulator we can use a similar strategy to move the manipulator, as for the other manipulators. Each joint which uses the Z-axis within the starting configuration can perform a rotation until the final position is reached. This will result in a configuration, which is similar to the starting configuration.
The next step is to rotate the first orthogonal joint. This can be easily be processed since it can only results in a collision between the first and the base link. This is not possible due to the fact that the constellation of this movement is also the same as the constellation within the resulting configuration, and the resulting configuration there are no collisions allowed.
The next step is to move the last of the orthogonal joints. This can only result in a collision of the tip link with the second link. This is not possible due to the same argument as already mentioned above.
The last step is to move the last joint which is not in its resulting configuration. The collision which can be generated through this movement is only the collisions which is within the resulting pose and thus the movement is possible without any collisions.Thus the resulting movement is possible and every configuration which is not in a collision with the manipulator itself can reach any other configuration which is not in collision with the manipulator itself. This follows from the third general consideration above in this section. Thus the conditions for a feasible path is meet for every possible configuration of the powerball if it is not in self collision.

5.3.2. Result of the evaluation

Success rates:

The success rate is calculated using Equation 5.3. The blocks success rate is calculated using Equation 5.4. The success rate is except for the powerball 1 as it can be seen in Table 5.9.

Manipulator	Path planning	Success rate	Blocks success rate
Katana 300 6m180	LBKPIECE	1	1
Katana 300 6m180	SBLK	1	1
Katana 400 6m180	LBKPIECE	1	1
Katana 400 6m180	SBLK	1	1
Youbot	LBKPIECE	1	1
Youbot	SBLK	1	1
Powerball	LBKPIECE	0.5455	0.60
Powerball	SBLK	0.5455	0.60

Table 5.9.: Success rate of the different trajectory path planning algorithms for different manipulators in empty space.

Run time:

The run time of the different manipulators in combination with the different algorithms is deiced in Table 5.10. The algorithms perform very fast for all the manipulators. But as it can be seen the run time of the Powerball is always the highest in comparison to the other manipulators.

Manipulator	Path planning	Mean	Median
Katana 300 6m180	LBKPIECE	0.0279	0.0260
Katana 300 6m180	SBLK	0.0187	0.0180
Katana 400 6m180	LBKPIECE	0.0277	0.0260
Katana 400 6m180	SBLK	0.0195	0.0180
Youbot	LBKPIECE	0.0371	0.0280
Youbot	SBLK	0.0222	0.0200
Powerball	LBKPIECE	0.1077	0.0930
Powerball	SBLK	0.0920	0.0795

Table 5.10.: Run time of the different trajectory path planning algorithms for different manipulators in empty space.

Position error:

Manipulator	Path planning	Mean	Median
Katana 300 6m180	LBKPIECE	0	0
Katana 300 6m180	SBLK	0	0
Katana 400 6m180	LBKPIECE	0	0
Katana 400 6m180	SBLK	0	0
Youbot	LBKPIECE	0	0
Youbot	SBLK	0	0
Powerball	LBKPIECE	0	0
Powerball	SBLK	0	0

Table 5.11.: Position error of the different trajectory path planning algorithms for different manipulators in empty space.

Orientation error:

Manipulator	Path planning	Mean	Median
Katana 300 6m180	LBKPIECE	$1.6263*10^{-019}$	$2.1684*10^{-019}$
Katana 300 6m180	SBLK	$1.6082*10^{-019}$	$2.1684*10^{-019}$
Katana 400 6m180	LBKPIECE	$1.6552*10^{-019}$	$2.1684*10^{-019}$
Katana 400 6m180	SBLK	$1.6480*10^{-019}$	$2.1684*10^{-019}$
Youbot	LBKPIECE	$1.3733*10^{-019}$	$2.1684*10^{-019}$
Youbot	SBLK	$1.6191*10^{-019}$	$2.1684*10^{-019}$
Powerball	LBKPIECE	$2.1684*10^{-019}$	$2.1684*10^{-019}$
Powerball	SBLK	$1.2649*10^{-019}$	$5.4210*10^{-020}$

Table 5.12.: Orientation error of the different trajectory path planning algorithms for different manipulators in empty space.

5.3.3. Discussion of the evaluation results

It can be easily seen that the resulting success rates, position and orientation errors similar for both algorithms. It is important to notice that the algorithms do not produce any position error. This is important because the position error of the pipeline is not influenced by this step. The orientation error is minimal thus there is no influence within the pipeline occurring from this step.
It is important to notice that SBLK performs faster than the LBKPICE algorithm. The final thing which should be noticed is that both algorithm does not always find a solution if they are combined with the Powerball manipulator. This is a problem of the geometries of the manipulator as well as some modeling issues for the different joints. Some of the joints are model with a range of more then 360°which may cause a problem to the solving algorithms.

5.4. Evaluation of the different trajectory path planning algorithms within cluttered space

We evaluated the performance of the two previous mentioned trajectory planning algorithms. The evaluation uses a cluttered space and thus results in a worse performance than the previous one concerning time and success rate.

5.4.1. Evaluation setting

To evaluate the algorithms we sample a random path within the joint space. To make the path feasible the joint configurations are sampled in such a way that each sample is within the valid range of the previous and the next sampling sample. The valid range of a sample is the range of possible deviation of the path at this sample. Each generated sample is checked if it does not result in a self collision. This results in a possible path for the manipulator. To enforce the manipulator to use this path the collision check is performed in such a way that only joint configurations along the path and some deviation for each joint value does not lead to a collision. If possible joint configuration is not in the valid range of previous sample configuration it is treated as collision. After the path is calculated the algorithm is called and the run time, position and orientation error are recorded. The manipulators initial pose, for each of the tests, is with each joint value at the half of the valid range.
This evaluation is performed with each manipulator in combination with each of the algorithms and 60 generated positions. Each position was retried 5 times. The possible deviation is 10°. The path consists of 5 steps including initial and end joint configuration.

5.4.2. Result of the evaluation

Success rates:

The success rate is calculated using Equation 5.3. The blocks success rate is calculated using Equation 5.4. In the following Table 5.13 the success rates as well as the block success rates of the different manipulators can be seen. The block success rate changes only in cause of the powerball. This change of the block success rates in comparison to the increase of the block size can be seen Figure 5.4.

Manipulator	Path planning	Success rate	Blocks success rate
Katana 300 6m180	LBKPIECE	1	1
Katana 300 6m180	SBLK	1	1
Katana 400 6m180	LBKPIECE	1	1
Katana 400 6m180	SBLK	1	1
Youbot	LBKPIECE	1	1
Youbot	SBLK	1	1
Powerball	LBKPIECE	0.0364	0.2000
Powerball	SBLK	0.1667	0.6364

Table 5.13.: Success rate of the different trajectory path planning algorithms for different manipulators in cluttered space.

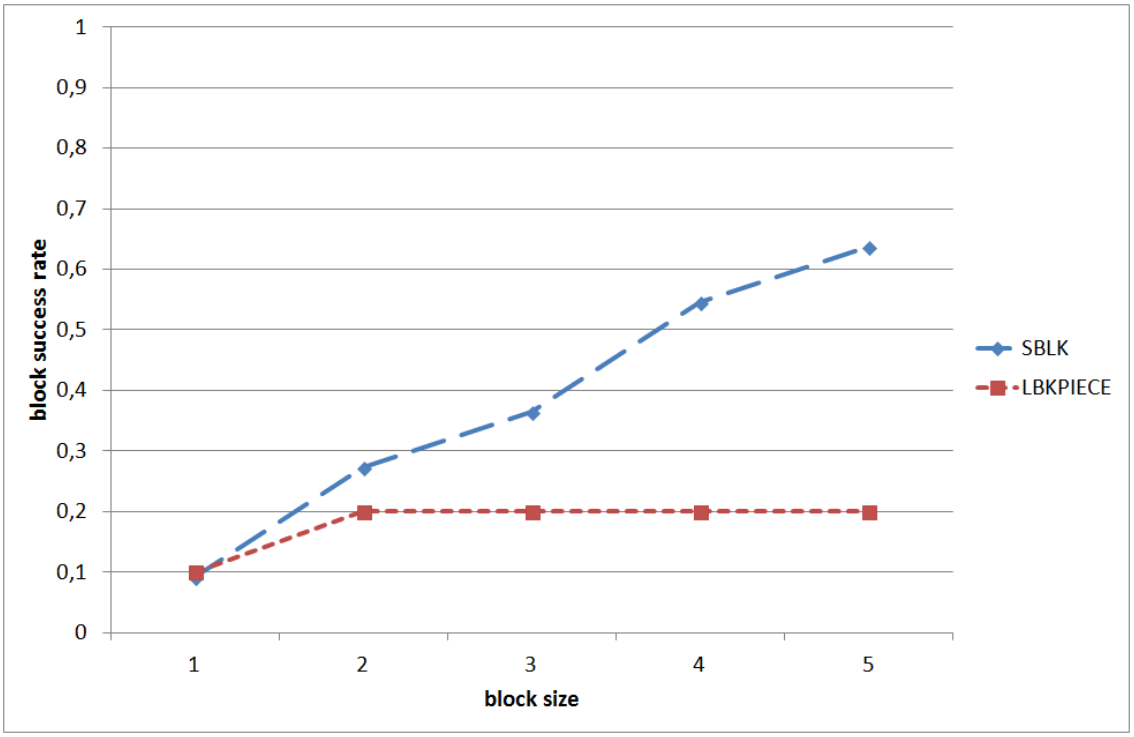

Figure 5.4.: Change of the block success rate for the Powerball in combination with both algorithms in the cluttered environment.

Run time:

The run time of the algorithms in combination with the different manipulators is depicted in Table 5.14.

Manipulator	Path planning	Mean	Median
Katana 300 6m180	LBKPIECE	6.4531	5.3710
Katana 300 6m180	SBLK	1.3067	0.9350
Katana 400 6m180	LBKPIECE	6.9122	5.1845
Katana 400 6m180	SBLK	1.3032	0.8330
Youbot	LBKPIECE	7.2912	5.8185
Youbot	SBLK	1.4953	1.1450
Powerball	LBKPIECE	69.4045	69.4045
Powerball	SBLK	87.2449	81.7380

Table 5.14.: Run time of the different trajectory path planning algorithms for different manipulators in cluttered space.

Position error:

The position error of the algorithms in combination with the different manipulators is depicted in Table 5.15.

Manipulator	Path planning	Mean	Median
Katana 300 6m180	LBKPIECE	0	0
Katana 300 6m180	SBLK	0	0
Katana 400 6m180	LBKPIECE	0	0
Katana 400 6m180	SBLK	0	0
Youbot	LBKPIECE	0	0
Youbot	SBLK	0	0
Powerball	LBKPIECE	0	0
Powerball	SBLK	0	0

Table 5.15.: Position error of the different trajectory path planning algorithms for different manipulators in cluttered space.

Orientation error:

The orientation error of the algorithms in combination with the different manipulators is depicted in Table 5.16.

Manipulator	Path planning	Mean	Median
Katana 300 6m180	LBKPIECE	$1.3010*10^{-019}$	$1.6263*10^{-019}$
Katana 300 6m180	SBLK	$1.6624*10^{-019}$	$2.1684*10^{-019}$
Katana 400 6m180	LBKPIECE	$1.5179*10^{-019}$	$2.1684*10^{-019}$
Katana 400 6m180	SBLK	$1.0481*10^{-019}$	0
Youbot	LBKPIECE	$1.4998*10^{-019}$	$2.1684*10^{-019}$
Youbot	SBLK	$1.7528*10^{-019}$	$2.1684*10^{-019}$
Powerball	LBKPIECE	$2.7105*10^{-019}$	$2.7105*10^{-019}$
Powerball	SBLK	$1.9516*10^{-019}$	$2.1684*10^{-019}$

Table 5.16.: Orientation error of the different trajectory path planning algorithms for different manipulators in cluttered space.

5.4.3. Discussion of the evaluation results

Both algorithms do not produce any significant position or orientation error. This is important because the position error of the pipeline is not influenced by this step. The algorithms perform for the Katana 300/400 6m180 and the Youbot with a similar success rate. The only difference between the algorithms with these manipulators is that SBLK performs faster. The combination of the algorithms and the Powerball manipulator results in a completely different behaviour. In the case of the Powerball manipulator LBKPIECE is faster than SBLK but often does not find a solution. Also a rerun of the algorithms does not always yield to a valid result. But it should be mentioned that a rerun could increase the success rate significantly, as it can be seen in Figure 5.4. This can result in a very problematic situation within a cluttered environment, because it is not possible to find a path even though a path would be possible. The time to find such a path is very high and could result in an abort of the overall process. This can lead to situations where no solution can be found, even though there exists a possibility to move the manipulator to the desired pose.

5.5. Evaluation of the different trajectory path execution planning algorithms

We evaluated the performance of the two trajectory execution planning algorithms presented in Section 4.5. We do not need to consider any type of collision since it is not possible to handle a collision caused by a change of the timings within this step of the tool chain. Neither positioning error nor orientation error is not measured due to the fact that the resulting error of the step can only be evaluated if the trajectory is executed at the manipulator in a real or simulated environment.

5.5.1. Evaluation setting

To evaluate the algorithms we sample a random path within the joint space. Each step is checked whether it does not result in a self-collision. After the path is calculated the algorithm is called and the run time is recorded. This evaluation is performed with each of the manipulators combined with each of the algorithms. Each combination have to solve 60 generated paths. Each position was repeated 5 times. The possible deviation is 10 °. The path consists of 10 steps including initial and end joint configuration.

5.5.2. Result of the evaluation

Success rates:

The success rate is calculated using Equation 5.3. The blocks success rate is calculated using Equation 5.4. The success rate as well as the block success rate is depicted in Table 5.17

Manipulator	Path execution planning	Success rate	Blocks success rate
Katana 300 6m180	Cubic spline	0.9133	1
Katana 300 6m180	Parabolic blend	1	1
Katana 400 6m180	Cubic spline	0.9167	1
Katana 400 6m180	Parabolic blend	1	1
Youbot	Cubic spline	0.9500	1
Youbot	Parabolic blend	1	1
Powerball	Cubic spline	0.9833	1
Powerball	Parabolic blend	1	1

Table 5.17.: Success rate of the different trajectory path execution planning algorithms for different manipulators.

Run time:

The run time of the different algorithms in combination with the different manipulators is depicted in Table 5.18.

Manipulator	Path execution planning	Mean	Median
Katana 300 6m180	Cubic spline	2.0107	2.0100
Katana 300 6m180	Parabolic blend	0.8118	0.8105
Katana 400 6m180	Cubic spline	2.0098	2.0100
Katana 400 6m180	Parabolic blend	0.8479	0.8165
Youbot	Cubic spline	2.0076	2.0070
Youbot	Parabolic blend	0.6635	0.6695
Powerball	Cubic spline	2.0183	2.0150
Powerball	Parabolic blend	1.6451	1.6170

Table 5.18.: Run time of the different trajectory path execution planning algorithms for different manipulators.

5.5.3. Discussion of the evaluation results

There are two facts which are important to point out within this evaluation. The first one is that the cubic spline interpolation take about 2 seconds. This is the default maximum planning time within the ROS arm navigation stack. Thus the algorithm use the complete allowed planning time to find a optimal solution. The second observation is that the parabolic blend method performs faster and always finds a result so it can be easily seen that this method should be the preferred.

5.6. Overall evaluation within the simulation

The experiments within this section evaluates the performance of the complete tool chain within an simulated environment. The used simulation environment is Gazebo[1] [97]. There are some issues that cause unpredictable crashes and thus reduces in some cases the number of tries and retries but it reflects an overall performance of the tool chain. This crashes were caused of unstable software as well as race condition in the starting procedure of the software.

5.6.1. Evaluation setting

To perform the evaluation the joint space is sampled in a random way. The resulting pose of the tool center point is calculated and afterwards checked if it is in self collision. There is also a check performed if the position is above the base plate to overcome problems with collision of the manipulator and the base plate. After a feasible pose is sampled the tool chain tries to move the manipulator tool center point to the given pose. The tool center points of the manipulator can be seen in the Appendix A. The start- and the end-time, the position and the orientation error are recorded. To retrieve the exact pose Gazebo is used as ground truth. But due to some problems with the simulation the position and orientation, at the Katana 300/400 6m180 and the Youbot is measured at the right gripper link. This does not lead to an additional error since the transformation between tool center point and right gripper link is fix during the execution. The gripper links are never moved to open or close the gripper within the evaluation. It is also important to mention that the position of the right gripper link does not change with different possible joint configurations, which leads to the same pose of the tool center point link. The tool chain consists of one of the inverse kinematics solvers discussed above in Section 4.2 and both path planning algorithms. The path planning algorithms are invoked one after each other if the first does not find to a solution. This should not happen often since there are no objects in the evaluation which can cause any collision. This also leads to the lower bound

[1] http://gazebosim.org/

regarding time and upper bound regarding success of the tool chain. The test is performed for 60 poses and 5 retries of each pose. The time limit to perform the evaluation is 50 minutes. As stated above it was not possible to perform all the target poses with all retries. Thus the actual sum of all poses (also every retry of the pose is counted separately) is listed below.

5.6.2. Result of the evaluation

Success rates:

The success rate is calculated using Equation 5.5. Where N is the number of tests and N_s is the number of found solutions. A solution is accepted if the position error which is calculated using Equation 5.1 less then 0.03.

$$success\ rate = \frac{N_s}{N} \tag{5.5}$$

The blocks success rate is calculated using Equation 5.6. Where N^b is the number of tests in a block and N_s^b is the number of found solutions, within a block. A solution is accepted if the position error which is calculated using Equation 5.1 less then 0.03.

$$block\ success\ rate = \frac{N_s^b}{N^b} \tag{5.6}$$

In Table 5.19 the success rates as well as the block success rates of the different manipulators can be seen. The change of the block success rates is depicted in Figures 5.5 and 5.6. Please note that the tests never reach the desired 60 test due to unexpected crashes.

Manipulator	Inverse kinematics algorithm	#tests	Success rate	Blocks success rate
Katana 300 6m180	KDL	52	0.9615	1
Katana 300 6m180	Stochastic	52	1	1
Katana 400 6m180	KDL	52	0.7692	1
Katana 400 6m180	Stochastic	52	1	1
Youbot	KDL	51	0.4118	0.5833
Youbot	Stochastic	50	0.6400	0.7500
Powerball	KDL	52	0.1346	0.1667
Powerball	Stochastic	55	0.6909	0.9231

Table 5.19.: Success rate of the different algorithm combinations for different manipulators in a empty simulated environment.

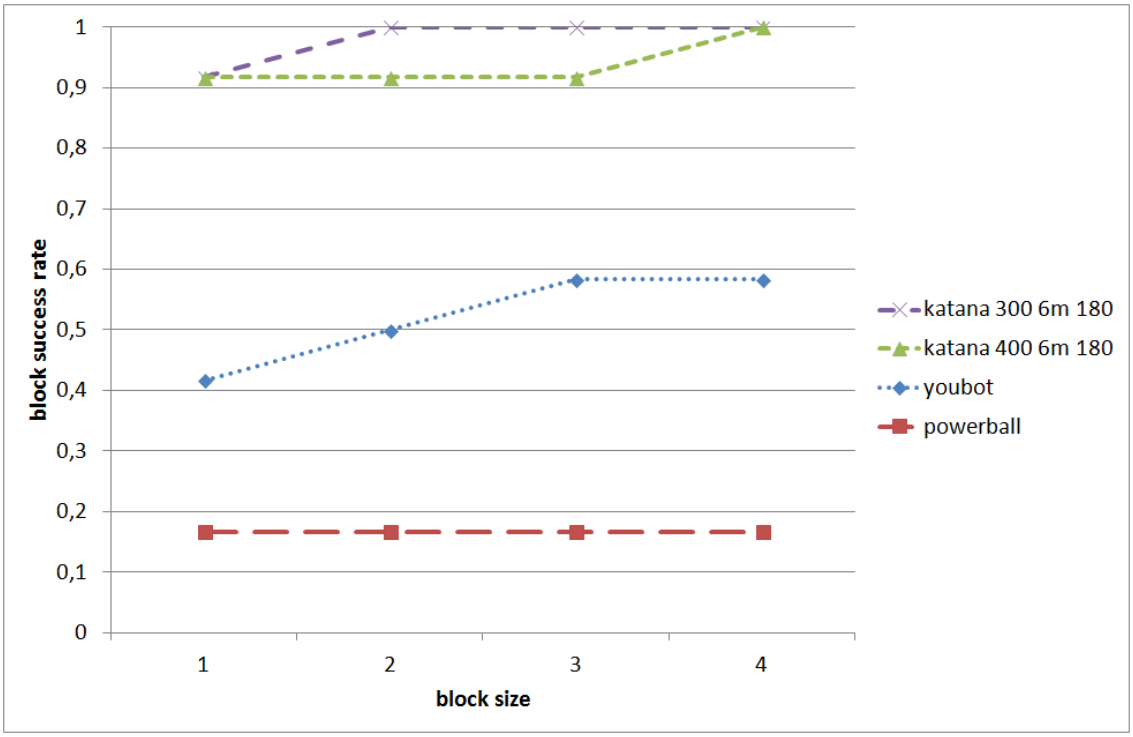

Figure 5.5.: Change of the block success rate of the different manipulators in combination with KDL in a empty simulated environment.

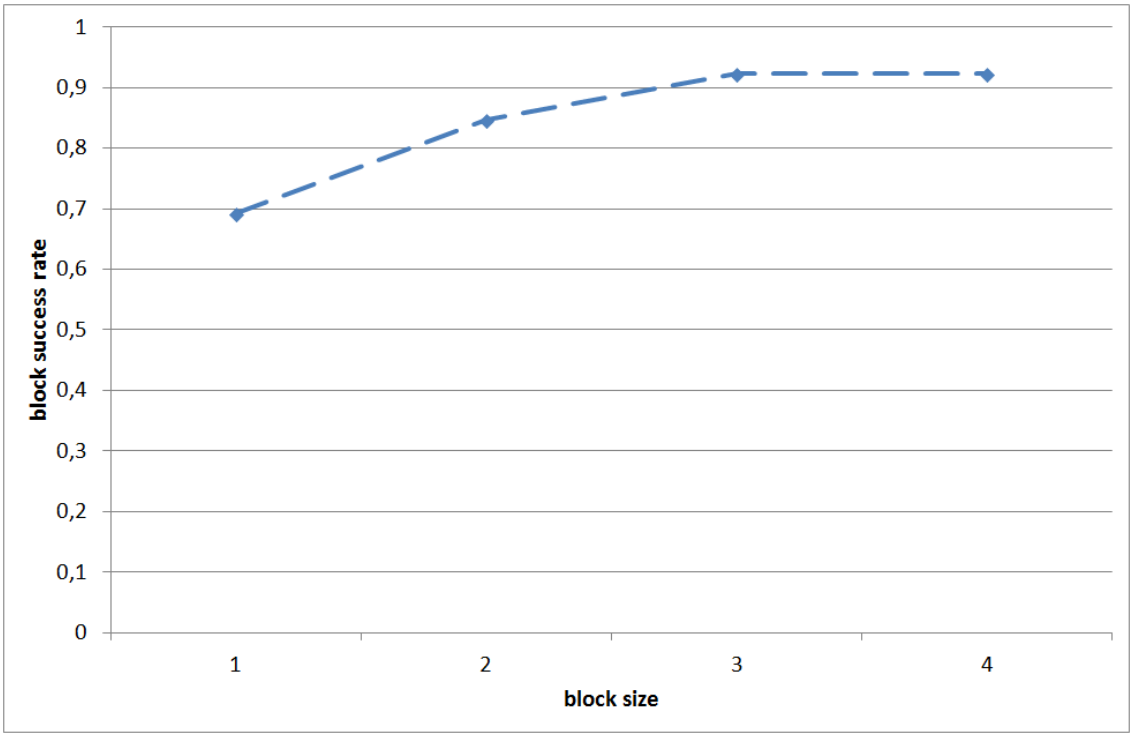

Figure 5.6.: Change of the block success rate of the Powerball in combination with the stochastic inverse kinematics in a empty simulated environment.

Run time:

Table 5.20 depicts the time between the start of the tool chain and a termination message is generated from the tool chain. The runtime values are given in seconds.

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	5.4905	5.5575
Katana 300 6m180	Stochastic	6.8093	6.4475
Katana 400 6m180	KDL	5.3388	5.4395
Katana 400 6m180	Stochastic	5.6652	5.5535
Youbot	KDL	6.0441	5.7400
Youbot	Stochastic	14.6961	16.8155
Powerball	KDL	6.5317	6.1960
Powerball	Stochastic	110.0090	43.0765

Table 5.20.: Run time of the different algorithm combinations for different manipulators in a empty simulated environment. The run time represents the time between the start of the tool chain and a termination message is generated from the tool chain. The run time is given in seconds.

Planning timing:

Table 5.21 shows the time between the start of the tool chain and the tool chain starts to move the manipulator. The runtime values are given in seconds.

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	2.0626	2.0600
Katana 300 6m180	Stochastic	3.4840	2.5470
Katana 400 6m180	KDL	2.0503	2.0440
Katana 400 6m180	Stochastic	2.5237	2.5085
Youbot	KDL	1.8684	2.0580
Youbot	Stochastic	14.6961	16.8155
Powerball	KDL	2.2681	2.2430
Powerball	Stochastic	86.6452	37.1335

Table 5.21.: Planning time of the different algorithm combinations for different manipulators in a empty simulated environment. The planning time reports the time between the start of the tool chain and the tool chain starts to move the manipulator. The planning time is given in seconds.

Position error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	0.0142	$1.4793*10^{-004}$
Katana 300 6m180	Stochastic	$1.4535*10^{-004}$	$7.4825*10^{-005}$
Katana 400 6m180	KDL	0.0529	$6.5204*10^{-005}$
Katana 400 6m180	Stochastic	$8.1876*10^{-005}$	$2.7908*10^{-005}$
Youbot	KDL	0.0438	0.0372
Youbot	Stochastic	0.0290	0.0178
Powerball	KDL	0.5262	0.3685
Powerball	Stochastic	0.1727	$4.8127*10^{-005}$

Table 5.22.: Position error of the different algorithm combinations for different manipulators in a empty simulated environment.

Orientation error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	0.0285	$3.5654*10^{-004}$
Katana 300 6m180	Stochastic	$7.2692*10^{-004}$	$5.6348*10^{-004}$
Katana 400 6m180	KDL	0.1671	$5.5497*10^{-004}$
Katana 400 6m180	Stochastic	$3.8096*10^{-004}$	$2.0234*10^{-004}$
Youbot	KDL	0.1294	0.0851
Youbot	Stochastic	0.0599	0.0568
Powerball	KDL	0.6410	0.8612
Powerball	Stochastic	0.5723	0.6971

Table 5.23.: Orientation error of the different algorithm combinations for different manipulators in a empty simulated environment.

5.6.3. Discussion of the evaluation results

We will only briefly discuss the results and the problems which arise during the simulation. It is important to notice that the result corresponds to the results of the inverse kinematics evaluation. This can be easily seen since the KDL algorithm performs faster but does not always find a solution.
The success rate can also be increased by a rerun of the algorithms as it can be seen in Figures 5.5 and 5.6. This behaviour was previously discussed within the different steps of the tool chain. It is important to notice that if one part of the tool chain is improved the complete tool chain is improved. This can be seen in any case if the inverse kinematic algorithm is changed from KDL to the stochastic inverse kinematic solver.
The position error and the orientation error, excepting the Powerball, are very low if a solution is found. This corresponds to the results of the different parts of the tool chain. It is also important to notice that the planning time is about 2 seconds for a manipulator if KDL is used. It is also important to notice that the problem occurring from the combination of the path planning algorithms and the Powerball also results in a poor performance. Although a rerun of the tool chain overcomes some problems with missing solutions it does not always yield a total success. This can be easily seen in the cases where the Youbot or the Powerball was used.

5.7. Overall evaluation within the simulation and cluttered environment

In this section we will evaluate the performance of the complete tool chain within an simulated environment. For the simulation we use Gazebo. There are some issues that cause unpredictable crashes and thus reduces in some cases the number of tries and retries but it reflects an overall performance of the tool chain. This crashes were caused of unstable software as well as race condition in the starting procedure of the software.

5.7.1. Evaluation setting

To perform the evaluation a path within the joint space is sampled in a random way. Each of the samples is checked if it is not in a self-collision. There is also a check performed if the sample leads to a pose which is above the base plate to overcome problems with collision of the manipulator and the base plate. After a

feasible path is sampled the tool chain tries to move to the tool center point to the desired pose. To make the environment cluttered a similar procedure is used as it was used in the evaluation of the cluttered trajectory path planning algorithms. The environment checks are performed in such a way that ever configuration which is not near a sample point is in collision. Thus the only possible way to move the manipulator is to move it along the sample path.
The start- and the end-time, the position and the orientation error are recorded. To retrieve the exact pose Gazebo is used as ground truth. But due to some problems with the simulation the position and orientation, at the Katana 300/400 6m180 and the Youbot is measured at the right gripper link. This does not lead to an additional error since the transformation between tool center point and right gripper link is fix during the execution. The gripper links are never moved to open or close the gripper within the evaluation. It is also important to mention that the position of the right gripper link does not change with different possible joint configurations, which leads to the same pose of the tool center point link. The tool chain consists of one of the inverse kinematics solvers discussed above in section 4.2. If the Powerball is used only one of the above described path planning algorithms is used instate of both path planning algorithms. The path planning algorithms are performed one after each other, except the Powerball is used, if the first does not find to a solution. This can happen from time to time since there are many virtual objects within the space, which can cause a collisions. We assume that a fully populated space is most difficult. Thus it leads to the upper bound regarding time and lower bound regarding success of the tool chain. The test is performed for 14 poses and 5 retries of each of the poses. The time limit to perform the evaluation for one algorithm combination is 50 minutes. This is used due evaluation needs. Please note that there is no answer from the tool chain if the task can not be performed. Thus the tool chain have to run for ever to check if it finds a solution. As stated above it was not possible to perform all poses with all retries and thus the sum of all poses (also every retry of the pose is counted separately) is listed below.

5.7.2. Result of the evaluation

Success rates:

The success rate is calculated using Equation 5.5. The blocks success rate is calculated using Equation 5.6. In Table 5.24 the success rates as well as the block success rates of the different manipulators is shown. In Figure 5.7 the block success rates and the effect of the changing the block sizes is depicted.

Manipulator	Inverse kinematics algorithm	#tests	Success rate	Blocks success rate
Katana 300 6m180	KDL	23	0.7826	1
Katana 300 6m180	Stochastic	23	0.8696	1
Katana 400 6m180	KDL	30	0.7000	1
Katana 400 6m180	Stochastic	30	0.6333	1
Youbot	KDL	26	0.0385	0.0667
Youbot	Stochastic	25	-	-
Powerball	KDL, SBLK	29	0.2414	0.3333
Powerball	KDL, LBKIEPCE	29	0.2069	0.2222
Powerball	Stochastic, SBLK	29	0.5172	0.5556
Powerball	Stochastic, LBKIEPCE	28	0.1786	0.2222

Table 5.24.: Success rates of the different algorithm combinations for different manipulators in a cluttered simulated environment.

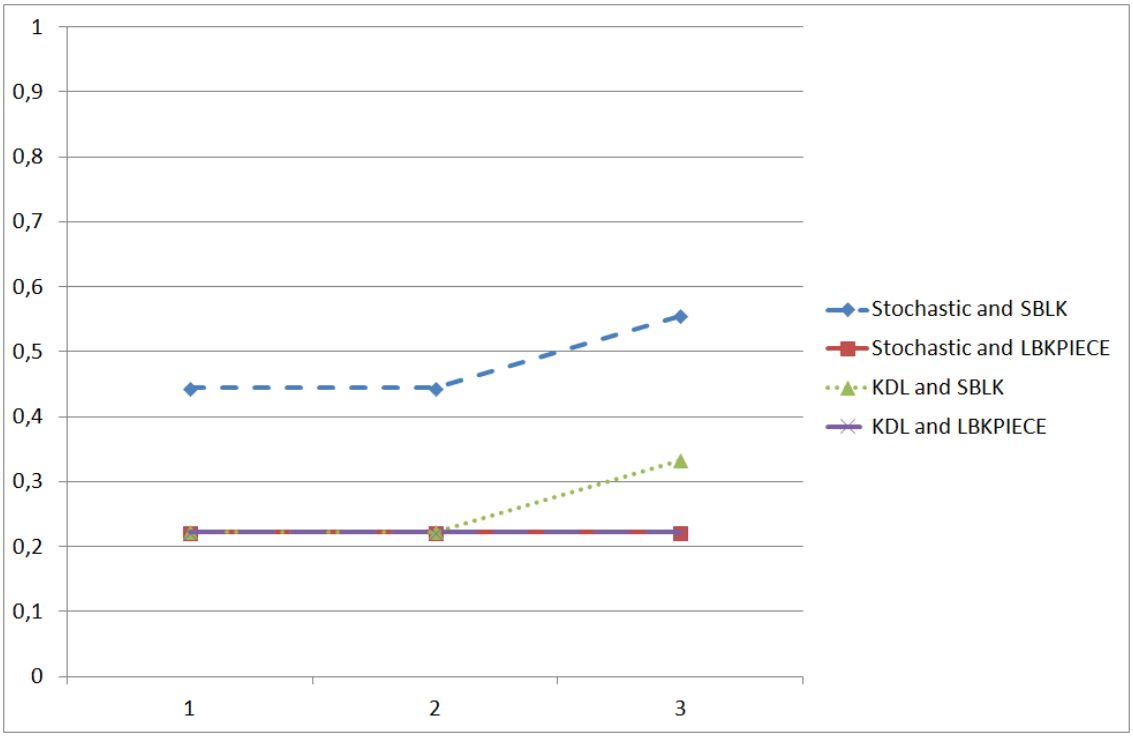

Figure 5.7.: Change of the block success rate of the Powerball in combination with all algorithms in an cluttered simulated environment.

Run time:

Table 5.25 depicts the time between the start of the tool chain and a termination message is generated from the tool chain. The run time is given in seconds.

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	4.4314	4.3725
Katana 300 6m180	Stochastic	5.0913	4.9345
Katana 400 6m180	KDL	4.2750	4.1480
Katana 400 6m180	Stochastic	5.8749	5.5960
Youbot	KDL	-	-
Youbot	Stochastic	-	-
Powerball	KDL, SBLK	232.4213	268.2130
Powerball	KDL, LBKIEPCE	299.3030	360.1450
Powerball	Stochastic, SBLK	247.8285	278.4060
Powerball	Stochastic, LBKIEPCE	353.2520	360.0710

Table 5.25.: Run time of the different algorithm combinations for different manipulators in a cluttered simulated environment. The run time describes the time between the start of the tool chain and a termination message is generated from the tool chain. The run time is given in seconds.

Planning timing:

Table 5.26 shows the time between the start of the tool chain and the tool chain starts to move the manipulator. The run time is given in seconds.

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	3.0652	2.8060
Katana 300 6m180	Stochastic	3.6473	3.5560
Katana 400 6m180	KDL	2.8730	2.8460
Katana 400 6m180	Stochastic	4.3847	4.2090
Youbot	KDL	-	-
Youbot	Stochastic	-	-
Powerball	KDL, SBLK	29.5923	0
Powerball	KDL, LBKIEPCE	0	0
Powerball	Stochastic, SBLK	78.3379	0
Powerball	Stochastic, LBKIEPCE	64.9650	0

Table 5.26.: Planning time of the different algorithm combinations for different manipulators in a cluttered simulated environment. The planning time describes the time between the start of the tool chain and the tool chain starts to move the manipulator. The planning time values is given in seconds.

Position error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	$1.5788*10^{-004}$	$4.9606*10^{-005}$
Katana 300 6m180	Stochastic	$1.4240*10^{-004}$	$3.3651*10^{-005}$
Katana 400 6m180	KDL	0.0015	$7.7399*10^{-005}$
Katana 400 6m180	Stochastic	0.0018	$6.3648*10^{-005}$
Youbot	KDL	0.0791	0.0781
Youbot	Stochastic	0	0
Powerball	KDL, SBLK	0.0089	0.0050
Powerball	KDL, LBKIEPCE	0.0057	0.0019
Powerball	Stochastic, SBLK	0.0062	$8.9964*10^{-006}$
Powerball	Stochastic, LBKIEPCE	0.0095	0.0033

Table 5.27.: Position error of the different algorithm combinations for different manipulators in a cluttered simulated environment.

Orientation error:

Manipulator	Inverse kinematics algorithm	Mean	Median
Katana 300 6m180	KDL	$1.9892*10^{-004}$	$7.3769*10^{-005}$
Katana 300 6m180	Stochastic	$2.3998*10^{-004}$	$9.9312*10^{-005}$
Katana 400 6m180	KDL	0.0025	$1.1933*10^{-004}$
Katana 400 6m180	Stochastic	0.0035	$9.9897*10^{-005}$
Youbot	KDL	0.1096	0.1076
Youbot	Stochastic	-	-
Powerball	KDL, SBLK	0.0121	0.0050
Powerball	KDL, LBKIEPCE	0.0129	0.0105
Powerball	Stochastic, SBLK	0.0182	$5.2870*10^{-004}$
Powerball	Stochastic, LBKIEPCE	0.0210	0.0199

Table 5.28.: Orientation error of the different algorithm combinations for different manipulators in a cluttered simulated environment.

5.7.3. Conclusion of the evaluation

We will now briefly discuss the results and in particular some unpredictable crashes of the simulation. It is important to notice that the results of the cluttered evaluations before are reflected within this conclusion. This can easily be seen by the performance of the LBKIEPCE and the SBLK algorithms. In combination with the Powerball manipulator the SLBK algorithm performs better and the LBKIEPCE is faster. Also the increase of the success rate of algorithms can be seen in Figure 5.7 which is similar to the effect of the trajectory path planning algorithms in a cluttered environment for the Powerball manipulator.
It is also important to notice that the stochastic inverse kinematics performs better in many cases but sometimes also it performs worse. A good example is the Youbot manipulator where the stochastic inverse kinematics does not lead to a solution but the KDL found a solution for some of the target poses. This is a result of the problems which came up with a cluttered environment and the stochastic inverse kinematics. The time to find a solution in such a cluttered environment is to high in order to find a solution before a time out for this part of the tool chain occurs.
Another important observation is that the position and orientation error is as good as within the previous evaluation without any obstacles. On the other hand the time increases significantly and thus results in a problem if an environment is cluttered like the environment used here.

5.8. Overall evaluation using real hardware

We evaluated the performance of the complete tool chain using real hardware in a slightly cluttered environment. The evaluation is performed with the Katana 400 6m180 and the Youbot. There are some issues that cause unpredictable crashes. This crashes were caused of unstable software as well as race condition in the starting procedure of the software. To overcome this problem each part of the evaluation was performed up to four times if a crash happened. There are also some problems with the base driver which is for us the last step in the tool chain. Although it was not considered within this thesis. It have some influence on the performance of the tool chain. The problems as well as the influences will be discussed within the discussion of this evaluation.

5.8.1. Evaluation setting

To perform the evaluation a simple experimental setting was created. This setting can be seen in Figure 5.8. The task is to move small balls from one place to another. To accomplish this task the manipulator is moved to a specified position with an open gripper. After this the manipulator is moved to the ball and the gripper is closed. The next step is to lift the ball. The last step is to move the ball to the new place and to open the gripper. This experiment does not use any kind of perception to avoid problems regarding detection error or problems with the visibility of any objects. The environment is cluttered in such a way that it is not simply possible to use a simple path between two ball positions. The walls in the environment which can be seen in Figure 5.8 create this cluttered environment. In case of the Youbot it is especially hard due to the fact that many positions are at the boarder of the workspace of the manipulator. The balls have a diameter of 4 cm and the holes to place a ball have a diameter of 3.5 cm. This results in a quite challenging task, which needs a high precision of the tool chain to check the performance. During the execution of each of the sub tasks like moving to the ball the success is checked. The run time is not recorded since there are some problems with unexpected software crashes, as mention above. Which prevents a sound and easy determination of the run time. Therefore it would be hard to decide when a task starts and when the task ends. The evaluation uses KDL as well as the stochastic inverse kinematics. To perform the path planning both algorithms are used. SBLK was the first one which can be used within the chain of planning algorithms. Both of the presented trajectory execution planning algorithms where used to check how they perform on real hardware with the driver of the hardware.

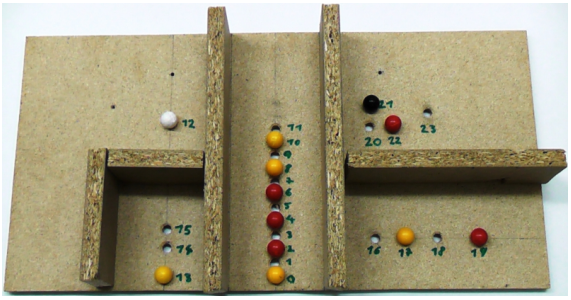

Figure 5.8.: Evaluation setup to perform pick and place tasks with the Katana 400 6m180 and the Youbot

5.8.2. Result of the evaluation

Katana 400 6m180

In Table 5.29 the success rate of the pick and place task using the Katana 400 6m180 is depicted. The success rate is calculated using Equation 5.3.

Inverse kinematics algorithm	Trajectory execution planning	Success rate	Pick ball	Place ball
KDL	Cubic spline	0.60	0.30	0.50
KDL	Parabolic blend	0.47	0.00	0.50
Stochastic	Cubic spline	0.94	0.92	0.83
Stochastic	Parabolic blend	0.58	0.00	0.83

Table 5.29.: Result of the pick and place task using the Katana 400 6m180 manipulator.

Youbot

In Table 5.30 the success rate of the pick and place task using the Youbot is depicted. The success rate is calculated using Equation 5.3.

Inverse kinematics algorithm	Trajectory execution planning	Success rate	Pick ball	Place ball
KDL	Cubic spline	0.75	0.66	0.83
KDL	Parabolic blend	0.71	0.58	0.83
Stochastic	Cubic spline	0.75 (0.81)	0.58 (0.66)	0.83
Stochastic	Parabolic blend	0.54 (0.60)	0.50	0.25

Table 5.30.: Result of the pick and place task using the Youbot manipulator. The percentages within the brackets are calculated with movements which were not finished successfully due to a crash in the diver software.

5.8.3. Discussion of the evaluation results

Before we start with the conclusion of this evaluation we will briefly discuss some observed issues during the evaluation. Which had an impact on the outcome of the evaluation. First of all, it is important to mention that there were some unexpected software crashes, which cause the task to fail and result in a restart of the task. The second important thing to mention is that the driver to move the real manipulator has some issues. Which have a huge impact on the evaluation. The driver which is used for the Youbot has some problems to finish a given trajectory. This can cause the manipulator to stop at a wrong position. This problem can be overcome due to a recall of the tool chain to perform a sub task (e.g. to move again to the position of the ball). But this can lead to a bad situation since in the case of a position with multiple possible solutions of the inverse kinematics the manipulator can try to move to the first solution and afterwards move to the next solution but does not finish one of them correctly. This is also the reason why there is sometimes movements of the manipulator, which does not lead to a success. Nor the movement was sometimes not as accurate as expected from the tool chain, which can be seen in more detail in Appendix C. The driver which is used for the Katana 400 6m180 has the problem that often causes a crash of the manipulator and the environment. The driver itself reduces a given trajectory to a trajectory with a maximum of 16 different joint configurations but does not check if this reduction results in a collision-free path. To avoid the crashes and thus damage on the hardware the movement used intermediate positions, which are above the hole. This intermediate points are part of the control flow to pick up and place a ball. The intermediate points reduce the success rate. This results from the fact that the tool chain could not find a solution to some of the additional points, which could be seen in more detail within Appendix C.
Beside the problems with the driver, the evaluation shows some interesting results, which we will discuss now in detail. First of all the parabolic blend function performance is worse in respect to the cubic spline planning. The evaluation also shows that the stochastic inverse kinematics algorithm finds more solution than KDL, if it is not used with the parabolic blend functions. But it is worth to mention that KDL performs better in the case of a fault in the driver software. This results in the fact that the stochastic inverse kinematics algorithm finds different solutions to the same pose and thus can result in a movement without success, if the driver software crashes from time to time. This problem can be overcome with the help of a caching strategy within the inverse kinematics algorithm.

Chapter 6

Conclusion

Within this thesis the ROS[1] tool chain to handle manipulation taks with different manipulators was investigate. The tool chain is based on arm navigation stack and extended with a new inverse kinematic solver. The focus was to point out problems and to improve the tool chain to successfully finish simple positioning tasks with a given manipulator. The only necessary information to work with this tool chain is a robot description and a driver to communicate with the hardware itself. Each of the needed different sub-tasks was discussed and also the methods used within the tool chain were explained in detail. The evaluation in Chapter 5 shows that the tool chain can be used quite good to perform a precise positioning of a part of the manipulator with a high success rate. The evaluations were performed with different popular research and teaching manipulators. Also a simple pick and place task was performed with different manipulators, which shows that the tool chain can finish this tasks with success even if the environment is cluttered.
It turns out that the used version of the arm navigation stack is not stable enough to be used out of the box. The new version of the arm navigation stack MoveIt may change this in future [92]. The evaluation shows that not all manipulator work with the default algorithms. Especially the inverse kinematic algorithm (KDL) which is used as default causes major problems. The low success to find a solution causes the complete tool chain to fail. Thus it is not possible to successfully perform a pick and place task. To overcome this the limitations of inverse kinematics solvers such as KDL with different manipulators we developed a new algorithm for the inverse kinematic, which is called stochastic inverse kinematic solver. This solver is based a global optimization algorithm and shows in the experiments in a simulated environment as well as using real hardware that the algorithm increases the success rate of the tool chain.
Additionally through the evaluation it turned out that the used driver for the manipulators has some limitations which cause problems. For example the driver for the Katana 400 6m180 can only deal with 16 trajectory points. Thus it is not always possible to move the manipulator as planed from the algorithms. This can cause the tool chain to fail.

[1] http://www.ros.org/wiki/

Chapter 7

Future Work

As already briefly discussed a combination of the different inverse kinematics methods can be useful to achieve a better performance of the tool chain concerning run time and success rate. Another topic, has to be addressed, is the caching of the different solutions of the inverse kinematics due to the fact that a solution is independent of the environment.
A topic of interest would also be path planning algorithms, which provides a better scaling with the number of objects. This would have a strong impact to the performance of the whole tool chain within cluttered environments.
Faster as well as a more stable implementations of different trajectory execution planning algorithms can also be seen as a future direction to improve the tool chain. Due to the fact that the tool chain is based on the assumption that the environment does not change during the execution a local planning method, which can deal with small changes of the environment, or some changes due to the execution on the real robot would be an interesting topic within a future work.
Another interesting topic would be the usage of multiple results of the inverse kinematics of a single pose to use this information to get a shorter as well as faster trajectory.
Although all parts of the tool chain have been intensively empirical evaluated there are more open issues like to evaluate the influence of parameters (e.g. population size of for the stochastic inverse kinematic solver). Additionally it would be necessary to prove if the assumption that the hardest task for the tool chain is a completely cluttered environment is valid.

Appendix A

The manipulators

This Appendix contains a description of the different manipulators. The base as well as the tip link are marked with there coordinate systems. The colour of the coordinate system is red for the X-axis, green for the Y-axis and blue for the Z-axis. The coordinate systems are right hand coordinate systems.

A.1. Katana 300 6m180

Within Figure A.1 the Katana 300 6m180 with the base as well as the tip link marked with there coordinate systems is depicted.

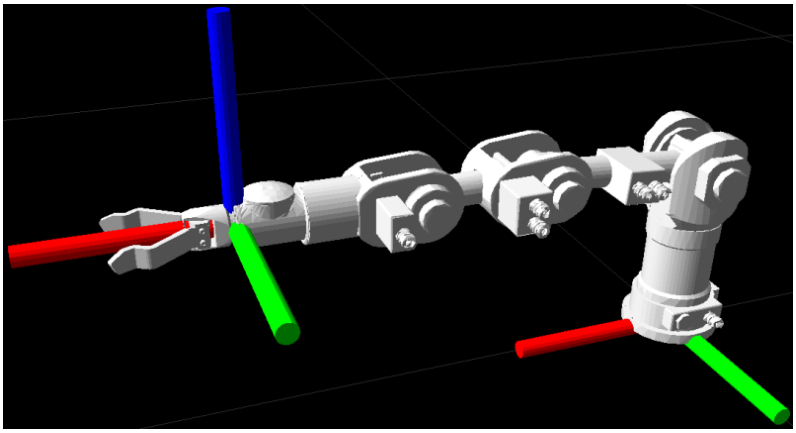

Figure A.1.: The Katana 300 6m180 viewed in rvize (`http://www.ros.org/wiki/rviz`).

A.2. Katana 400 6m180

Within Figure A.2 the Katana 400 6m180 with the base as well as the tip link marked with there coordinate systems is depicted. Figure A.5 shows the real Katana 400 6m180 manipulator.

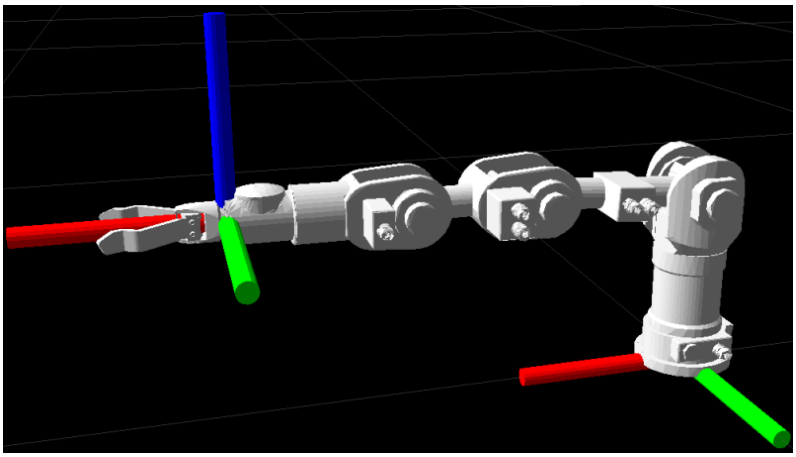

Figure A.2.: The Katana 400 6m180 viewed in rvize (`http://www.ros.org/wiki/rviz`).

Figure A.3.: The Neuronics Katana 400 6m180 manipulator. The image is taken from `http://www.openpr.de/news/229910/Quantensprung-Roboter-funktioniert-ohne-Computeranschluss.html`.

A.3. Youbot

Within Figure A.4 the youbot with the base as well as the tip link marked with there coordinate systems is depicted. Figure A.5 shows the real youbot manipulator.

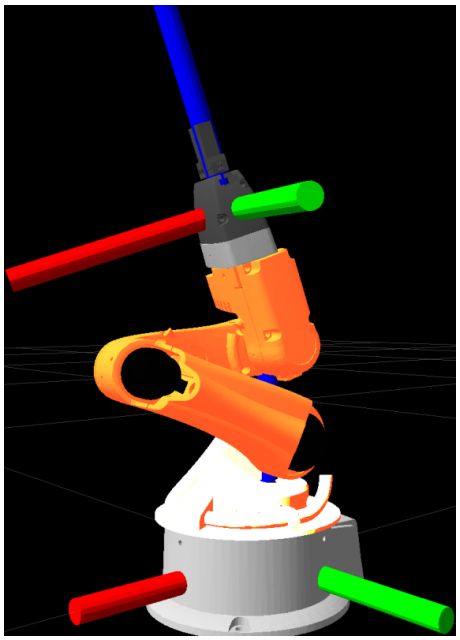

Figure A.4.: The Youbot viewed in rvize (`http://www.ros.org/wiki/rviz`).

Figure A.5.: The Kuka YouBot manipulator. The image is taken from `http://www.youbot-store.com/youbot-store/youbots/products/kuka-youbot-5-degree-of-freedom-arm-with-2-finger-gripper`.

A.4. Powerball

Within Figure A.6 the powerball with the base as well as the tip link marked with there coordinate systems is depicted. Figure A.7 shows the real powerball manipulator.

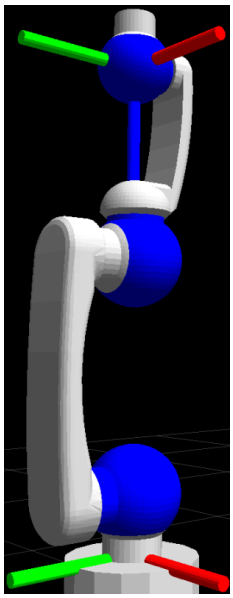

Figure A.6.: The Powerball viewed in rvize (http://www.ros.org/wiki/rviz).

Figure A.7.: The Schunk Powerball manipulator. The image is taken from http://mobile.schunk-microsite.com/?id=9.

Appendix B

Workspace of the manipulators

This Appendix contains the different workspaces of the manipulators which are used during the evaluation. Each of the workspaces will be presented in projection on the X-Y, X-Z, Y-Z plane. To generate this workspaces plots the joint space was sample uniform in each axis. The pose of each sample was calculated with the help of the forward kinematic solver of KDL. This pose was afterwards projected on each of the plains.

B.1. Katana 300 6m180

The workspace of the Katana 300 6m180 projected on the X-Y plane is depicted in Figure B.1, projected on the X-Z plane is shown in Figure B.2 and projected on the Y-Z plane is depicted in Figure B.3.

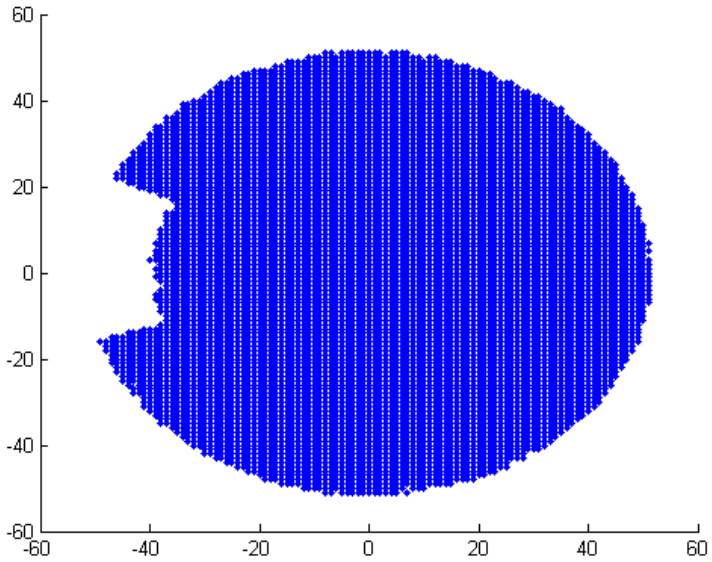

Figure B.1.: Workspace of the Katana 300 6m180 projected on the X-Y plane.

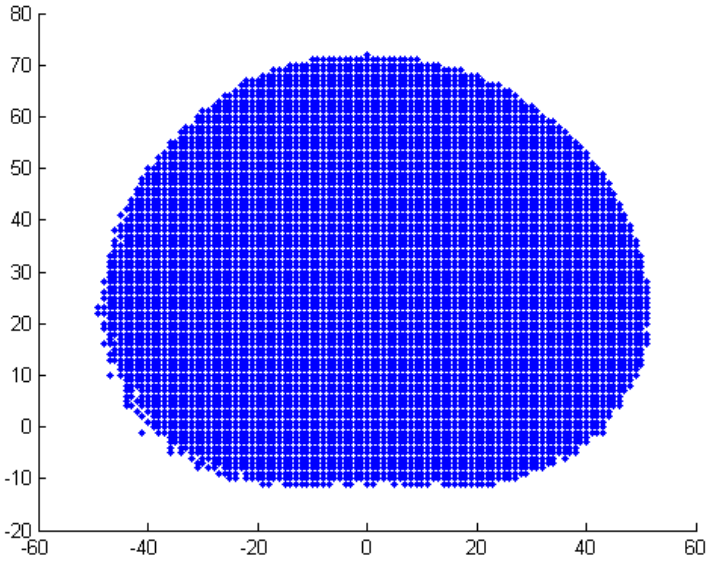

Figure B.2.: Workspace of the Katana 300 6m180 projected on the X-Z plane.

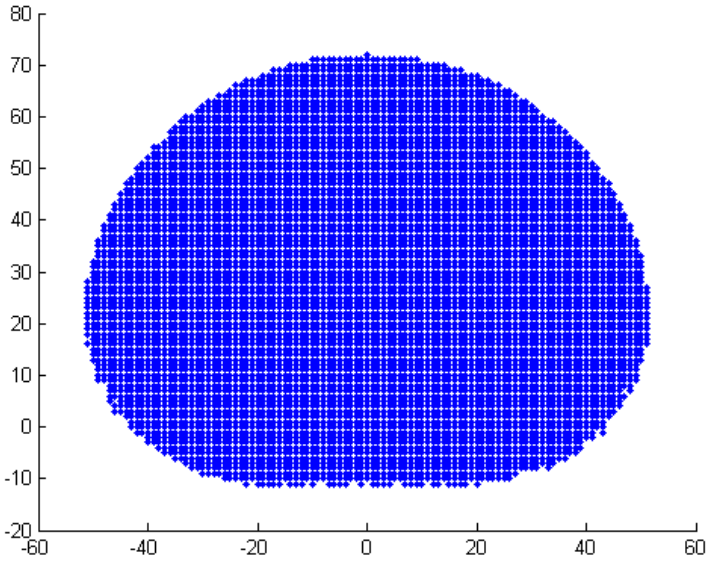

Figure B.3.: Workspace of the Katana 300 6m180 projected on the Y-Z plane.

B.2. Katana 400 6m180

The workspace of the Katana 400 6m180 projected on the X-Y plane is depicted in Figure B.4, projected on the X-Z plane is shown in Figure B.5 and projected on the Y-Z plane is depicted in Figure B.6.

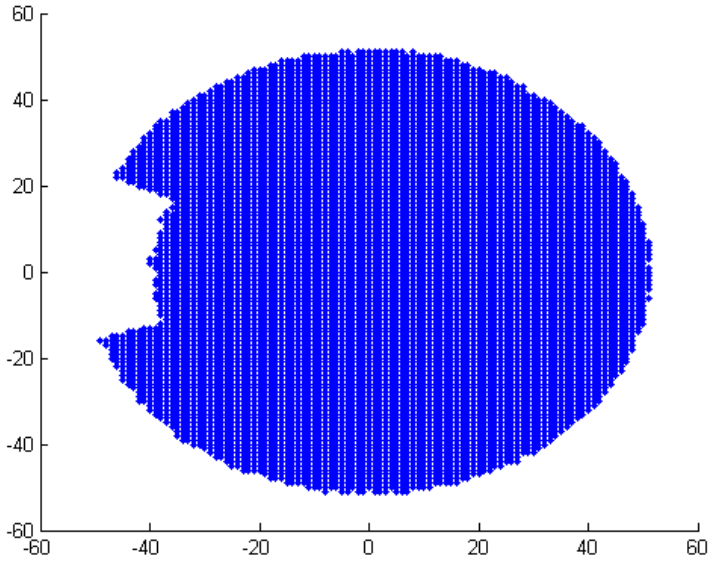

Figure B.4.: Workspace of the Katana 400 6m180 projected on the X-Y plane.

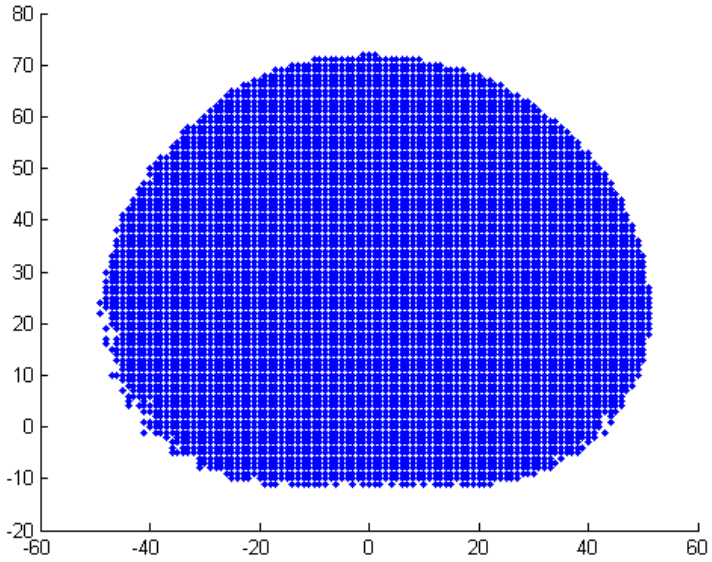

Figure B.5.: Workspace of the Katana 400 6m180 projected on the X-Z plane.

B.3. Youbot

The workspace of the Youbot projected on the X-Y plane is depicted in Figure B.7, projected on the X-Z plane is shown in Figure B.8 and projected on the Y-Z plane is depicted in Figure B.9.

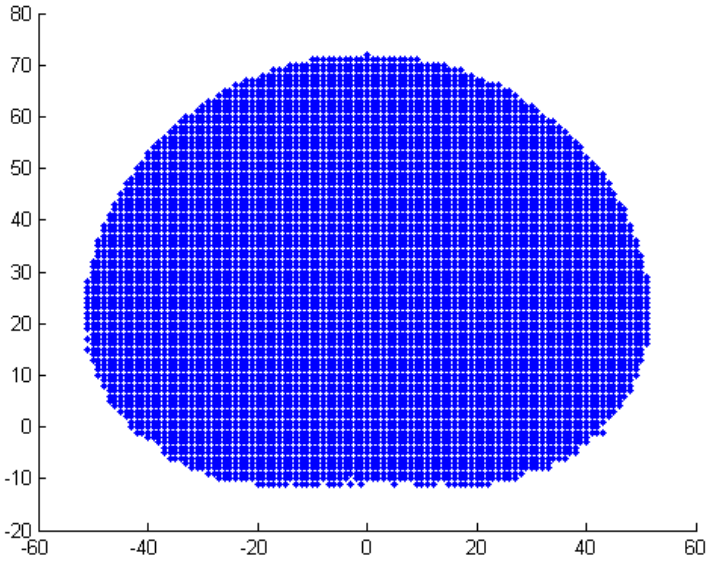

Figure B.6.: Workspace of the Katana 400 6m180 projected on the Y-Z plane.

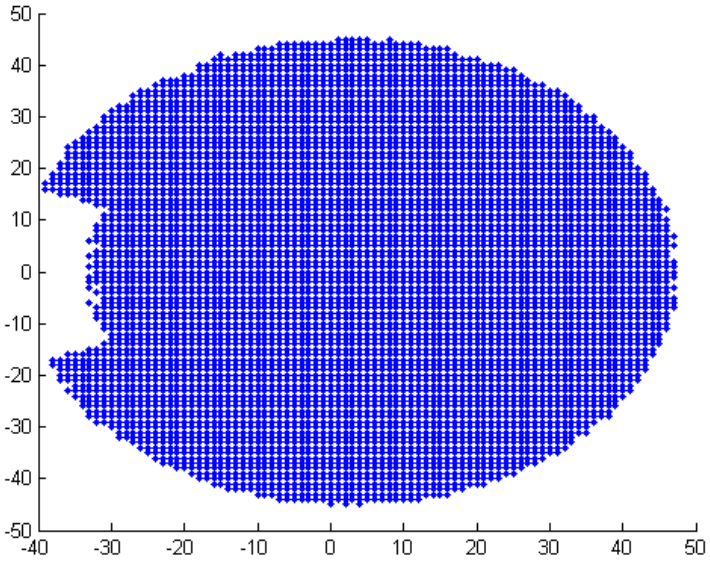

Figure B.7.: Workspace of the Youbot projected on the X-Y plane.

B.4. Powerball

The workspace of the Powerball projected on the X-Y plane is depicted in Figure B.10, projected on the X-Z plane is shown in Figure B.11 and projected on the Y-Z plane is depicted in Figure B.12.

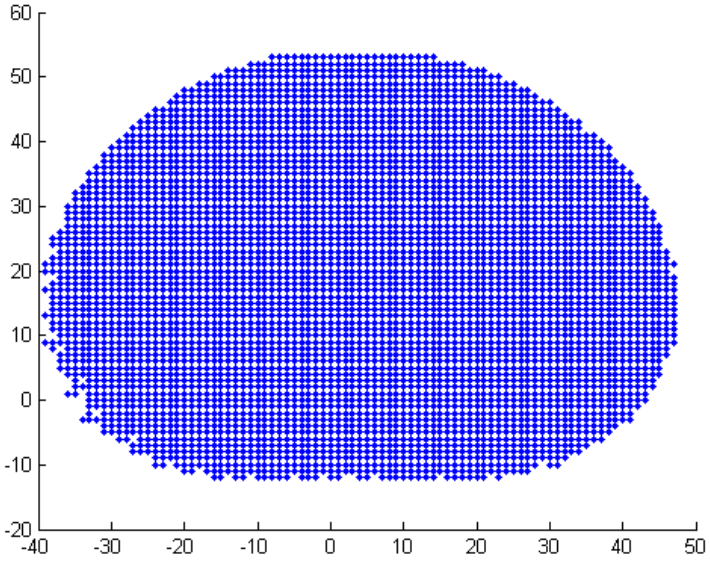

Figure B.8.: Workspace of the Youbot projected on the X-Z plane.

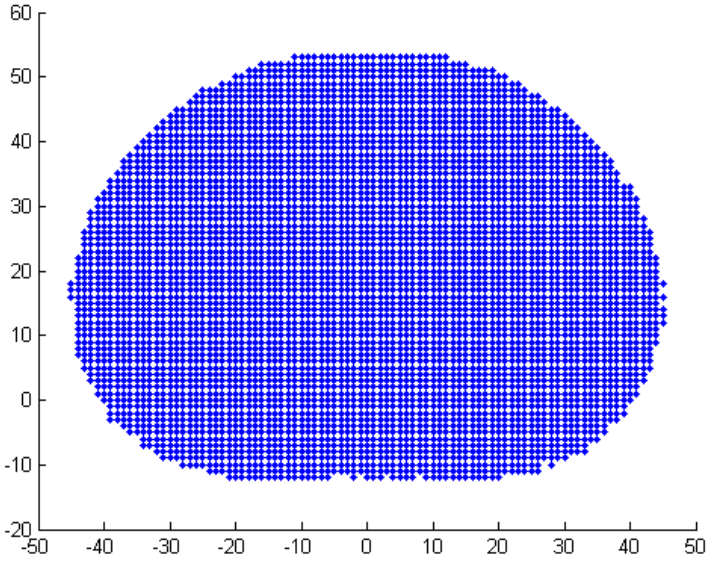

Figure B.9.: Workspace of the Youbot projected on the Y-Z plane.

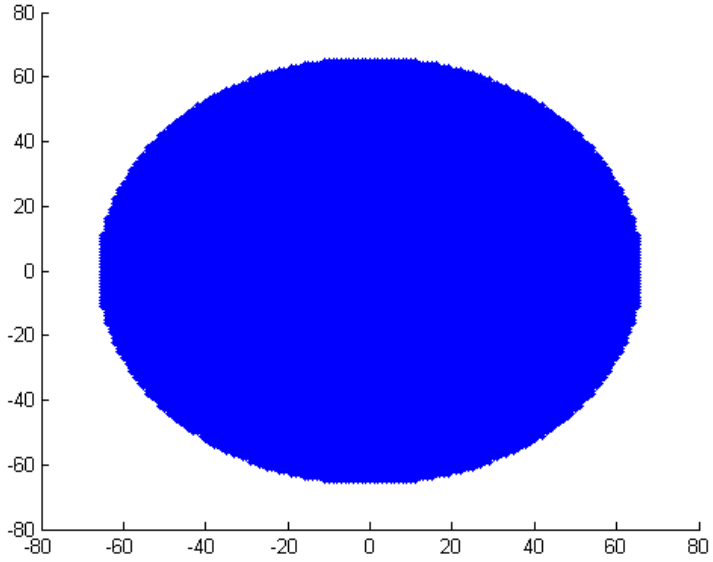

Figure B.10.: Workspace of the Powerball projected on the X-Y plane.

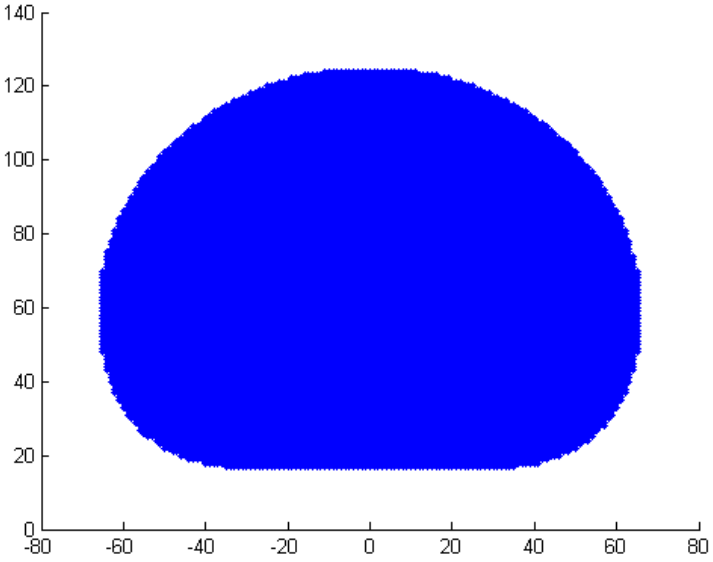

Figure B.11.: Workspace of the Powerball projected on the X-Z plane.

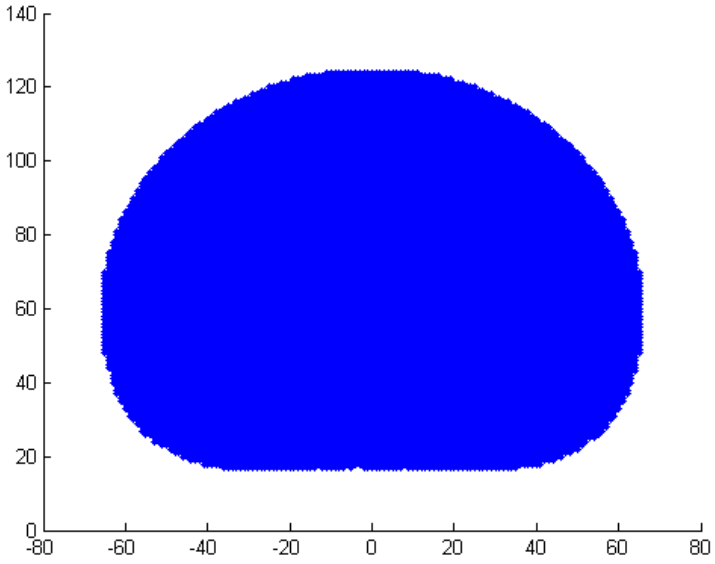

Figure B.12.: Workspace of the Powerball projected on the Y-Z plane.

Appendix C

Evaluation result on real hardware raw data

C.1. Katana 400 6m180

To pick a ball the gripper of the manipulator is opened and the tool center point is placed 10 cm above the hole (p1). Then it is moved to a pre-grasp position which is 3 cm above the hole (p2). The next step is to move the manipulator to the ball and close the gripper (p3). After this step the tool center point is placed in a lift position which is 3 cm above the hole (p4) to test if the ball is really picked. The last step to pick a ball is to move the tool center point 10 cm above the hole (p5).
To place the ball the tool center point is placed 10 cm above the hole (p6). Afterwards it is placed 1 cm above the hole (p7) and the gripper is opened. The last step to place a ball is to move the tool center point 10 cm above the hole (p8).

C.1.1. KDL and cubic spline

Pick ball						Place ball			
Position	p1	p2	p3	p4	p5	Position	p6	p7	p8
0	X	X	X	-	-	15	X	X	X
2	X	X	X	X	X	18	-	X	-
13	X	X	X	X	X	1	X	X	X
4	X	X	X	X	X	14	X	X	X
17	X	X	X	X	X	3	X	X	X
6	-	X	X	X	-	16	X	X	X
19	-	X	X	X	-	5	X	X	X
8	-	X	X	X	-	23	-	-	-
22	-	-	-	-	-	7	-	X	-
10	-	X	X	X	-	20	-	X	-
21	-	-	-	-	-	9	-	X	-
12	-	-	-	-	-	11	-	X	-

Table C.1.: Success of different movements within the evaluation result on real hardware and the use of the Katana 400 6m180 and KDL and cubic spline. A cross means that the Katana 400 6m180 was able to move to the specified pose.

C.1.2. KDL and parabolic blend

Pick ball						Place ball			
Position	p1	p2	p3	p4	p5	Position	p6	p7	p8
0	X	X	-	-	-	15	X	X	X
2	X	X	-	-	-	18	-	X	-
13	X	X	-	-	-	1	X	X	X
4	X	X	-	-	-	14	X	X	X
17	X	X	-	-	-	3	X	X	X
6	X	X	-	-	-	16	X	X	X
19	-	X	X	X	-	5	X	X	X
8	-	X	X	X	-	23	-	-	-
22	-	-	-	-	-	7	-	X	-
10	-	X	X	X	-	20	-	X	-
21	-	-	-	-	-	9	-	X	-
12	-	-	-	-	-	11	-	X	-

Table C.2.: Success of different movements within the evaluation result on real hardware and the use of the Katana 400 6m180 and KDL and parabolic blend. A cross means that the Katana 400 6m180 was able to move to the specified pose.

C.1.3. Stochastic inverse kinematic and cubic spline

Pick ball						Place ball			
Position	p1	p2	p3	p4	p5	Position	p6	p7	p8
0	X	X	X	X	X	15	X	X	X
2	X	X	X	X	X	18	X	X	X
13	X	X	X	X	X	1	X	X	X
4	X	X	X	X	X	14	X	X	X
17	X	X	X	X	X	3	X	X	X
6	X	X	X	X	X	16	X	X	X
19	X	X	X	X	X	5	X	X	X
8	X	X	X	X	X	23	-	X	-
22	-	X	X	X	-	7	X	X	X
10	X	X	X	X	X	20	X	X	-
21	X	X	X	X	X	9	X	X	X
12	X	X	X	X	X	11	-	X	-

Table C.3.: Success of different movements within the evaluation result on real hardware and the use of the Katana 400 6m180 and stochastic inverse kinematic and Cubic Spline. A cross means that the Katana 400 6m180 was able to move to the specified pose.

C.1.4. Stochastic inverse kinematic and parabolic blend

Pick ball						Place ball			
Position	p1	p2	p3	p4	p5	Position	p6	p7	p8
0	X	X	-	-	-	15	X	X	X
2	X	X	-	-	-	18	X	X	X
13	X	X	-	-	-	1	X	X	X
4	X	X	-	-	-	14	X	X	X
17	X	X	-	-	-	3	X	X	X
6	X	X	-	-	-	16	X	X	X
19	X	X	-	-	-	5	X	X	X
8	X	X	-	-	-	23	-	X	-
22	-	X	X	-	-	7	X	X	X
10	X	X	-	-	-	20	X	X	X
21	X	X	-	-	-	9	X	X	X
12	X	X	-	-	-	11	-	X	-

Table C.4.: Success of different movements within the evaluation result on real hardware and the use of the Katana 400 6m180 and stochastic inverse kinematic and parabolic blend. A cross means that the Katana 400 6m180 was able to move to the specified pose.

C.2. Youbot

To pick a ball the gripper of the manipulator is opened and the tool center point is placed in a pre-grasp position 3 cm above the hole (p1). The next step is to move the manipulator to the ball and to close the gripper (p2). After this step the tool center point is placed in a lift position which is 3 cm above the hole (p3) to test if the ball is really picked.
To place the ball the tool center point is placed 1 cm above the hole (p4) and the gripper is opened.

C.2.1. KDL and cubic spline

Pick ball				Place ball	
Position	p1	p2	p3	Position	p4
0	X	X	X	15	X
2	X	X	X	18	X
13	X	X	X	1	X
4	X	X	X	14	X
17	X	X	X	3	X
6	X	X	X	16	X
19	X	X	X	5	X
8	X	X	X	23	-
22	-	-	-	7	X
10	X	X	X	20	-
21	-	-	-	9	X
12	-	-	-	11	X

Table C.5.: Success of different movements within the evaluation result on real hardware and the use of the Youbot and KDL and cubic spline. A cross means that the Youbot was able to move to the specified pose.

C.2.2. KDL and parabolic blend

Pick ball				Place ball	
Position	p1	p2	p3	Position	p4
0	X	X	X	15	X
2	X	X	X	18	X
13	X	X	X	1	X
4	X	-	X	14	X
17	X	X	X	3	X
6	X	X	X	16	X
19	X	X	X	5	X
8	X	-	-	23	-
22	-	-	-	7	X
10	X	X	X	20	-
21	-	-	-	9	X
12	-	-	-	11	X

Table C.6.: Success of different movements within the evaluation result on real hardware and the use of the Youbot and KDL and parabolic blend. A cross means that the Youbot was able to move to the specified pose.

C.2.3. Stochastic inverse kinematic and cubic spline

Pick ball				Place ball	
Position	p1	p2	p3	Position	p4
0	X	X	X	15	X
2	X	X	X	18	X
13	X	X	X	1	X
4	X	X	X	14	X
17	X	X	X	3	X
6	X	X	X	16	X
19	X	-	X	5	X
8	X	X	X	23	-
22	-	-	-	7	X
10	X	X	X	20	-
21	-*	-*	-*	9	X
12	-	-	-	11	X

Table C.7.: Success of different movements within the evaluation result on real hardware and the use of the Youbot and Stochastic inverse kinematic and cubic spline. A cross means that the Youbot was able to move to the specified pose. -* means that the manipulator start moving but does not finish the sub task.

C.2.4. Stochastic inverse kinematic and parabolic blend

Pick ball				Place ball		Comment
Position	p1	p2	p3	Position	p4	
0	X	X	X	15	-	
2	X	X	X	18	X	
13	-*	-	-	1	-	Unknown software failure
4	X	X	X	14	-	
17	X	X	X	3	-	
6	X	X	X	16	-	
19	X	-	X	5	X	
8	X	X	X	23	-	
22	-	-	-	7	X	
10	X	X	X	20	-	
21	-*	-	-	9	-	Unknown software failure
12	-	-	-	11	-	

Table C.8.: Success of different movements within the evaluation result on real hardware and the use of the Youbot and Stochastic inverse kinematic and parabolic blend. A cross means that the Youbot was able to move to the specified pose. -* means that the manipulator start moving but does not finish the sub task.

Bibliography

[1] A. Gfrerrer, "Kinematik und robotik," 2008.
(Cited on pages xiii, 5, and 6.)

[2] R. Liu, P. Serré, and J.-F. Rameau, "A tool to check mobility under parameter variations in over-constrained mechanisms," *Mechanism and Machine Theory*, vol. 69, pp. 44–61, 2013.
(Cited on pages xiii and 9.)

[3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *Proceedings of IEEE Aerospace Conference*, 1999.
(Cited on page 1.)

[4] Dominick Vanthienen, Tinne De Laet, Wilm Decre, Ruben Smits, Markus Klotzbücher, Koen Buys, Steven Bellens, Luca Gherardi, Herman Bruyninckx and Joris De Schutter, "itasc as a unified framework for task specification, control, and coordination, demonstrated on the pr2," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (San Francisco, CA), Sept. 2011.
(Cited on page 1.)

[5] M. C. Sachin Chitta, E. Gill Jones and K. Hsiao, "Mobille manipulation in unstructured environments," *IEEE Robotics and automation magazine*, pp. 58–71, June 2012.
(Cited on pages 1 and 17.)

[6] C. Mühlbacher, G. Steinbauer, M. Reip, and S. Gspandl, "Improving the ros arm navigation stack by using stochastic inverse kinematics," in *Proceedings of the Austrian Robotics Workshop 13*, 2013.
(Cited on pages 2 and 30.)

[7] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower pair mechanisms based on matrices," 1955.
(Cited on page 6.)

[8] J. R. Peters, *Machine learning of motor skills for robotics*.
PhD thesis, University of Southern California, 2007.
(Cited on page 11.)

[9] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementations*.
MIT press, 2005.
(Cited on pages 12 and 13.)

[10] L. Wang and B. Ravani, "Recursive computations of kinematic and dynamic equations for mechanical manipulators," *Robotics and Automation, IEEE Journal of*, vol. 1, no. 3, pp. 124–131, 1985.
(Cited on page 12.)

[11] P. Baerlocher and R. Boulic, "An inverse kinematics architecture enforcing an arbitrary number of strict priority levels," *The visual computer*, vol. 20, no. 6, pp. 402–417, 2004.
(Cited on page 12.)

[12] R. Manseur and K. L. Doty, "A complete kinematic analysis of four-revolute-axis robot manipulators," *Mechanism and Machine Theory*, vol. 27, no. 5, pp. 575 – 586, 1992.
(Cited on page 13.)

[13] R. Manseur and K. L. Doty, "Fast inverse kinematics of five-revolute-axis robot manipulators," *Mechanism and Machine Theory*, vol. 27, no. 5, pp. 587 – 597, 1992.
(Cited on page 13.)

[14] J. M. Hollerbach and G. Sahar, "Wrist-partitioned, inverse kinematic accelerations and manipulator dynamics," *The International journal of robotics research*, vol. 2, no. 4, pp. 61–76, 1983.
(Cited on page 13.)

[15] S. Elgazzar, "Efficient kinematic transformations for the puma 560 robot," *Robotics and Automation, IEEE Journal of*, vol. 1, no. 3, pp. 142–151, 1985.
(Cited on page 13.)

[16] J. M. Hollerbach, "Optimum kinematic design for a seven degree of freedom manipulator," in *Robotics Research: The Second International Symposium*, pp. 215–222, Cambridge: MIT Press, 1985.
(Cited on page 13.)

[17] R. Manseur and K. L. Doty, "A fast algorithm for inverse kinematic analysis of robot manipulators," *The International journal of robotics research*, vol. 7, no. 3, pp. 52–63, 1988.
(Cited on pages 13 and 18.)

[18] M. Raghavan and B. Roth, "Inverse kinematics of the general 6r manipulator and related linkages," *Journal of Mechanical Design*, vol. 115, p. 502, 1993.
(Cited on pages 13 and 18.)

[19] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 72–89, 1986.
(Cited on pages 13, 14, 15, 25, and 26.)

[20] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi, "Minimum cost trajectory planning for industrial robots," *European Journal of Mechanics-A/Solids*, vol. 23, no. 4, pp. 703–715, 2004.
(Cited on page 13.)

[21] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*, pp. 449–464, Springer, 2009.
(Cited on pages 13, 22, and 26.)

[22] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
(Cited on pages 13, 19, and 26.)

[23] J. H. Reif, "Complexity of the movers problem and generalizations," in *IEEE Symp. on Foundat. of Comp. Sci.*, pp. 421–427, 1979.
(Cited on page 13.)

[24] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.
(Cited on pages 14 and 21.)

[25] J.-C. L. David Hsu and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry and Applications*, vol. 9, pp. 495–512, 1999.
(Cited on pages 14 and 22.)

[26] J. J. Kuffner Jr and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, pp. 995–1001, IEEE, 2000.
(Cited on pages 17 and 21.)

[27] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 618 –624, may 2009.
(Cited on page 17.)

[28] M. Wenz, *Automatische Konfiguration der Bewegungssteuerung von Industrierobotern*.
PhD thesis, Universitt Fridericiana zu Karlsruhe, 2008.
(Cited on pages 17, 18, 23, 27, and 29.)

[29] D. Tolani, A. Goswami, and N. I. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs," *Graphical models*, vol. 62, no. 5, pp. 353–388, 2000.

(Cited on page 18.)

[30] K. Low and R. Dubey, "A comparative study of generalized coordinates for solving the inverse-kinematics problem of a 6r robot manipulator," *The International journal of robotics research*, vol. 5, no. 4, pp. 69–88, 1986.
(Cited on page 18.)

[31] I.-M. Chen and Y. Gao, "Closed-form inverse kinematics solver for reconfigurable robots," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3, pp. 2395–2400, IEEE, 2001.
(Cited on page 18.)

[32] C. Wampler and A. Morgan, "Solving the 6¡ i¿ r¡/i¿ inverse position problem using a generic-case solution methodology," *Mechanism and Machine Theory*, vol. 26, no. 1, pp. 91–106, 1991.
(Cited on page 18.)

[33] R. Diankov, *Automated Construction of Robotic Manipulation Programs*.
PhD thesis, Carnegie Mellon University, Robotics Institute, 2010.
(Cited on pages 18, 28, and 29.)

[34] D. Manocha and Y. Zhu, "A fast algorithm and system for the inverse kinematics of general serial manipulators," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3348–3353, IEEE, 1994.
(Cited on page 18.)

[35] D. Manocha and J. F. Canny, "Real time inverse kinematics for general 6r manipulators," in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pp. 383–389, IEEE, 1992.
(Cited on page 18.)

[36] A. Goldenberg, B. Benhabib, and R. Fenton, "A complete generalized solution to the inverse kinematics of robots," *Robotics and Automation, IEEE Journal of*, vol. 1, no. 1, pp. 14–20, 1985.
(Cited on page 19.)

[37] L. Kelmar and P. K. Khosla, "Automatic generation of kinematics for a reconfigurable modular manipulator system," in *IEEE International Conference on Robotics and Automation*, (Philadelphia, PA), pp. 663–668, Apr. 1988.
(Cited on page 19.)

[38] G. Tevatia and S. Schaal, "Inverse kinematics for humanoid robots," in *IEEE International Conference on Robotics and Automation*, (San Francisco, CA), pp. 294–299, Apr. 2000.
(Cited on pages 19 and 20.)

[39] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *The International Journal of Robotics Research*, vol. 4 (3), pp. 109–116, Sept. 1985.
(Cited on page 19.)

[40] J. M. Hollerbach and K. C. Suh, "Redundancy resolution of manipulators through torque optimization," *IEEE JOURNAL OF ROBOTICS AND AUTOMATION*, vol. RA-3(4), pp. 308–316, Aug. 1987.
(Cited on page 19.)

[41] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Tramsactions on systems, man, and cybernetics*, pp. 93–101, Jan. 1989.
(Cited on page 19.)

[42] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, 2004.
(Cited on page 19.)

[43] Y. T. Tsai and D. E. Orin, "A strictly convergent real-time solution for inverse kinematics of robot manipulators," *Journal of Robotic Systems*, vol. 4: 21, pp. 477–501, Mar. 1987.
(Cited on page 19.)

[44] L.-C. Wang and C. C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 4, pp. 489–499, 1991.

(Cited on page 19.)
[45] J. K. Parker, A. R. Khoogar, and D. E. Goldberg, "Inverse kinematics of redundant robots using genetic algorithms," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pp. 271–276, IEEE, 1989.
(Cited on pages 19 and 30.)
[46] H. Cheng and K. C. Gupta, "A study of robot inverse kinematics based upon the solution of differential equations," *Journal of Robotic Systems*, vol. 8(2), pp. 159–175, Nov. 1991.
(Cited on page 19.)
[47] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 1, pp. 298–303, IEEE, 2001.
(Cited on page 20.)
[48] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 522–531, ACM, 2004.
(Cited on page 20.)
[49] A. A. T. M. Eimei Oyama, Nak Young Chong and S. Tachi, "Inverse kinematics learning by modular architecture neural networks with performance prediction networks," in *IEEE International Conference on Robotics and Automation*, (Seoul, Korea), pp. 21–26, May 2001.
(Cited on page 20.)
[50] A. Guez and Z. Ahmad, "Solution to the inverse kinematics problem in robotics by neural networks," in *Neural Networks, 1988., IEEE International Conference on*, pp. 617–624, IEEE, 1988.
(Cited on page 20.)
[51] A. B. A. Ramdane-Cherif, B. Daachi and N. Levy, "Kinematic inversion," in *Intl. Conference on Intelligent Robots and Systems*, (Lausanne, Switzerland), pp. 1904–1909, Oct. 2002.
(Cited on page 20.)
[52] M. K. I. A. Sucan and S. Chitta, "Combining planning techniques for manipulation using realtime perception," in *Conf. Robotics and Automation*, p. 28952901, 2010.
(Cited on page 20.)
[53] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, pp. 628 – 649, 1991.
(Cited on page 20.)
[54] K. Gupta, "Practical global motion planning for many degrees of freedom: a novel approach within sequential framework," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 2038 –2043 vol.3, may 1994.
(Cited on page 20.)
[55] S. C. B. Cohen and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *International Conference on Robotics and Automation*, 2010.
(Cited on page 21.)
[56] B. Cohen, G. Subramanian, S. Chitta, and M. Likhachev, "Planning for manipulation with adaptive motion primitives," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5478 –5485, may 2011.
(Cited on page 21.)
[57] J. M. Ahuactzin, E.-G. Talbi, P. Bessiere, and E. Mazer, "Using genetic algorithms for robot motion planning," in *10th Europ. Conf. Artific. Intelligence*, (London), pp. 671–675, 1992.
(Cited on page 21.)
[58] J. A. B. Nathan Ratliff, Matt Zucker and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE lntemational Conference on Robotics and Automation*, (Kobe, Japan), pp. 489–494, May 2009.
(Cited on page 21.)
[59] E. T. P. P. M. Kalakrishnan, S. Chitta and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE lntemational Conference on Robotics and Automation*, (Shanghai, China), p. 45694574, May 2011.
(Cited on page 21.)
[60] S. M. LaValle, "Rapidly-exploring random trees a ew tool for path planning," 1998.

(Cited on page 21.)

[61] P. Isto, "A two-level search algorithm for motion planning," in *International Conference on Robotics and Automation*, (Albuquerque, New Mexico), pp. 2025–2031, 1997.
(Cited on page 21.)

[62] J. M. A. Emmanuel Mazar and P. Bessière, "The ariadne's clew algorithm," *Journal of Artificial Inteligence Resarch*, vol. 9, pp. 295 – 316, 1998.
(Cited on page 21.)

[63] B. O. B. D. L. K. J. C. Amato, N. M. and D. Vallejo, "Obprm: An obstacle-based prm for 3d workspaces," in *In Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, p. 155168, 1998.
(Cited on page 21.)

[64] B. Glavina, "Solving findpath by combination of goal-directed and randomized search," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pp. 1718 –1723 vol.3, may 1990.
(Cited on page 21.)

[65] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *IEEE lntemational Conference on Robotics and Automation*, (Minneapolis, Minnesota), pp. 113–120, Apr. 1996.
(Cited on page 21.)

[66] J.-C. L. R. M. D. Hsu, L. E. Kavraki and S. . Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *In Robotics: The Algorithmic Perspective*, (Wellesley, MA), p. 141154, 1998.
(Cited on page 21.)

[67] T. Horsch, F. Schwarz, and H. Tolle, "Motion planning with many degrees of freedom-random reflections at c-space obstacles," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3318 –3323 vol.4, may 1994.
(Cited on page 21.)

[68] J. Laumond and T. Simeon, "Notes on visibility roadmaps and path planning," in *In Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, pp. 317–328, 2000.
(Cited on page 21.)

[69] V. Boor, M. Overmars, and A. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, pp. 1018 –1023 vol.2, 1999.
(Cited on page 21.)

[70] S. Wilmarth, N. Amato, and P. Stiller, "Maprm: a probabilistic roadmap planner with sampling on the medial axis of the free space," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, pp. 1024 –1031 vol.2, 1999.
(Cited on page 21.)

[71] C. Holleman and L. E. Kavraki, "A framework for using the workspace medial axis in prm planners," in *IEEE lntemational Conference on Robotics and Automation*, (San Francisco, CA), pp. 1408–1413, Apr. 2000.
(Cited on page 21.)

[72] C. L. Nielsen and L. E. Kavraki, "A two level fuzzy prm for manipulation planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1716–1721, 2000.
(Cited on page 21.)

[73] M. H. Overmars, "A random approach to motion planning," *RUU-CS*, no. 92-32, 1992.
(Cited on page 21.)

[74] M. H. Overmars and P. Svestka, "A probablisitic learning approach to motion planning," *UU-CS*, no. 1994-03, 1994.
(Cited on page 21.)

[75] J.-C. L. R. M. T.-Y. L. Jèrôme Barraquand, Lydia Kavraki and P. Raghavan, "A random sampling scheme for path planning," *The International Journal of Robotics Research*, pp. 759–774, 1997.
(Cited on page 21.)

[76] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration for fast path planning," in

Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, pp. 2138 –2145 vol.3, may 1994.
(Cited on page 21.)

[77] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 521–528, IEEE, 2000.
(Cited on page 22.)

[78] R. Bohlin and L. Kavraki, "A lazy probabilistic roadmap planner for single query path planning," *International Journal of Robotics Research*, 2000.
(Cited on page 22.)

[79] R. Bohlin and L. E. Kavraki, *Handbook on Randomized Computing*, pp. 221–249.
Kluwer Academic Publishers, 2001.
(Cited on page 22.)

[80] D. Costantinescu and E. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000.
(Cited on page 23.)

[81] A. Piazzi and A. Visioli, "Global minimum-jerk trajectory planning of robot manipulators," *Industrial Electronics, IEEE Transactions on*, vol. 47, no. 1, pp. 140–149, 2000.
(Cited on page 23.)

[82] S. Macfarlane and E. A. Croft, "Jerk-bounded manipulator trajectory planning: design for real-time applications," *Robotics and Automation, IEEE Transactions on*, vol. 19, no. 1, pp. 42–52, 2003.
(Cited on pages 23 and 26.)

[83] J.-J. Slotine and H. Yang, "Improving the efficiency of time-optimal path-following algorithms," *Robotics and Automation, IEEE Transactions on*, vol. 5, pp. 118 –124, feb 1989.
(Cited on page 23.)

[84] L. Zlajpah, "On time optimal path control of manipulators with bounded joint velocities and torques," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 2, pp. 1572 –1577 vol.2, apr 1996.
(Cited on page 23.)

[85] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.
(Cited on page 23.)

[86] K. Shin and N. McKay, "A dynamic programming approach to trajectory planning of robotic manipulators," *Automatic Control, IEEE Transactions on*, vol. 31, no. 6, pp. 491–500, 1986.
(Cited on page 23.)

[87] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 2, pp. 115–123, 1987.
(Cited on page 24.)

[88] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *Automatic Control, IEEE Transactions on*, vol. 30, pp. 531 – 541, jun 1985.
(Cited on page 24.)

[89] T. Kunz and M. Stilman, "Turning paths into trajectories using parabolic blends," 2011.
(Cited on page 24.)

[90] T. Kunz and M. Stilman, "Time-optimal path following with bounded joint accelerations and velocities," 2011.
(Cited on pages 24 and 35.)

[91] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Proceedings of the 2012 Robotics: Science and Systems Conference*, vol. 8, pp. 09–13, 2012.
(Cited on pages 24 and 35.)

[92] I. Sucan, M. Moll, and L. Kavraki, "The open motion planning library," *Robotics Automation Magazine, IEEE*, vol. 19, pp. 72 –82, dec. 2012.
(Cited on pages 25, 34, and 67.)

[93] H. Bruyninckx, "Open robot control software: the orocos project," in *International Conference on Robotics 8 Automation*, (Seoul, Korea), pp. 2523–2528, May 2001.

(Cited on page 27.)

[94] J. Jesse, "A kinematic model for the icub," tech. rep., Department of computer science - EPFL, 2009. (Cited on pages 27 and 28.)

[95] K. Shoemaker, "Animating rotation with quaternion curves," *ACM*, vol. 19, pp. 245–254, Nov. 1985. (Cited on page 30.)

[96] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997. (Cited on page 30.)

[97] N. Koenig and J. Hsu, "The many faces of simulation: Use cases for a general purpose simulator," in *ICRA Workshop on Developments of Simulation Tools for Robotics & Biomechanics*, 2013. (Cited on page 55.)