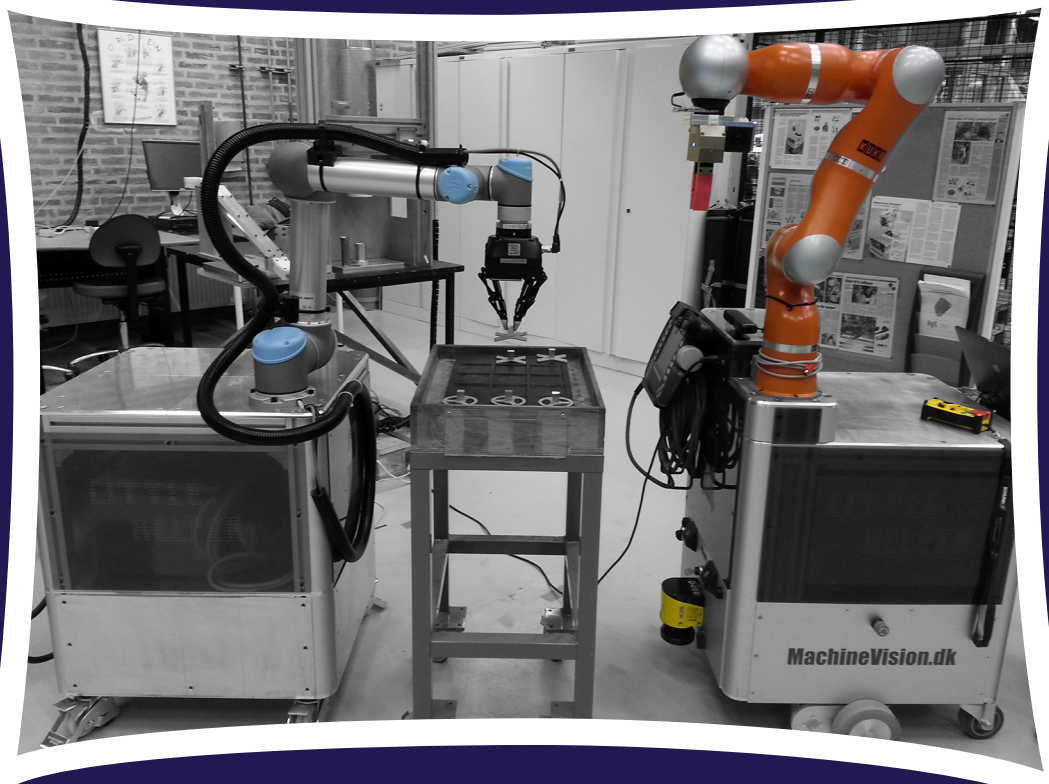
Motion Planning in a Skill-Based System



Rune Etzerodt Morten Palmelund-Jensen Casper Abildgaard Pedersen

Master's Thesis in Manufacturing Technology - 2014
Department of Mechanical and Manufacturing Engineering

AALBORG UNIVERSITY
STUDENT REPORT

Department of Mechanical and Manufacturing Engineering. Manufacturing Technology.
Fibigerstræde 16
DK - 9220 Aalborg Ø
Phone: +45 99 40 71 17
info@m-tech.aau.dk
http://www.m-tech.aau.dk

Title:
Motion Planning in
a Skill-Based System

Project Period:
Spring semester, 2014.

Project group:
VT4 3.014

Group members:

Rune Etzerodt

Morten Palmelund-Jensen

Casper Abildgaard Pedersen

Supervisor:
Professor Ole Madsen
Co-supervisor:
Post doc Dimitris Chrysostomou
Number of Pages:
108
Number of Appendices:
10
Completed:
June 3th 2014

Synopsis:

This project concerns the incorporation of motion planning in a Skill-Based System (SBS). The vision with SBS is outlined first, whereupon the structure of it is explained. For this structure a long-term and a short-term concept proposal is made for incorporation of motion planning. Among three, the motion planning software MoveIt is chosen for implementation. A motion planning study is conducted for the sampling-based algorithms of the Open Motion Planning Library, which MoveIt utilises. A benchmark is made to select applicable motion planners in coherence with the theory examined in the motion planning study. Motion planning is implemented in the SBS as proof-of-concept for the short-term proposal. Furthermore, motion planning with multiple manipulators and motion planning in a sensor-based environment is implemented and tested.

By signing this document, each of the group members confirms that everyone has participated equally and thereby the group collectively liable for the content of this paper.

Preface

This project is conducted at the 4^{th} semester at the master programme of Manufacturing Technology at Aalborg University, Department of Production and Mechanical Engineering, during the spring semester 2014.

The report is divided into 12 chapters, 10 appendixes, and a Danish summary (Resumé). Each chapter begins with a short recap, in *italic*, explaining the content of the chapter. Figures, formulas, and tables are numbered by an index number, corresponding to the respective chapter or appendix number. For example is figure 2.1 the first figure in chapter 2.

Sources are specified as follows: [LaValle, 2006, p. 8]. Where [LaValle] is a reference for the author in the source, [2006] is the year of publication, and [p. 8] is the page number in the source, where the page number is optional. If the source is specified before a punctuation mark, then the reference is for the specific sentence. If the source is specified after a punctuation mark, then the reference is for the whole paragraph.

The report is divided into the following parts; *Introduction, Main Report, Additional Work, and Conclusion*, where chapter 2 in *Introduction* is only mandatory if the reader is not familiar with the related work conducted by the project group, during the 3^{rd} semester, and the *Robotics and Automation Group* at *Aalborg University*. Supplementary written material is to be found in the *Appendix*.

In addition to the report, an appendix CD is enclosed. The CD contains; videos, source code, the 3^{rd} semester report, and the report of this project. When referring to contents on the CD, a footnote is used to describe the path to the content.

The project group would like to thank the Robotics and Automation Group; especially Ph.D student Rasmus Skovgaard Andersen, Ph.D student Jens Skov Damgaard, Ph.D student Casper Schou, Post doc Dimitris Chrysostomou and Professor Ole Madsen at the Department of Mechanical and Manufacturing Engineering at Aalborg University.

Aalborg University, June 03, 2014

Resumé

Dette projekt omhandler design og implementering af baneplanlægningsløsninger for mobile manipulatorer i et Skill-based system (SBS), med fokus på artikulerede manipulatorer.

Arbejdet er funderet i det SBS, som er udviklet, og eksisterer i flere udgaver, på Aalborg Universitet. SBS bruges som en software struktur for den mobile manipulator familie "Little Helper". Fokus har været på Little Helper 4, som er den nyeste mobile manipulator i familien. Little Helper 4 består af en Universal Robots UR5 manipulator, en Robotiq RQ3 griber og en mobil platform. SBS ligger rammerne for udviklingen af to konceptuelle løsningsforslag, til inkorporering af baneplanlægning for mobile manipulatorer; et langsigtet og et kortsigtet koncept forslag. Det langsigtede løsningsforslag omhandler koncepter der ikke nødvendigvis kan implementeres i et eksisterende SBS, men det beskriver derimod hvilke ultimative mål der kan arbejdes hen imod for at forbedre baneplanlægning. Mere præcist omhandler det langsigtede løsningsforslag baneplanlægning, samt generering og opretholdelse af et verdensmiljø til brug i baneplanlægningen. Derudover er et forslag til inkorporering af læring præsenteret, for at gøre systemet mere intelligent og nedsætte brugerkravene. Det kortsigtede løsningsforslag, tager udgangspunkt i det langsigtede løsningsforslag, men det er tiltænkt at skulle kunne blive implementeret i det eksisterende SBS. Dette omhandler et forslag til en software struktur, som gør det muligt at baneplanlægge for artikulerede manipulatorer. Der er ikke blevet fokuseret på baneplanlægning for mobile platforme, i det kortsigtede løsningsforslag.

Som baggrund for den praktiske implementering af baneplanlægning, er en kort software analyse udført. V-REP, RobWork og MoveIt er analyseret, og på baggrund af det kortsigtede løsningsforslag, er MoveIt valgt til at blive benyttet i implementeringen. Baneplanlægningsprogrammet MoveIt bruger veldefinerede interfaces, hvilket gør det muligt at anvende brugerdefinerede baneplanlægningsalgoritmer og kinematiske løsere.

I Moveit er der implementeret et baneplanlægningsbibliotek (OMPL), som indeholder en række sample baseret baneplanlægningsalgoritmer. Algoritmerne fra dette bibliotek er blevet undersøgt gennem et litteraturstudie, med fokus på de anvendte egenskaber for algoritmerne. Efterfølgende er algoritmerne blevet sammenlignet gennem MoveIt, for at underbygge konklusionerne fra litteraturstudiet. På baggrund

af dette er baneplanlægningsalgoritmen RRTConnect valgt.

På baggrund af det kortsigtede løsningsforslag og det valgte baneplanlægningsprogram, MoveIt, er baneplanlægning blevet implementeret i Little Helper 4 SBS, som proof of concept. Dette inkludere ændringer i SBS, driveren for UR5 manipulatoren og kommunikationen imellem. Funktionaliteten af implementeringen er verificeret gennem et kvalitativt forsøg.

Som supplement til implementering af baneplanlægning i et SBS, er baneplanlægning blevet anvendt på baggrund af sensor input og til styring af to manipulatorer. Dette er gjort i to individuelle forsøg. Sensor input er anvendt til etablering af verdensmiljø og efterfølgende baneplanlægning på baggrund af det etablerede miljø. Derudover er baneplanlægning udført for både Little Helper 3 og Little Helper 4, med overlappende arbejdsområder. Dette arbejde har inkluderet oprettelse af en fælles model for begge mobile manipulatorer og håndtering af kommunikationen til begge.

Contents

Part I

Introduction

Chapter 1

Introduction

Production of today experience a paradigm-shift from mass production to mass customisation. This creates a need for flexible and transformable production, as shown in figure 1.1. If the production in addition shall be competitive, compared to countries with cheap labour, then automation is a need. A combination of these needs calls for flexible, transformable, and cost-effective automation. [Hvilshøj et al., 2012]

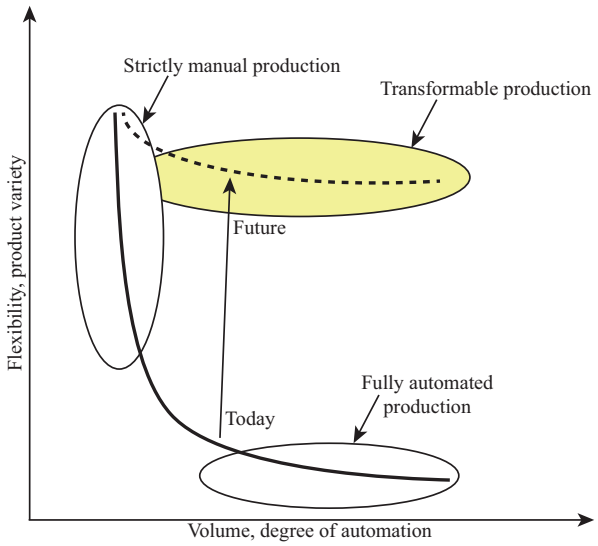


Figure 1.1: Illustrates the vision for production systems in the future. [Hvilshøj et al., 2012]

Industrial manipulators are at present inflexible, by being fixed and dedicated to a specific task. To cope with inflexibility, manipulators must have the ability to move around and perform various tasks, without time demanding programming of the manipulator. Incorporation of mobility is however a challenging

task in unstructured and changing environments. Another challenge, to increase the flexibility, is the classical manipulator-centric programming, since programs are typically created from scratch, dedicated to a specific environment. This calls for a paradigm-shift within programming of manipulators to a task-level programming paradigm. [Bøgh et al., 2012]

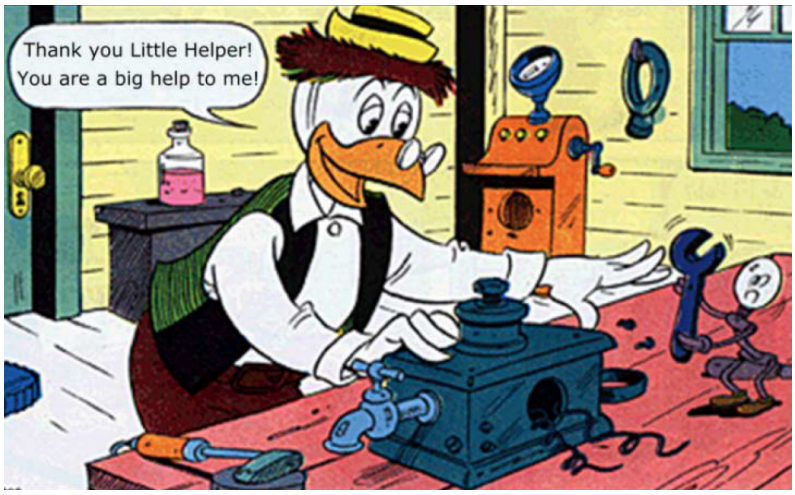


Figure 1.2: The goal for little helper. [Hvilshøj et al., 2012]

The vision for industrial manipulators of the future can be described on the basis of figure 1.2. *A robot, which can assist and perform tasks in a production system with minimal human intervention.* There are scientific and engineering challenges, which must be solved in order to succeed, in creating a collaborative industrial mobile manipulator [Angerer et al., 2012]. The challenges involves, among others, the following:

- Safety
- Reconfigurability
- Knowledge integration

Safety is an important concern, regarding collaborative mobile manipulators, because harm on humans and equipment must be avoided. Inclusion of vision sensors, which updates a representation of the environment, is a way to make the manipulators aware of the humans and equipment. Mounting of the sensors is a vital prospect, because it has to be ensured that all risks for a collision is captured, within the sensors field-of-view. [Angerer et al., 2012]

Reconfigurability is the ability to adapt to a dynamic production system. The ability to adapt, for a collaborative industrial mobile manipulator, is important, because manufacturing tasks can be changed frequently. [Angerer et al., 2012]

Knowledge integration is the collaborative industrial manipulators ability to collaborate with humans and interacting with the environment, with minimal human intervention. It is necessary that the manipulator knows what to do, without demanding hourly programming to obtain a competitive productivity. [Angerer et al., 2012]

1.1 Abstraction of Motion Planning

An abstraction on *motion planning*, and the definitions it entails, is provided in its essential form, since these will be utilised throughout the report.

A computationally representation of the "world" is the basis for motion planning, for determining feasible and infeasible configurations of a manipulator. The "world", is defined as the *environment* throughout the report. A *configuration space* is defined as all possible transformations the manipulator can be in. The configuration space can be formulated in different state spaces, such as joint state or Cartesian state. The *obstacle region* C_{obs} and the *free region* C_{free} are both derived from the configuration space. A basic motion planning problem is shown in figure 1.3. The aim is to get from an *initial state* (A) to a *goal state* (B) in C_{free}. Together they form a *query*. If a result exists, then a *path* can be created in C_{free}, as illustrated by the arrow, from point A to point B. [LaValle, 2006, p. 159]

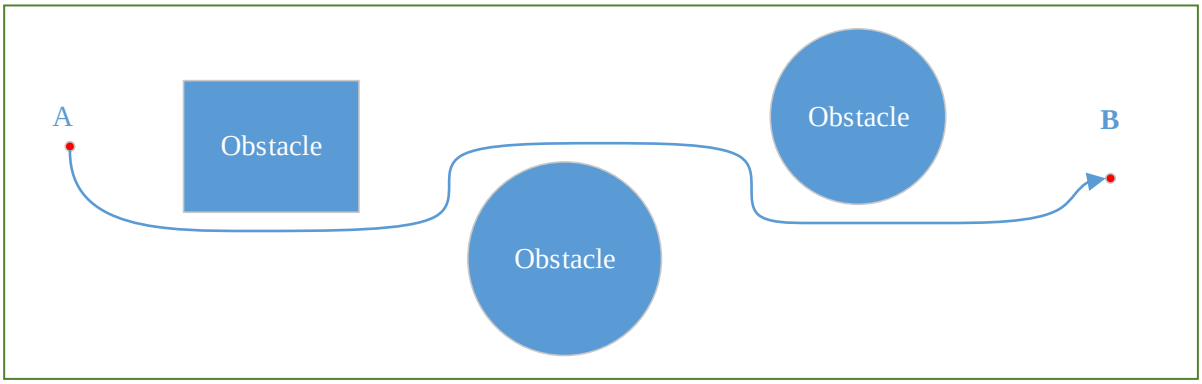


Figure 1.3: A basic motion planning problem, where the goal is to move from point A to point B without colliding with the obstacles in the configuration space.

1.2 This Project

One way for a mobile manipulator to obtain the ability to work in an unstructured environment, is by utilisation of motion planning. Motion planning is the ability to plan (and execute) a collision free path. This eases the programming of a mobile manipulator, allowing for lower set-up times. This conviction, regarding motion planning, is the basis for the work described throughout this project. This project will, from the point of view of a mobile manipulator, focus on articulated manipulation. The project is going to concern design, development, and implementation of motion planning in a framework for mobile manipulators. This will include conceptual design proposals for the mobile manipulator software framework of today and of the future.

1.3 Related Work

The focus in this section is regarding motion planning for the articulated manipulation part of mobile manipulators. The main difference between the articulated manipulation of a mobile manipulator and a classical fixed manipulator is the possible unstructured environment and human collaboration.

A common frame of reference is created to compare different mobile manipulators and their capabilities. This is followed by a description of motion planning, which is a common capability for each of the investigated mobile manipulators. This is described to establish a baseline, regarding the motion planning capabilities of mobile manipulators.

1.3.1 Frame of Reference

Different types of mobile manipulators are in the following compared, to determine their main capabilities. The justification of the comparison, of different types of mobile manipulators, is elaborated in the following:

Field of Research

The field of research regarding mobile manipulators can be divided into two main approaches; one concerning specific designed mobile manipulators (PR2 [Oyama et al., 2009], Justin [Ott et al., 2006]) and one concerning mobile manipulators created of commercial of the shelf (COTS) parts (The Industrial Manipulation Platform [Hermann et al., 2011], Abby [Venator et al., 2013], TUM-Rosie [Kunze et al., 2011]). The specific designed manipulators have the advantage that the developer knows the exact details of the system [Zacharias et al., 2010], which can be used during modelling and low-level control of the

hardware. The COTS manipulators concerns mainly application and integration of hardware [Venator et al., 2013] [Hermann et al., 2011], because exact data and accessibility to low-level control may not be an option.

Field of Application

Two main fields of application for mobile manipulators are home/office environments and industrial environments. The two fields of application have different requirements regarding manipulation tolerances[Hvilshøj et al., 2012]. There have been no distinction between the two fields of application because the fundamentals of the capabilities are similar, for example motion planning in a changing environment and collaboration with humans.

An extract of the mobile manipulators, within the groups described above, are shown in table 1.1.

1.3.2 Elaboration of Main Capabilities

The topics of the *Main capabilities* column, in table 1.1, are elaborated:

- **Vision** is used to create models of the environment or locate objects to be manipulated. [Venator et al., 2013]
- **Motion Planning** is the movement of the manipulator without collision. Some mobile manipulators are cable of conducting motion planning continuously (on-the-fly), where the motion planning set-up can adopt to a changing environment. [Srinivasa et al., 2012]
- **Grasp Planning** is the ability to grasp objects, often identified and positioned by use of vision. The grasping approach is determined from simulations or a database look-up for the concerned object. [Hermann et al., 2011]
- **Low level control of manipulators** is for example is changing and optimisation of control parameters (PID) for the individual actuators, to obtain the desired response of the system. [Ott et al., 2006]
- **"Skill-based" executions of tasks** which is the execution of a collection of device primitives based on objects and not 3D coordinates. This can for example be to pick up a object, located by the vision system [Hermann et al., 2011]. The skill-based execution of tasks is elaborated in section 2.1.1
- **Execution of bimanual tasks** which are tasks requiring use of two manipulators/hands. This can for example be unscrewing of a bottle cap, where one hand are fixing the bottle and one are unscrewing the cap, or assembly of a bolt and nut. [Zacharias et al., 2010]

Name	Manipulator	Area of application	Main capabilities	Year/Organisation
PR2 [Willow Garage, 2012], [Oyama et al., 2009]	Two custom 7 dof manipulators, 1 dof end effectors and a 4 dof torso/-head	Human environment/indoor household	Navigation, motion planning, vision, low level control of manipulators	2009, Willow Garage
HERB 2.0 [Srinivasa et al., 2012]	Two Barrett 7-dof WAM and Barrett hands (COTS)	Human environment, manipulation of table top objects	Motion planning (on the fly), "Skill-based" executions of tasks, human collaboration	2012, Carnegie Mellon University, Pittsburgh
The Industrial Manipulation Platform (IMP) [Hermann et al., 2011]	Two 7-dof KUKA LWR manipulators (COTS) and two 13-dof DLR Four-Finger-Hands	Industry	Grasp and motion planning (on the fly), vision, "Skill-based" executions of tasks	2010, FZI Karlsruhe
ABBY [Venator et al., 2013]	6-dof ABB IRB 120 manipulator and one pneumatic parallel gripper (COTS)	Industry	Motion planning, vision	20013, Case Western Reserve University
(Rollin') Justin [Ott et al., 2006], [Zacharias et al., 2010]	Two 7-dof DLR (KUKA LWR) manipulators and two DLR-Hand-II grippers, with 43-dof of the upper body	Data not available	Vision, motion planning, execution of bimanual tasks	2006, DLR
TUM-Rosie [Kunze et al., 2011]	Two 7-dof KUKA LWR manipulators (COTS) and two DLR-HIT hands	Human environment/indoor household	Vision, motion planning, grasp planning	2011, TUM Technische Universität München

Table 1.1: Related work regarding mobile manipulators. The table concerns articulated manipulation and not the wheeled base. The PR2 was produced for sale to for example research institutions, whereas the others are design and build as "one of a kind".

1.3.3 Motion Planning for Mobile Manipulators

Motion planning, described in the following, is focused on the capabilities of the mobile manipulators from table 1.1.

Motion planning can be conducted for individual manipulators [Venator et al., 2013], two manipulators at once [Srinivasa et al., 2012], and for both manipulators and locomotive platforms at once [Hermann et al., 2011]. The goal for motion planning can for example be specified by a user or determined by use of database look-up. Database look-up can be based on identified objects, and the most suited grasping position can be determined, with the motion planning for the manipulator in mind [Hermann et al., 2011]. Manipulation of objects can be combined with a motion planner, capable of specifying plans, which includes rearranging of the environment, to be able to reach a desired goal. The environment can be based on a model of the manipulator and obstacles, along with sensor input [Venator et al., 2013] or consist of a full 3D model based on sensor input. [Srinivasa et al., 2012]

Motion planners can obey start and goal constraints, for example an end-effector position. In addition to this, process constraints for the path can be obeyed [Srinivasa et al., 2012] [Hermann et al., 2011]. This can for example be keeping a cup of water level to avoid spilling or opening of a door where the end effector have to follow a constrained path.

To reduce the initial waiting time after initiating a planning request, mobile manipulators can have the ability to execute an initial plan, even before the motion planner knows if the entire path is feasible. The planning proceeds, while the initial plan is executed, possible correcting the path [Hermann et al., 2011]. A changing environment can be accounted for during execution of paths, by replanning the path. This can be advantageous, when working in highly unstructured environments or working in close cooperation with humans. [Srinivasa et al., 2012] [Hermann et al., 2011]

The total planning time can be reduced by reuse of former created paths. This can be efficient both in static environments, but also in changing environments. This is because not all parts of the plan have to be created all over again. [Hermann et al., 2011]

Chapter 2

The Little Helper Concept

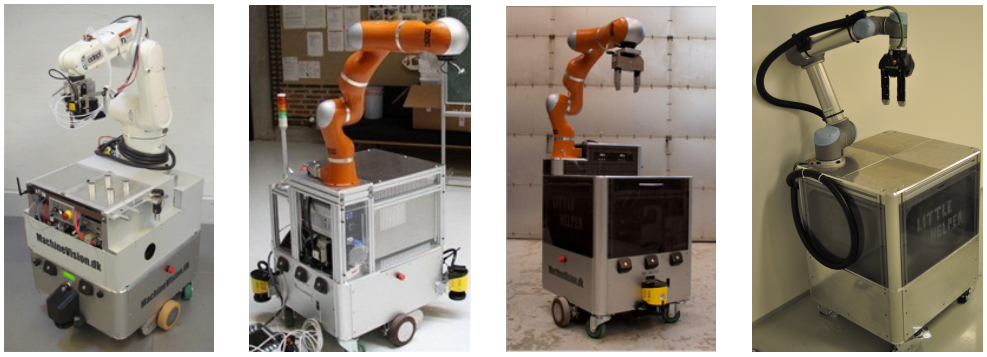
This chapter introduces the autonomous industrial mobile manipulator concept "Little Helper" design and developed at Aalborg University. This includes both the members of the Little Helper family and a presentation of the overall structure of the software framework. The latest member of the Little Helper family, Little Helper 4, is described in more detail, since it is the basis for the work conducted throughout this project.

2.1 The Little Helper Family

At *Aalborg University* the research regarding mobile manipulators are concentrated around a family of mobile manipulators named *Little Helper*. At present the family consist of three different Little Helpers; Little Helper 2, 3, and 4, shown in figure 2.1. The types may differ in type of hardware, but each features an articulated manipulator, a gripper and a mobile platform. The platform is either manual or autonomous. Little Helper 3 and Little Helper 4 can be programmed by use of task-level programming, made in a skill-based system.

2.1.1 Skill-Based System

The skill-based system (SBS) is the core of all Little Helpers. The SBS on each of the different Little Helpers varies at present, but they all share the same general structure. This project is based upon a SBS, thus an introduction to the definitions are here given. The outlining of the SBS is based on [Bøgh et al., 2012].



(a) Little Helper 1. **(b)** Little Helper 2. **(c)** Little Helper 3. **(d)** Little Helper 4.

Figure 2.1: The Little Helper family at Aalborg University. Little Helper 1 does not exist anymore. [Machine Vision, 2013]

An overview of the layered structure of the SBS is given in figure 2.2. The top layer of the SBS is *missions*, which are controlling the order and numbers of *tasks* to be executed. Tasks contains a sequence of *skills*. Each skill consist of a sequence of *motion primitives*. The four layers are described in the following.

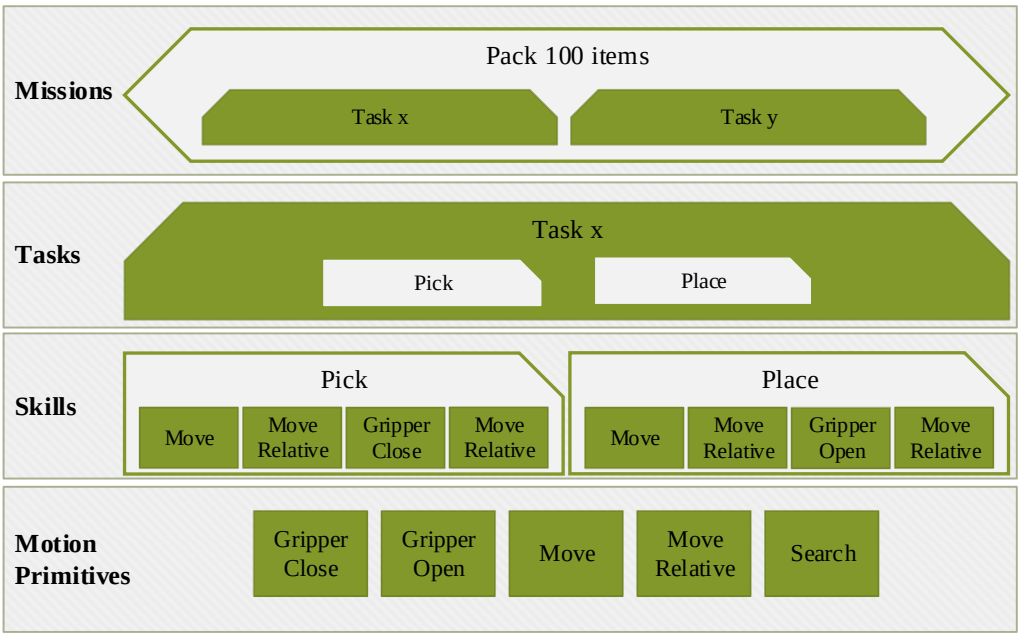


Figure 2.2: The layered structure of the SBS. Inspired by [Carøe et al., 2012].

Motion Primitives

Beginning at the lowest layer in figure 2.2, the motion primitives (also called *device primitives*) are the basic commands of a SBS. They can be seen as the smallest building blocks of the SBS. Motion primitives can either be directly or indirectly executable on the hardware. By indirectly is meant that the motion primitive utilises a proxy, such that standard commands can be translated to hardware specific commands that can be executed directly by the hardware. [VT3-2013, 2013]

A motion primitive is for example the ability to move the tool centre point (TCP) for a manipulator, or open or close the gripper, as seen in the lowest layer of figure 2.2.

Skills

A skill utilises the current world state as input, as well as task specific parameters. Initially the skill checks whether all preconditions are fulfilled, before executing the skill, based on the given inputs. If the preconditions are satisfied, then the skill executes the sequence of motion primitives that are included in the given skill. The sequence of motion primitives alters the state of the world. After the skill is executed, then state variables of the world are compared to predicted variable states, to evaluate whether the task is successfully completed. A graphical representation of a generic skill is shown in figure 2.3. [Pedersen et al., 2013]

An example of the preconditions, post conditions and parameters for a pick skill is shown in table 2.1. The motion primitive contained in a pick skill is shown in figure 2.4.

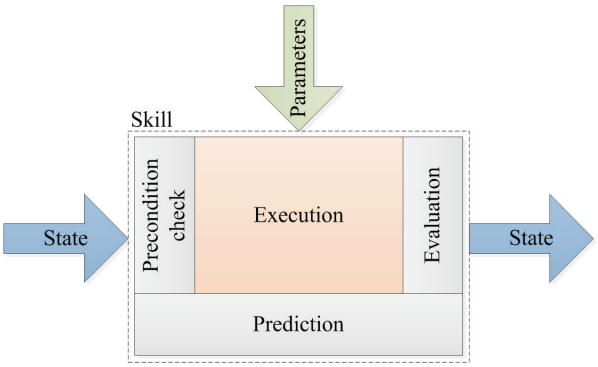


Figure 2.3: Structure of a generic skill. [Bøgh et al., 2012]

Skills are intended to be object-oriented and not based on 3D coordinates, enabling the manipulator to work in an unstructured environment. This means that a pick skill will be defined as "pick <object>" and not "pick <3D coordinates>". This structure enables for teaching of skills, by choosing the object and parameters for the given task.

Topic	**State**
Preconditions	Gripper open?
Parameters:	Approach direction Leave direction Object (location) Object size Grasping force
Postconditions:	Gripper closed? Correct (size) of object?

Table 2.1: Example of preconditions, parameters, and postconditions for a pick skill.

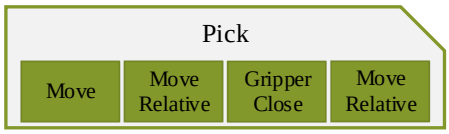


Figure 2.4: Example of the motion primitives a pick skill can contain. Inspired by [Carøe et al., 2012].

A skill in the Little Helper 4 software framework deviates slightly for the general definition. This is described in section 2.2.2.

Tasks

A task is defined as a sequence of instantiated skills that either are programmed in pre-hand or generated by a potential task-level planner [Pedersen et al., 2013]. In general tasks can be seen as a whole process, as for example moving an object from A to B. A task consisting of a pick and a place skill is illustrated in figure 2.5.

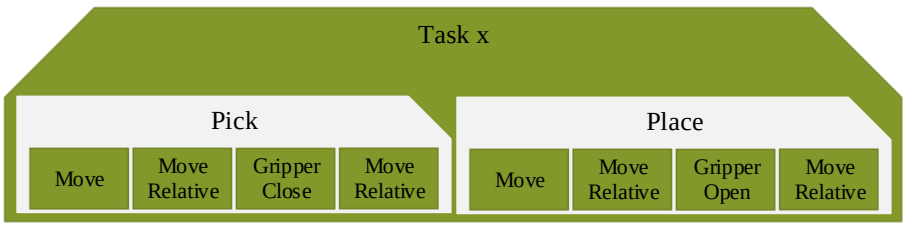


Figure 2.5: Example of the skills a task can contain. Inspired by [Carøe et al., 2012].

Missions

Missions are the controlling agency of the SBS. A mission consist of one or more tasks. The order and number of tasks are defined based on the desired production output. The Enterprise Resource Planning (ERP) software, controlling a whole production, can dictate the order and number. Missions can include more than one Little Helper. [Schou et al., 2013]

2.2 Little Helper 4

Little Helper 4 is the latest addition to the Little Helper family. Little Helper 4 was developed during the 3^{rd} semester by the project group. The design, development, and creation included both the physical hardware set-up for Little Helper 4 and the skill-based Little Helper 4 software framework. The report "Creation of an Autonomous Industrial Mobile Manipulator" [VT3-2013, 2013] describing the work, is appended on the enclosed CD[1].

A brief introduction to the work done during the 3^{rd} semester is given, since Little Helper 4 will be used as the base for practical implementation through this project. The software developed and created during the 3^{rd} semester will in the following be referred to as *the original Little Helper 4 software framework*. The modified software framework, which is described through this report is referred to as *the Little Helper 4 software framework*.

2.2.1 Main Hardware Components of Little Helper 4

The main hardware components of Little Helper 4, are the Universal Robots UR5 manipulator (UR5), the Robotiq 3-finger adaptive gripper (RQ3), and the mechanical platform. Each of the components are shortly described in the following, regarding main functionalities and specifications.

Universal Robots UR5 Manipulator

The Universal Robots UR5 is a 6 degrees of freedom articulated manipulator with a payload of 5.0 kg and a range of 850 mm [Universal Robots, 2013b]. The manipulator itself is allowed to work without any safety precautions, due to the force in a collision cannot excess 150 N [Universal Robots, 2013a]. The manipulator consist of an UR5 articulated arm, a teach pendant, and a CB2 control box. The hardware is shown in figure 2.6. The three main interfaces to the manipulator are:

[1] See: <3rd Semester Report/Creation of an Autonomous Industrial Mobile Manipulator.pdf>

- **Polyscope** online programming interface on the UR5 teach pendant. [Universal Robots, 2013c]
- **URScript** scripting language, to be executed via the existing Polyscope interface. [Universal Robots, 2013c]
- **C-API** low-level language with most functionalities, but requires creation of a new controller. [Universal Robots, 2013c]

Each interface has different pros and cons regarding functionality, ease of application, and cooperation with peripheral equipment like a PC. The interfaces are described in detail in appendix B.

(a) The Universal Robots UR5 manipulator.

(b) Universal Robots teach pendant and CB2 controller.

Figure 2.6: Universal Robots UR5 manipulator, teach pendant, and CB2 controller. [Universal Robots, 2013d]

Robotiq 3-finger Adaptive Gripper

The Robotiq 3-finger Adaptive Gripper (RQ3) is an adaptive, under-actuated 10 degrees of freedom gripper. The under-actuation is due to only 4 of the degrees of freedom are actuated. This entails that the fingers adapt around the object being grasped, but also that the size of object cannot exact be determined from the position of the actuators. The RQ3 gripper is shown in figure 2.7a.

Little Helper 4 Platform

The platform was designed, based on the design of Little Helper 3. This basis was chosen due to appearance and interchangeability of hardware. The convergence in the appearance of the two platforms was created to emphasise that the two Little Helpers are closely related. The interchangeability of hardware concerns a modular design of the Little Helper 4 platform, which consist of an upper and lower part. The lower part has been prepared for replacement of the manual platform with the Neobotix MP-L655 platform. A Neobotix platform is currently mounted on Little Helper 2 and 3. The upper part contains

(a) Robotiq 3-finger Adaptive Gripper (RQ3) [Robotiq, 2013].

(b) The Little Helper 4 platform.

Figure 2.7: The Robotiq adaptive gripper and the Little Helper 4 platform.

the Universal Robots CB2 control box, power supply for the gripper, relays and an Ethernet router. The platform is shown in figure 2.7b. The platform has been equipped with a pan-tilt unit, with a stereo camera system and a RGB-D sensor during this project. The mounting of the pan-tilt unit is described in appendix I.

2.2.2 Original Software Framework for Little Helper 4

The original software framework for Little Helper 4 was developed based on a goal to make the motion primitives, skills, and tasks hardware independent. This goal gave rise to a framework divided in two layers, a generic layer, and a hardware specific layer. The generic layer contains a task level and a skill level. The hardware specific layer consist of a branch for each specific piece of hardware, divided in a proxy level and a hardware level. The structure of the framework is shown in figure 2.8, including the type of communication between each level. The framework was created with offset in the Robot Operation System (ROS), which is a set of tools and packages used to develop robot applications [ROS, 2013].

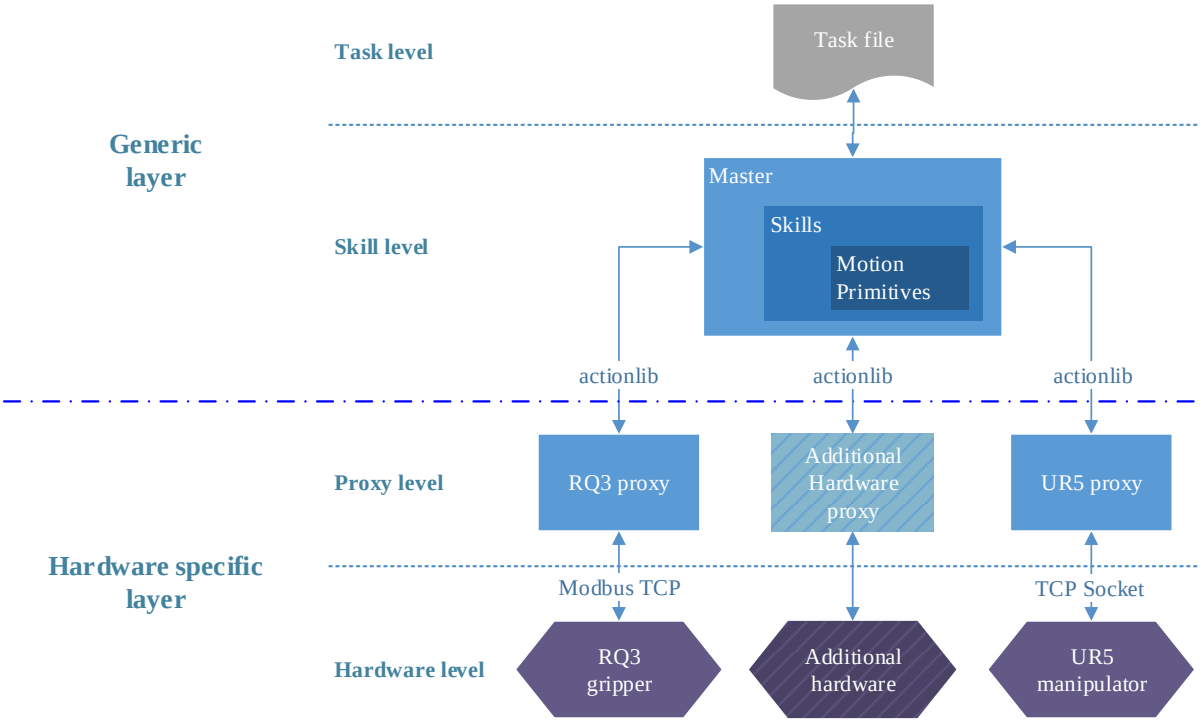


Figure 2.8: Software framework. Each component of the Proxy level can consist of both a proxy and a driver. [VT3-2013, 2013]

To achieve hardware interchangeability in the framework, by use of this approach, the hardware are divided into different categories. Each category is represented by a set of generic hardware functionalities, which have to be fulfilled by a piece of hardware, to be represented in the category. Generic hardware functionalities are common functionalities for a type of hardware, and could be the ability to conduct a linear movement, for a manipulator, in either joint space or Cartesian space. All functionalities outside the generic hardware functionalities are denoted hardware specific functionalities. The framework is only generic as long as the used skills, and thereby motion primitives, utilise generic hardware functionalities. This causes the skills and motion primitives to be divided generic or hardware specific categories.

The generic framework is supported by the structure, where the hardware proxies and drivers are decoupled from the skills and motion primitives. This also includes an interface between the two layers, where hardware neutral commands are exchanged by use of ROS actionlibs. Actionlibs are used as the interface, to control the order of execution among the motion primitives and making sure, the hardware has executed the received commands before returning a succeeded status.

The proxy level can consist of a combined proxy/driver, which both handles the communication from the hardware to the skill level, and control of the hardware, which is the case for the UR5 proxy. The driver

part of the UR5 proxy is inspired from a driver, which is part of the robot simulation and control tool RobWork [Ellekilde and Jorgensen, 2010]. The RQ3 proxy consist of a separate proxy and driver. The driver is from the ROS-Industrial Robotiq package[2] [Nicolas Lauzier, 2013].

The skills in the original Little Helper 4 framework are not object oriented, as described in section 2.1.1, but based on 3D coordinates. This invokes the need for a way to teach 3D coordinates to skills. The manipulator of the original Little Helper 4 framework is used to teach 3D coordinates, based on kinesthetic teaching, in which the operator moves the manipulator to the desired positions. Parameters, not directly related to the positions are taught through the terminal user interface of the software framework.

[2]The package can be found at: `https://github.com/ros-industrial/robotiq`

Part II

Main Report

Chapter 3

Goal for the Project

This chapter presents the overall goal for the project. This gives rise to a proposal of how the goal shall be achieved. The proposal is summarised into five hypotheses, which each has to be answered, to ensure the overall goal is achieved. Each hypothesis is concluded with one or multiple subsidiary goal statements that will ensure the respective hypotheses can be answered.

The overall goal for this 4^{th} semester project is to conceptual implement and verify motion planning, within a skill-based system for present and future research projects.

The development shall take its origin in two conceptual proposals that shall be created based on development sessions. One proposal shall be a long-term proposal, where limitations regarding development and implementation shall not be taken into account. The second proposal shall be a short-term proposal, which has to be applicable for practical implementation in the skill-based system of the Little Helpers.

Next, a software motion planning tool for implementation shall be found. This shall be done by a screening of applicable software solutions, to find the solution that is assessed to be most applicable for a skill-based system. The chosen motion planning software tool shall be further analysed with respect to types of motion planners, through a literature study and a benchmarking, to choose one or a set of the most suitable motion planning algorithms.

As proof of concept, the short-term conceptual proposal shall lead to a solution, where motion planning shall interact with a skill-based system. The skill-based system, which it is chosen to implement the motion planning in, is the software framework of Little Helper 4. This shall clear the way and provide practical experiences, for when motion planning shall be implemented in future research projects.

Practical tests to verify the implementation shall be conducted. The tests must show usage of motion planning in a skill-based system, to proof the concept, and thereby the overall goal.

The overall goal can be summarised into the following listed hypotheses. Each hypothesis is concluded with one or multiple subsidiary goal statements that will ensure the respective hypothesis is answered.

Hypothesis 1

Concept proposals can be conceived, such that motion planning can be integrated into a skill-based system.
A long-term and a short-term concept proposal shall be conceived. The short-term proposal shall additionally be developed to the extent that it is possible, to confirm the applicability. The concepts shall be discussed with some of the developers of the skill-based system for Little Helpers, to receive feedback and verify the concepts. The aim is to confirm the legitimacy of the concepts.

Hypothesis 2

One or a set of suitable motion planning algorithms can be identified for Little Helper 4.
A literature study concerning motion planning shall be conducted, with focus on the applicability for Little Helper 4. The correlation between the literature and practical experiences, with the studied motion planning algorithms, shall be tested by benchmarking motion planning algorithms.

Hypothesis 3

Motion planning can in practise be implemented in a skill-based system.
Based on the conceived short-term concept proposal, motion planning shall be implemented in the Little Helper 4 software framework. Testing shall be conducted on the Little Helper 4 set-up, to confirm the implementation of motion planning within the software framework.

Additionally it is chosen to challenge some of the possibilities that arises, when incorporating motion planning into a skill-based system for Little Helpers.

This is done by introducing motion planning to both Little Helper 3 and Little Helper 4 in simulation and in practise. Motion planning with multiple manipulators may become relevant if a Little Helper shall be developed with two manipulators in the future.

It is chosen to introduce environment perception by sensors. This enables the possibility to motion plan in an unstructured environment, which is a beneficial addition to collaborative robotics, like the Little Helper family.

This can be summarised into the two following hypotheses that must be answered.

Hypothesis 4

Motion planning with two Little Helpers can be conducted.
To test the applicability of motion planning with two manipulators at the same time, then Little Helper 3 shall be implemented in the motion planning software tool as well. The applicability shall be tested in a real set-up, by placing the two Little Helpers within each other's work envelope, when motion planning.

Hypothesis 5

Motion planning can be conducted based on an environment representation with sensor input.
To test the applicability of motion planning, in a changing environment, a sensor-input shall be implemented, such that it can be utilised when motion planning. The test shall be conducted by placing obstacles, which the manipulator shall avoid, between an initial and goal state.

The listed hypotheses, including their respective subsidiary goal statements, will be answered throughout the report in the given order, and summarized in the final conclusion.

Chapter 4

Concept Proposals for Incorporating Motion Planning in a Skill-Based System

This chapter proposes two concepts for incorporating motion planning into a skill-based system (SBS). One proposal is a long-term concept proposal and one is a short-term concept proposal.

4.1 Purpose

The main purpose of this chapter, is to answer **Hypothesis 1** in chapter 3, and thereby to present conceptual ideas, which have been generated during the project, of how motion planning can be implemented in a SBS. The result is two submissions, which are respectively considered a long-term concept and a short-term concept proposal. The purpose of the long-term concept proposal is to introduce motion planning in a Little Helper SBS of the future. The aim of the short-term concept proposal is to present how motion planning can be handled in the current Little Helper SBS. As proof of concept, for the short-term proposal, an actual implementation of motion planning in the Little Helper 4 software framework, is conducted and described in chapter 8.

4.2 Long-term Concept Proposal

The following proposal shall be seen as a long-term solution for motion planning in the skill-based system, which may not be completely applicable to a Little Helper software framework at present. The concept is obtained through brainstorm sessions within the project group, and afterwards presented,

discussed, and verified in overall terms, in a session with a part of the *Robotics and Automation Group* at *Aalborg University*. The proposed framework for the concept is shown in figure 4.1.

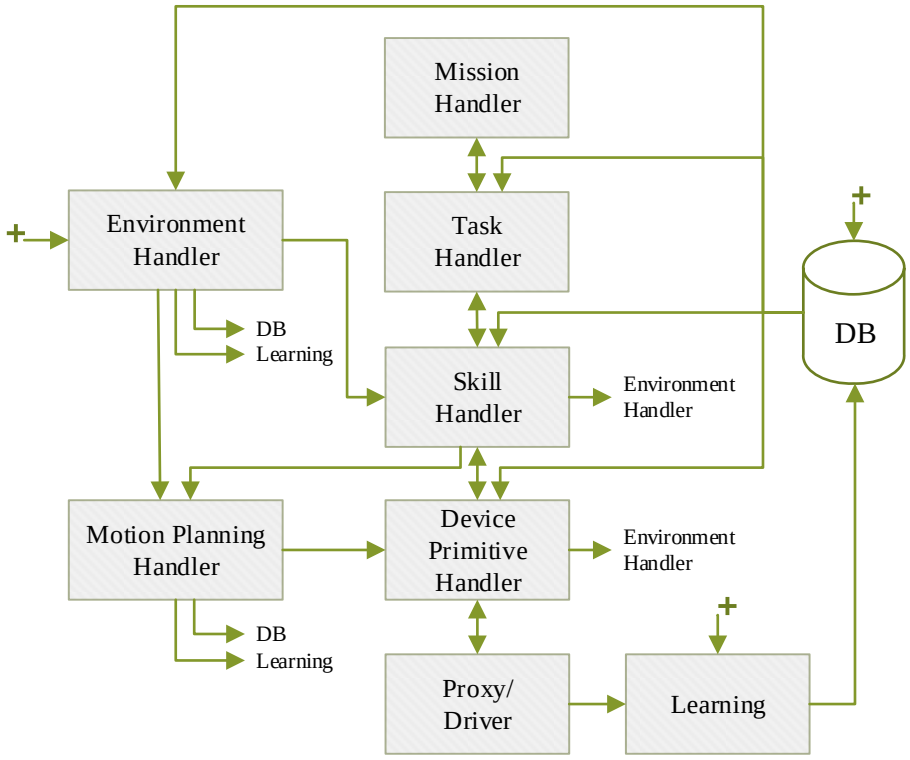


Figure 4.1: Framework of the long-term concept proposal for implementation of motion planning in a SBS. The plus signs on the figure illustrates communication from the arrows with a corresponding name. DB is an acronym for database.

4.2.1 Structure of the Skill-based System

The structural division of missions, tasks, skills, device primitives, and proxies are all maintained in the skill-based system of the long-term concept proposal, as seen in figure 4.1. This is because there has not been found any reasons to alter it, with respect to the concept proposal of incorporating motion planning in it.

In practise, it is proposed that the structure will reflect figure 4.1, meaning decoupling the skill-based system into multiple ROS nodes. This will benefit the maintenance and development processes. Additionally computationally expensive operations, such as handling vision or logging to a database, may be outsourced to off-board computers by the peer-to-peer functionalities ROS offers, schematic illustrated in figure 4.2.

The long-term concept proposal has to allow collaboration between multiple Little Helpers. Several

Little Helpers can collaborate and share information by use of a the peer-to-peer functionalities and a common database, as illustrated in figure 4.2.

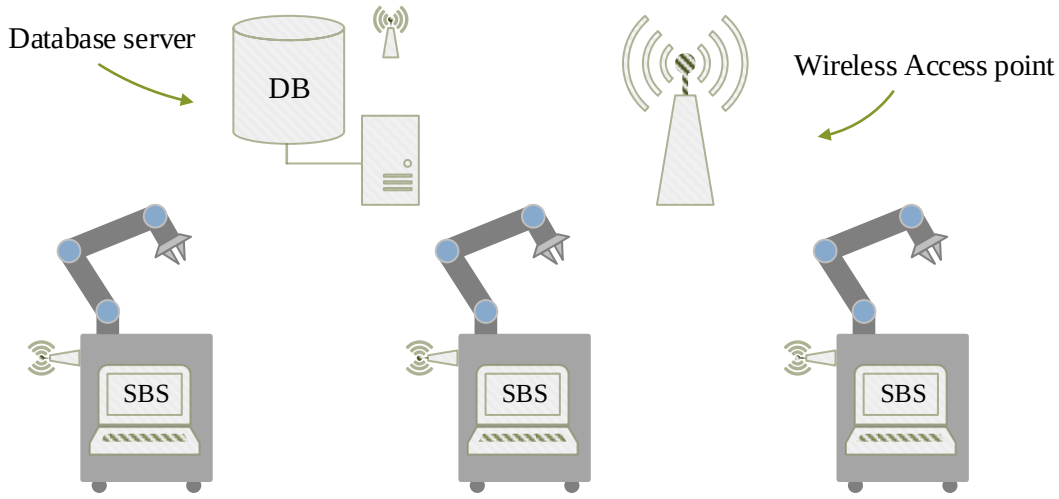


Figure 4.2: Illustrates how multiple Little Helpers can collaborate, and share information with each other via wireless network and a database server.

4.2.2 Tasks, Skills, and Device Primitives in the Skill-based System

As seen in figure 4.1, it is proposed to centralise the stored information. Thus, information the missions, tasks, skills, and device primitives may contain is moved to a common database in this proposal. The main objective for centralising the information is for learning purposes, which will be elaborated later in this section. The result of extracting the missions, tasks, skills, and device primitives into a common database, is having *handlers* instead. A task handler's assignment is to request the tasks stated in the mission. A skill handler's assignment is to request needed skills, to fulfil the given task. Likewise is the assignment of a device handler to request for the needed device primitives, corresponding to the hardware of the system and the skill to be executed.

The device primitives shall have continuously communication with some hardware components for supervision of the system, such as vision sensors to keep the environment up to date. This means that the device primitive handler may have some device primitives executing all the time.

Communication between the device handler and the skill handler includes skill functionalities not related to motion planning. This can be communication related to force or vision. Keeping the communication is additionally assessed beneficial, since this will maintain the possibility for the skill-based system to function without motion planning. This may be applicable if situations occurs, where it is not feasible to obtain the environment or where motion planning simply is not needed.

4.2.3 Motion Planning Handler

Implementing motion planning abilities into a SBS is not limited to motion planning for an articulated manipulator, if other actuated equipment is applied. Motion planning for, for example, grasping or moving a mobile platform (locomotive) will be a necessity, if a skill-based system shall function more autonomously. A *motion planning handler* is needed to encapsulate software packages, if not all these possibilities are contained in one package. A motion planning handler is not needed, if a motion planning software tool is able to cope with all needed types of motion planning. A selection of present motion planning software tools are to be further presented in chapter 5.

4.2.4 Environment Handler

An environment representation is needed for the motion planning. Motion planning software for multiple types of equipment's are assessed to have different needs, concerning an environment representation. This imposes a need for multiple types of sensors to create a representative environment.

Creating an environment based on sensor input, and not CAD models, is assessed to be less time consuming, because the environment does not have to be completely documented. The drawback is that the sensor-based models are assessed to become less accurate, than if the same model is represented in a CAD model and in addition it will be more computational expensive to create the model. Some parts of the environment may therefore be created as CAD models, such as work pieces and the manipulator, but it cannot be expected that every object in an environment is created as a CAD model. Thus, a need for combining the sensor input and CAD models arises. Additionally it will be beneficial if the models, generated by one mobile manipulator, is shared between multiple mobile manipulators. This gives rise to the creation of an *environment handler*.

An environment handler is expected to interface to; the database for CAD models, the skill handler, the device primitive handler, the motion planning handler, and other applications in relation to the environment, as shown in figure 4.1. Thus, the environment handler becomes a gathering point for all information in relation to the environment. This enables the possibility to exploit the peer-to-peer capability of ROS, by having computationally expensive calculations on an off-board computer. This may, on a greater scale, also be developed, such that multiple Little Helpers each can share and contribute with their environment perception to a common database, as illustrated in figure 4.2.

An application that can interface to the environment handler may be the creation of a factory environment for locomotive navigation. In [Henry et al., 2014] an example of creating such a large-scale environment is done to an office environment, by use of sensor data. They have developed an iterated closest point (ICP) algorithm that enhances a process called "simultaneous localization and mapping" (SLAM) [Endres et al., 2012]. The process is able to scan, and thereby map an entire office environment, by sampling

depth images and combining these into a complete 3D environment. At present [Henry et al., 2014] achieves a mean accuracy of 15-20cm. This will need to be further enhanced, if used for industrial tasks - for comparison [Hvilshøj et al., 2012] states that an industrial assembly needs an accuracy of less than ± 1mm. Their solution is mainly aimed at locomotive motion planning, thus the two values are not directly comparable.

4.2.5 Learning

Learning is seen as crucial for motion planning enhancement, and a tool for a SBS to become more autonomously [Kallmann and Jiang, 2010]. Learning can be implemented in multiple manners with varying complexities, whereas common for all, is that learning shall increase the performance of a skill-based system, and ultimately make Little Helpers act based upon previously experienced tasks.

It is experienced that learning is typically based on experience. Experience is here seen as previous conducted tasks, where the experience can be based on the tasks overall assignment and the collection of executed trajectories in the tasks. Due to the changing environment, the Little Helpers are intended to work in, iteration of a tasks do not necessarily include the same trajectories. This gives further rise to the need for a shared database, allowing for exchange of the data basis for learning. The sharing of data allows for maintaining the flexibility of conducting various tasks with a relatively low number of iterations. Different research projects are introduced in the following, which all have implemented learning in different manners.

One proposal is to use motion planning algorithms, which utilises iterative learning, based on paths created in the past. Such a motion planning algorithm is created for example by [Li and Shie, 2002][1]. The motion planning algorithm is able to reuse previously created solutions, to create a new solution.

Optimising motion planning is achieved in [Srinivasa et al., 2012] for the mobile manipulator HERB 2.0, previously mentioned in section 1.3 on page 6. They utilises a set of goals for the trajectory to be able to optimise for a low-cost solution. Their aim is to learn to predict an initial goal, at which a trajectory should end. The initial goal for the goal-set is made from learned data, where five learning algorithms are compared against selecting a goal randomly. The best performing learning algorithm makes it possible to achieve a final cost within 8% of the minimum cost of the given path, whereas the motion planner achieves a final cost within approximately 35% of the minimum cost, when selecting the goal randomly. [Srinivasa et al., 2012]

[1] Reconfigurable Random Forest algorithm.

Learning based on motion planning in a SBS framework

It is chosen to elaborate learning suggestions from [Kallmann and Jiang, 2010], where a motion planning framework for skills with learning, is introduced. [Kallmann and Jiang, 2010]

They have implemented learning to enhance motion planning of a tree-based search, by guiding the development of the tree with their Attractor Guided Planner (AGP) algorithm. The utilisation of attractor points, to attract a tree towards previously conducted trees, are beneficial, because it preserves the structure of a successful path, and even if a new obstacle collides with parts of the path, then the rest of the path is still considered reusable. [Kallmann and Jiang, 2010]

The implementation of learning is done by saving previously conducted tasks and then later reusing the paths, if the new task is assessed to be similar to a previously conducted task. Similarity metric is utilised to analyse if previously planned tasks are similar to a new task. The metric similarity is utilised, when comparing a query task, with a previous conducted task, found in the database. A previous conducted task is only considered reusable, if the local environment and query has similar characteristics. The main difficulty is to determine a strategy for what has to be taken into account, when comparing. To do comparison, metric techniques are utilised, which are able to take relevant obstacles into consideration, when considering candidate paths. Thus, obstacles that is too far from the initial or goal configuration are ignored. [Kallmann and Jiang, 2010]

When the comparison has identified a similar task in the database, then the stored paths for the concerned task is utilised to guide the planning of the new path, by use of extracted attractor points. This is done by piecewise linear interpolating the previous path, and defining the points connecting two segments, in the interpolation, for the attractor points. The interpolation, and thereby the attractor points, is next collision checked. [Kallmann and Jiang, 2010]

If a set of attractor points are found applicable to be reused, during the development of the new path, then the motion planning becomes biased towards regions of these attractor points. The biasing is done by use of Gaussian distributions placed at the respective attractor points in the configuration space. The distribution is used to replace the motion planning algorithm. The scale of the Gaussian is depended on how close the tree is to the given attractor. [Kallmann and Jiang, 2010]

Experiences are;

> *"... a significant improvement in computation speed for dynamic environments which maintain some structure between queries".*
>
> [Kallmann and Jiang, 2010]

Learning may in the future also be beneficial to more general parts of a skill-based system. In [Kallmann and Jiang, 2010] it is proposed to enable the possibility to improve the skills, or even create new skills from learning, by interpolating between similar solutions. This can for example be; a pick skill and a vision skill may be needed in the same task. If the task is executed multiple times, a learning phase may interpolate the two skills into a pick-vision skill, and maybe thereby enhance the performance. The same idea may be applicable on task level [Kallmann and Jiang, 2010]. The results for the Little Helper SBS will be a redefinition of skills, being more dynamical, instead of pre-determined building blocks of a task. This will obviously first be applicable, when the system has become so autonomously that no user-interaction is needed for the skills.

4.2.6 Database

It is assessed that learning first can be utilised to its fullest, if system information is effectively saved and made accessible. By system information is for example meant joint configurations to a given time or 3D point cloud information to a given time. Additionally it is assessed beneficial to save system information in a global database, such that it will not only be restricted to the individual SBS. It shall be emphasized that the shown database in figure 4.1, not necessarily implies that all information shall be stored in one common database in practise, thus exactly how the structure of such a database shall be, is not within the scope of this project.

A generic database for fault analysis and performance evaluation, similar to the proposed, has been presented in [Niemueller et al., 2012]. They utilises a system with a scalable database, to log system information from ROS in real-time, controlling the mobile manipulator HERB 2.0. They categorise the vast amount of data using the Data-Information-Knowledge-Wisdom (DIKW) hierarchy. Their aim is to understand when Actionable Knowledge appears, which is usable knowledge for the manipulator to make decisions based on. An example of a DIKW structure is seen in figure 4.3. [Niemueller et al., 2012]

The aim is to make robotics behave based on a subset of Actionable Knowledge, thus reach the Wisdom level.This is however not achieved at the publication time of [Niemueller et al., 2012].

Based on the developed database, two applications are presented; one where the system fails a task and the necessary information for error recovery is retrieved from the saved data and one where it is shown how to retrieve quantitative performance data of the manipulators behaviour. At present, the system is local on one HERB 2.0 mobile manipulator, but they likewise see great potential in having a cloud-based logging, such that information can be shared. [Niemueller et al., 2012]

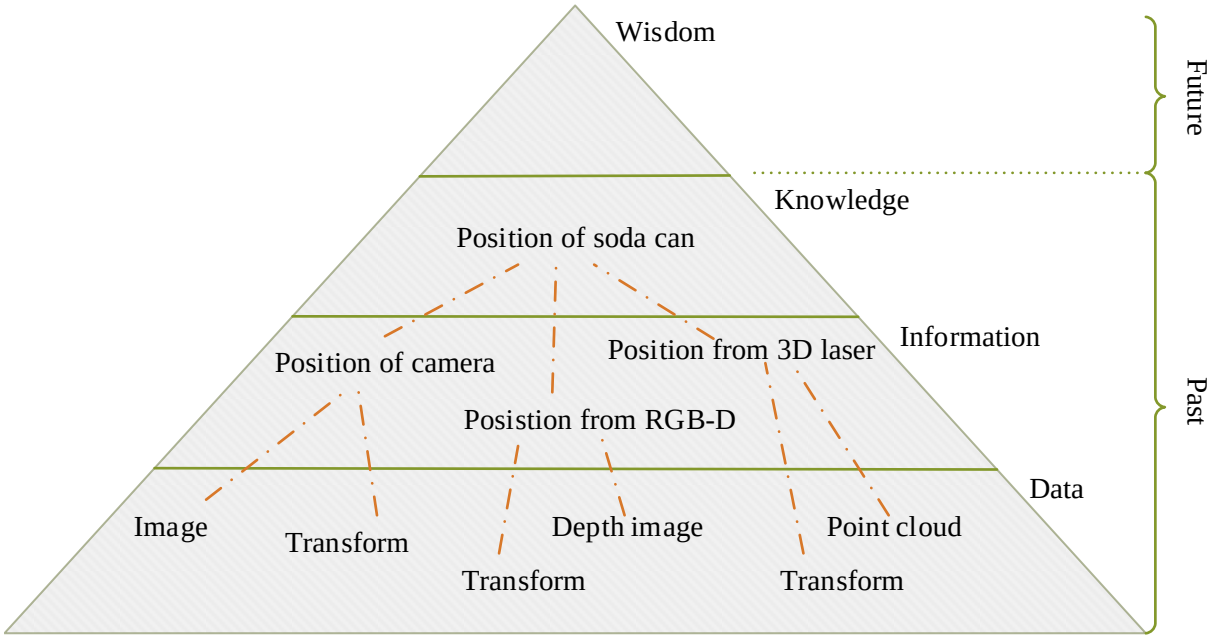


Figure 4.3: Example of how to classify a database utilising DIKW hierarchy. Inspired by [Niemueller et al., 2012].

4.3 Short-term Concept Proposal

The short-term proposal is seen as an applicable solution for the SBS for the Little Helpers. The proposal shall be the foundation for the practical implementation of motion planning in the Little Helper 4 software framework, described in chapter 8. The short-term concept proposal is first described in overall terms, followed by a description of each of the parts of interest in the concept. The overall framework structure of the concept proposal is seen in figure 4.4.

Both *Mission* and *Task* in figure 4.4 are faded, since these parts stay structural unaffected by the implementation of motion planning, in the short-term proposal. Only the content of task files will be altered, due to the introduction of motion planning parameters. The rest of the framework is to be elaborated in the following.

4.3.1 Skills and Device Primitives in the Skill-based System

The introduction of motion planning in the SBS eases the implementation of skills applied to objects, because the system itself avoids obstacles between positions. This removes the need for via-points, for example between pick and place skills. Pick and place skills of the Little Helper 4 SBS consist of approach and leaving points, to avoid collision with objects being handled. If the object to be handled are

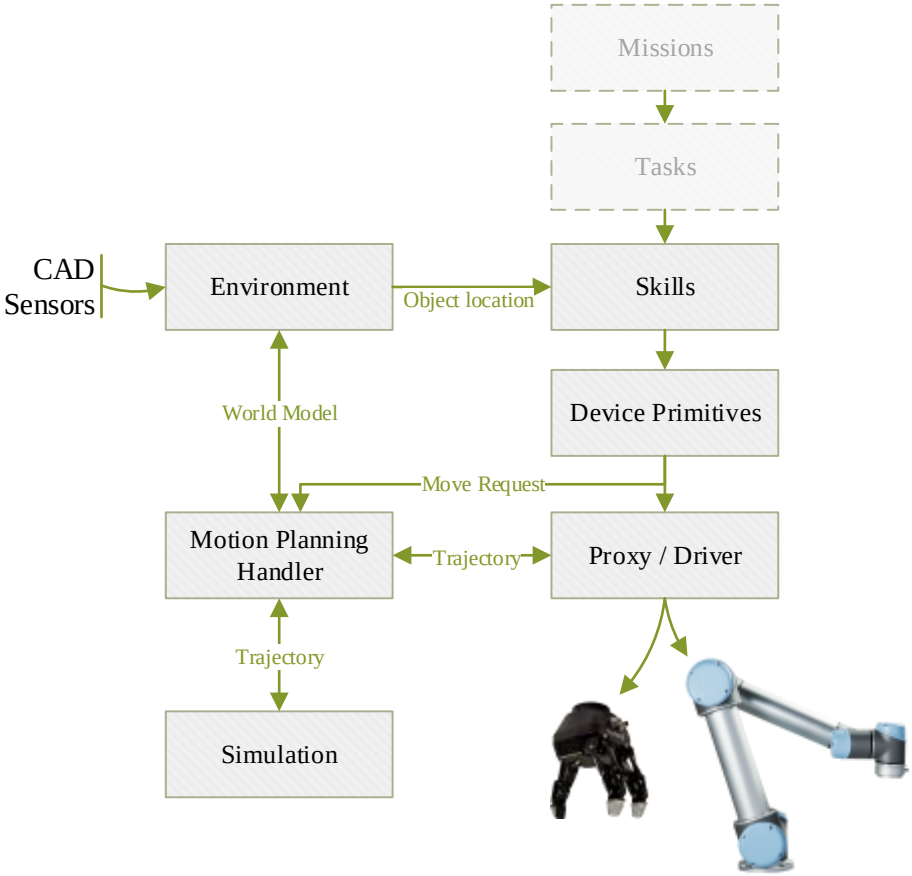


Figure 4.4: Overview of the software framework of the short-term concept proposal.

represented in the environment, approach and leaving points can be omitted. This is because the object has to be taken into account, during motion planning. The introduction of motion planning can therefore induce changes in the structure of specific skills. Structural changes of skills have not been investigated further. The model of the environment is further described in section 4.3.3.

Motion planning is, as seen in figure 4.4, conceived as a middle layer between device primitives and the proxy/driver for the hardware, which corresponds to the long-term proposal, described in section 4.2. Motion planning is therefore conceived as a device primitive, from the point of view of a skill. Motion planning is also conceived as device primitive from the point of view of the proxy/driver.

By treating motion planning as a device primitive, seen from the skill point of view, then skills will change from utilising a "move" device primitive to a "motionplan" device primitive. The parameters of the "motionplan" device primitive, are send as a request for a motion plan to a specified position, to a given motion planner.

Device primitives that is not related to movement of the manipulator stays unaffected of the addition of motion planning abilities.

4.3.2 Motion Planning Handler

As seen in figure 4.4, a *motion planning handler* is introduced in the SBS, corresponding to the long-term proposal. The motion planning handler shall be able to receive a motion planning request, and pass it along to the correct motion planning software tool.

The reason for also presenting a motion planning handler in the short-term proposal, is that it is assessed that different motion planning software tools is needed for the articulated manipulator (for example MoveIt [MoveIt, 2014b]) and for the gripper (for example GraspIt [Miller and Allen, 2004]).

The respective motion planning software tools shall use a motion planning request to initialise the creation of a trajectory. Additional needs for the motion planning software tools are a specification of feasible and infeasible areas of the configuration space, hence the need for an environment representation. The creation and maintenance of the environment is described in section 4.3.3.

Each motion planning software tool is expected to include motion planning algorithms with different advantages. By advantages are meant that algorithms may be dedicated to a certain kind of motion planning problem. This can be that some are faster converging in environments with no obstacles, and some may be more capable of solving complicated motion planning requests. The general type of motion planning problem could be defined as part of the teaching phase of a task. To be able to do this it is proposed to categorise the motion planning algorithms, such that there can be chosen between a variety of motion planning algorithms accordingly to the general type of the given tasks.

When a trajectory has been created, it shall be executed either on the real hardware or in a simulation. Simulation in a SBS is elaborated in section 4.3.5. Execution on the real hardware is not of concern here, since the implementation is hardware specific, the implementation for the UR5 manipulator is described in chapter 8. Feedback from the simulation and real hardware is needed, to determine the initial configuration of the hardware, and thereby the start configuration for the motion plan.

4.3.3 Environment

As the *environment handler* of the long-term proposal in section 4.2, the short-term proposal is expected to include a service to create and maintain the environment. The environment representation is to be created such that the motion planning software tools are able to check, whether a path is feasible or infeasible. An environment representation is also expected to be used by skills, to require for example the position of an object to be picked. Sensor-based localisation can both be based on existing CAD models

of objects and objects without a known model [Saxena et al., 2011]. The creation and maintenance of an environment is described with respect to a model-based representation and a sensor-based representation.

Model-based Representation

A model-based representation of the environment will be based on CAD models. Compared to sensor information, these are assessed to give a more accurate environment representation. The drawback is that they are inflexible and time consuming to obtain. A model-based representation is the obvious choice for the platform of a mobile manipulator, because it does not change relative position to the manipulator, and is always within the working envelope of the manipulator. Objects being handled by the manipulator or workstations, requiring high precision, can be represented by use of model-based models.

Sensor-based Representation

An immediate choice for sensor type is vision, such as for example 3D time-of-flight sensors [May et al., 2006] and RGB-D sensors [Henry et al., 2014]. RGB-D sensors are further elaborated in appendix H.3. Both sensor types provides visual information and a depth perception.

Sensor information for the environment representation is expected to work as a supplement, to the model-based representation in the short-term. This is because changes in the environment is expected to occur for collaborative robotics.

Calibration of the Environment

Both model-based and sensor-based representation of an environment must be calibrated relative to the mobile manipulator. Sensor input is assessed to be represented relative to the respective sensor, entailing the need for a calibration of the sensor. Two calibrations approaches are proposed; calibration based on haptic markers and calibration based on visual markers. Haptic calibration uses the manipulator and force feedback, to determine placement of haptic markers or calibration surfaces for the object being calibrated [Pedersen, 2011]. This could for example be the calibration of a manipulator relative to a mobile platform. Calibration of vision sensors relative to a manipulator can be conducted by use of visual markers, for example QR codes [Andersen et al., 2013]. Visual markers can also be used to calibrate the mobile manipulator relative to model-based representations, for example workstations [Hvilshoej et al., 2010].

4.3.4 Error Handling

Error handling is an important aspect, and an effective error handling can aid lowering the needs for user inputs. Even though processes are designed to be robust, error handling must be incorporated in the system. Errors described here are focused on errors, related to, or caused by motion planning. Errors concerning motion planning can be, among others:

- Failed motion planning (an algorithm fails to create a successful path).
- External activation of emergency stop.
- Collision (inconsistencies between the real and modelled environment).

The system shall in general handle errors in two different ways, automatic or manual. Automatic error handling is applicable on known errors, with known solutions. Manual error handling is applicable on errors, without a known solution or if the known solution is too complicated, compared to the occurrences, to create an automatic recovery.

A failed motion planning attempt can be automatic handled by conducting the motion planning request again - perhaps with changed parameters, as for example increasing the allowable planning time. If a collision has happened, then an operator can be called to verify whether there is inconsistency between the modelled and real environment or not, and if possible, correct the error.

4.3.5 User Interaction

A main reason for implementation of motion planning into the SBS of Little Helpers is to reduce the need for user interaction. The ultimate goal is to do without user interaction. This is however assessed to be infeasible in the short-term proposal, which is why some general ideas about how user interaction can be reduced, with respect to motion planning, are presented.

The utilisation of motion planning makes use of via-points redundant, and thereby reducing the required user input. Motion planning also lowers the demands to the user of the system, if the skills are applied to 3D coordinates. This is because no intermediate configurations of the manipulator, is needed to avoid obstacles contained in the environment.

Motion planning does however invoke a set of parameters to be determined. This can be with respect to the utilised algorithm and corresponding parameters. It is assessed that this can be avoided with sets of predefined parameters, suited for different applications. The needed user-inputs must be kept on a level such that the SBS of Little Helpers can continue to be used by non-experts.

Specification of Parameters

Specification of parameters for skills are done as a part of the teaching phase. Parameters can either be specified by the user or specified from predefined values. By specifying all parameters related to skills in the task file, then the tasks becomes independent from the specific SBS they are created on. They can thereby be executed on any SBS with the intended parameters.

Specification of Process Constraints

In addition to position constraints, process constraints for motion planning may likewise be of concern. Process constraints can be the velocity of the TCP or joints, the pattern of movements, the angle of a tool, etc. In the long-term it is assessed that process constraints will be predefined through, specified work sheets, currently being developed as a part of the ACAT project, which is shortly presented in appendix I. Until this is fully developed, it is proposed to directly specify the process constraints.

In [Kallmann and Jiang, 2010] process constraints with respect to movements are specified by a humanoid robot, imitating a person. By capturing and analysing the motion, then the respective constraint is classified into one of the following five groups; stationary, translation-only, rotation-only, patterned-translation, and patterned-rotation. The capturing of a constraint is done, by comparing the Cartesian pose of the manipulator at different instances of time. To make the system robust, a set of thresholds have been determined for the capturing. [Kallmann and Jiang, 2010]

By altering the application to kinaesthetically teaching, instead of imitation, then it is assessed this is an applicable user interaction to Little Helpers, if process constraints becomes a concern.

Simulation

Simulation tools can be used to visualise created trajectories. The ability to simulate the result of motion planning makes it possible to conduct a system test, without being connected to the real manipulator. This eases the user interaction during development, the creation of functionalities for the system, and avoid potential dangerous situations during debugging.

The complexity of the simulation is based on the desired output; if the need is to do a visual verification of the trajectory to be executed, a simple position simulator can be utilised. More advanced simulators can also include the dynamics of a manipulator, physics, and potentially the control parameters from the controller of the manipulator. This will create a simulation response, which can be used to verify for example velocities and accelerations of trajectories.

4.4 Sub-conclusion

The main purpose of this chapter was to answer **Hypothesis 1** in chapter 3, hence to introduce the conceived conceptual proposals regarding integration of motion planning in a SBS, where the proposals were divided into a long-term concept proposal and a short-term concept proposal.

The aim of the long-term concept proposal was to pave the way for how motion planning could be incorporated in a SBS of the future. This was done by proposing an incorporation of motion planning on multiple types of equipment, such as; articulated manipulators, grippers, and locomotion, in a SBS software framework. Based on this, it was deduced that a motion planning handler was needed, if one motion planning software tool did not encompass all these functionalities. To suit the needs of the motion planning handlers multiple types of motion planning software tools, a need for an environment representation was established. Based on this, it was determined that an environment handler was needed. The environment handler had to be able to both encompass CAD models, as well as sensor-based representations. It was additionally proposed to conduct enhancement on the different motion planning software tools. To cope with this, it was identified that learning had to be incorporated in the SBS. It was deduced that the system would become more autonomously, by incorporating learning, hence the needed user interaction was assessed to be negligible. Learning entailed the need for a common database for all Little Helpers, where knowledge could be shared. The addition of a common database induced the possibility to extract information from tasks, skills, and device primitives, to the common database, and thereby was handlers introduced. This was determined necessary, to be able to share and enhance these. It was concluded that the overall structure of a SBS was maintained, although handlers were introduced.

The short-term concept proposal was based on the long-term concept proposal, but aimed at being more applicable for a present SBS. It was deduced to maintain the motion planning handler in the short-term concept, to encompass for multiple types of motion planning software tools. The motion planning handler did likewise induce the need for an environment representation to the SBS. It was determined to maintain the rest of the overall structure of the original SBS of Little Helper 4. This was to support the desire, to do an implementation on the Little Helper 4 software framework. Proposals to cope with the user interaction, error handling, and calibration was likewise proposed.

The two concept proposals were, if possible, supported by exemplifications on how other research projects tackles similar functionalities, as the given proposal within the concept proposals.

The resulting software framework for both the long-term and the short-term concept proposal, were discussed and verified in overall terms with the *Robotics and Automation Group* at *Aalborg University*.

Chapter 5

Applicable Software

The purpose of this chapter is first stated. This is followed by a screening of applicable software solutions, which have to fulfil the needed functionalities, described in the short term proposal for the skill-based system (SBS) in section 4.3. Based on the screening, a software is selected for implementation in the SBS of Little Helper 4. At last, the selected software is elaborated.

5.1 Purpose

The purpose of this chapter is to probe different software solutions, which possess the functionalities needed to implement motion planning in the Little Helper 4 software framework, as described in section 4.3. This comprises capabilities within motion planning, simulation, sensor usage, and utilisation of CAD models and sensor input for environment representation. In addition to the selection, the selected software is to be elaborated, to obtain a better insight, of how the software works. This is used for implementation in the Little Helper 4 software framework.

5.2 Motion Planning Software Screening

There exists several software solutions capable of motion planning, and among those, three software solutions is selected for screening. The first one is V-REP, which has been utilised by the *The Robotics and Automation Group* at *Aalborg University*. The next is RobWork, which the project group acquired knowledge about during a visit at *University of Southern Denmark*, as a part of the development of Little Helper 4 [VT3-2013, 2013]. The last is MoveIt, which the project group became aware of, by working with ROS. Each of the examined solutions is open source. The following subsections briefly outlines V-REP, RobWork, and MoveIt.

5.2.1 V-REP

V-REP is a simulation software, mainly used for simulating robotic motions, which was released in 2010. The architecture of V-REP makes it possible to individually control each object in the simulation environment, from among others, ROS nodes. It is possible to insert objects, such as manipulators and grippers. For those and any other type of mechanisms, forward and inverse kinematics can be calculated. Furthermore, three physics engines are available, for simulation of real-world physics and object interactions, such as collision response and grasping. At last, motion planning is possible for kinematic chains, such as an articulated manipulator. [Coppelia Robotics, 2014]

5.2.2 RobWork

The development of RobWork was started in 2006 at the *University of Southern Denmark* and is a software for simulation, and control of robotic systems. The goal of the framework is to:

> *"Provide a single framework for offline and online robot programming including modelling, simulation and (real-time) control of robotics"*
>
> [Ellekilde and Jorgensen, 2014]

It is composed of a number of C++ libraries with various functionalities. The functionalities are modelling of, among others, industrial manipulators; motion planning with path optimisation, collision detection, and inverse kinematics; manipulator, controller, sensor, and grasping simulation with kinematics and dynamics. RobWork features the possibility to integrate custom motion planning algorithms. [Ellekilde and Jorgensen, 2014]

5.2.3 MoveIt

Willow Garage have in 2013 developed the software solution, MoveIt, that integrates motion planning, kinematics, collision checking, manipulation, navigation, sensor input, and control [Chitta, 2013]. There has been focus on making the architecture behind MoveIt flexible, thus the possibility for plug-ins is enabled in the architecture. This entails that users are capable of integrating custom components, such as motion planners, kinematic solvers, etc. Benchmarking abilities have been implemented in MoveIt, to make it possible to test various motion planning algorithms, scenes, set-ups and parameters against each other. [MoveIt, 2014b]

5.3 Selection of Motion Planning Software

MoveIt is selected for implementation in the Little Helper 4 software framework, the reasons for the choice are elaborated in the following.

MoveIt is developed directly for integration with ROS. This matches the software framework of Little Helper 4, since it is based on ROS. This might ease the implementation of MoveIt in the SBS of Little Helper 4 compared to V-REP and RobWork.

MoveIt contains a default motion library, Open Motion Planning Library (OMPL), which contains a broad variety of motion planning algorithms. MoveIt has additionally the ability to include other motion planning libraries, or users can implement their own motion planning algorithms. A wide range of motion planning algorithms are desirable, to allow for a comprehensive benchmarking. MoveIt features, through OMPL 20 algorithms, whereas RobWork per default includes five algorithms [Ellekilde and Jorgensen, 2014]. The number of algorithms have not been determined for V-REP.

MoveIt is the newest of the three software solutions, and this induces interest in exploring its functionalities. There are some disadvantages with selecting a new software. One is the lack of experience in MoveIt, compared to V-REP because the *The Robotics and Automation Group* at *Aalborg University* has experience with it. It was assessed that the necessary information and help could be obtained through online communities.

5.4 Functionalities in MoveIt

MoveIt is a motion planning software solution, with open interfaces for utilisation of motion planning algorithms, kinematics solvers, manipulators, and sensors. The overall functionalities of MoveIt are described in the following. The description is biased in the architecture of the system, in that each of the components are described. The architecture of MoveIt is shown in figure 5.1. The outlining of MoveIt is based on [MoveIt, 2014b].

Move_group

Move_group is the main node of MoveIt, which integrates functionalities from the other components of MoveIt. Move_group handles the creation and maintenance of an environment. Maintenance can be done based on sensor input, which concerns both *Robot Sensors* and *Robot 3D Sensors*. The 3D sensor data is filtered before being used in the environment, which is also handled by the move_group. The environment representation is divided in two; a representation of manipulators and a representation

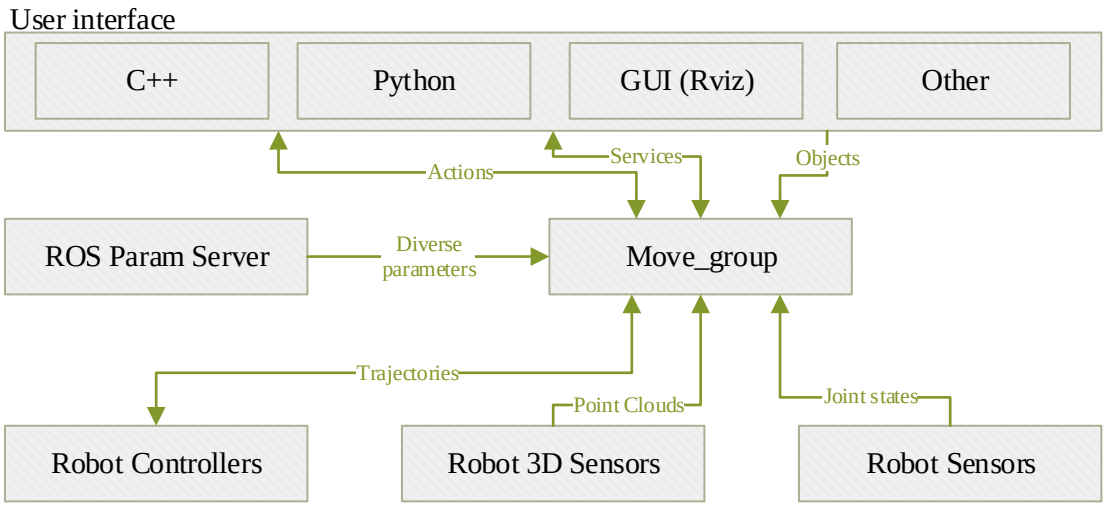


Figure 5.1: The high-level architecture of MoveIt, which shows the connections to the main node move_group. The rectangles represent ROS nodes. Inspired by [MoveIt, 2014b].

of the scene. *Manipulators* are a representation of a chain of links and joints, which can be part of a kinematics chain for motion planning. A scene is a presentation of the entire environment, without the manipulator.

User Interface

It is possible to utilise the functionalities of the move_group node via three different user interfaces; C++, Python or a graphical user interface (GUI). The GUI is implemented as a plug-in to the visualisation tool Rviz. Kinematic solving and receiving of motion planning requests are for example furthermore handled through the user interface.

Motion planning algorithms can be utilised through the user interface, allowing for use of custom algorithms. OMPL is the default algorithm library in MoveIt and consist of a set of motion planning algorithms, which are utilised through move_group. In addition to the algorithms, the library contains post-processing capabilities, which improves the output trajectory.

When working with robotics, kinematics is a necessity. MoveIt makes use of a plug-in structure, allowing the users to develop their own kinematics solvers. As default, MoveIt utilises the Kinematics and Dynamics Library (KDL). Another option is to use IKFast, which is an inverse kinematics solver provided in OpenRave. This solver is analytical and should provide more stable and faster solutions.

Collision detection is utilised, to validate paths and configurations. The Flexible Collision Library (FCL) carries out the collision detection in MoveIt. It supports collision detection for meshes, primitive shapes, such as boxes and cylinders, and filtered sensor data.

ROS Param Server

The ROS Param Server is used for storing configuration parameters used by MoveIt. This can include kinematic descriptions, collision disabling, and link to visualisation and collision files.

Robot Controllers

Robot controllers concerns the connection from MoveIt to manipulators. The information being send through this connection is trajectories to be executed and an accepted response from the manipulator.

Robot 3D Sensors

Robot 3D sensors concerns 3D sensor input, which is divided in two categories: point clouds and depth images. Image information is send to the move_group for filtering and further use.

Robot Sensors

Robot sensors concerns feedback from the manipulators in the environment. This is for example the state of the joint configuration of a manipulator. This is used by the move_group to maintain a correct representation of the environment.

5.5 Sub-conclusion

In this chapter, three software solutions, applicable for motion planning in the Little Helper 4 software framework, were probed. MoveIt was the software of choice, based on the capabilities of the software and the short-term concept proposal from section 4.3. The overall functionalities of MoveIt were outlined, to lay the foundation for further work with MoveIt. MoveIt is created with an open structure, allowing for interface to various types of hardware, corresponding to the generality of the SBS.

Chapter 6

Motion Planning Study

This chapter first introduces the purpose of studying motion planning. Thereafter two philosophies is introduced. The philosophy matching the requirements of this project is chosen. Finally, a study about the algorithms in the Open Motion Planning Library is carried out. The study of the algorithms is in this chapter on an application level, where some of the algorithms are described in appendix C.

6.1 Purpose

The purpose of this chapter is to partially answer **Hypothesis 2** in chapter 3, by first introducing two motion planning philosophies. The capabilities and the difference between the two are described to validate the selection of MoveIt, which uses the Open Motion Planning Library (OMPL). Subsequent, an examination of the motion planners on an application level is carried out, to make it possible to compare the motion planners. Finally this chapter and a benchmark (see chapter 7) must enable a selection of a single, or as few motion planners as possible, to be used in the skill-based system (SBS) of Little Helper 4.

6.2 Motion Planning Philosophies

Motion planning can be divided into two main philosophies: *Combinatorial motion planning* and *Sampling-based motion planning* [LaValle, 2006, p. 185, 249]. The following introduces the two types of philosophies, where sample-based motion planner also includes general definitions that are utilised in the rest of the report.

6.2.1 Combinatorial Motion Planning

Combinatorial motion planning algorithms, also referred to as exact algorithms, deals with motion planning, by finding paths in a discretised and complete representation of the continuous configuration space. Either this type of algorithms will find a solution or correctly, report that no solution exits, such an algorithm is defined as being complete. [LaValle, 2006, p. 249]

When using combinatorial motion planning, it is important to know the representation of the environment, in which a given algorithm has to function, since the performance of the algorithm is highly depended on the input. This is the opposite philosophy to the sampling-based motion planning algorithms, which does not need to know the complete representation in pre-hand. [LaValle, 2006, p. 185, 249]

It can be beneficial to use combinatorial algorithms for some special cases, because highly efficient algorithms can be developed, since they do not rely on approximations. However since they have to be developed to the specific representation of the environment, then they often will end of being impractical to implement. [LaValle, 2006, p. 250]

6.2.2 Sampling-Based Motion Planning

Sampling-based motion planning algorithms have advanced motion planning since the 1990's and are at present considered the state-of-the-art philosophy within motion planning [Burns and Brock, 2006]. In contrast to combinatorial motion planning, the main idea about sampling-based motion planning is that it does not explicit construct the obstacles in the configuration space (C-Space). Instead, a search is conducted, which probes the configuration space by use of sampling vertices and connect these with edges. The searching induces the need for collision detection that can detect the obstacles, and thereby navigate in the obstacle free space. [LaValle, 2006, p. 185-187]

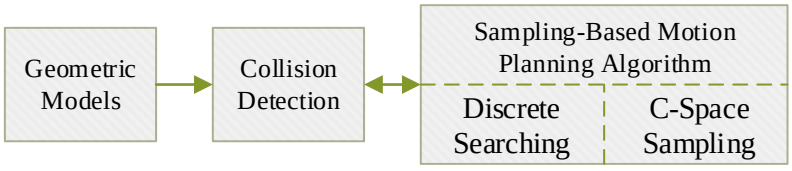


Figure 6.1: The Sampling-based motion planning philosophy. [LaValle, 2006, p. 185]

The collision detection is essential to get a valid path, but in practise it is not a part of the sampling-based algorithms (seen in figure 6.1). The collision detection is typically conducted elsewhere, and provides information whether a given state is valid or not. The process is often referred to as a *collision checker*, since the motion planning algorithm enquires the validity of the state. Normally the collision check is

done during sampling of a state, but sometimes it can be beneficial to postpone the collision check until a full path is made. An algorithm is often prefixed with *"Lazy"* if the collision check is postponed.

The reason for postponing the collision checking is, for example, that the probabilistic roadmap (PRM) algorithm uses up to 90% of its computational time on collision checking. The probability that a short connection between two configurations is collision free is large, and most connections in the PRM are not in the final path. This entails that the time used for checking collision for each connection, before a final path is found, is redundant. [Sánchez and Latombe, 2003].

Sampling-based motion planners are typically categorised as *single-query* or *multiple-query*. The difference being that a single-query has a single initial-goal query pair given, for which, a tree data structure is built to solve the query. Multiple-query planners first constructs a roadmap of undirected edges in a pre-processing phase. This roadmap is then used to answer multiple queries. The multi-query planners are beneficially in highly structured static environments, but if the environment changes the pre-processed graph gets invalid. Hence, in dynamic or in pre-hand unknown environments single-query algorithms are generally better suited. [Karaman and Frazzoli, 2011]

Contrary to combinatorial motion planning, sampling-based motion planning are not complete. Instead the *probabilistically completeness* is used to characterise the ability to find a solution. To be probabilistically complete means that with enough samples, the probability that the algorithm finds an existing solution converges to one. Many sampling-based motion planning algorithms are probabilistically complete, and the rate of convergence can be used as a performance parameter, thus rate of convergence, which describes the performance of an algorithm, is a better way to evaluate different algorithms, but it is in practice difficult to establish it. [LaValle, 2006, p. 185-186]

Sampling-based motion planning can be performed with *differential constraints*, which restricts the allowable velocity in each configuration through a path. MoveIt typically generates kinematic paths i.e. paths without differential constraints. The kinematic paths are time-parameterised afterwards to generate a trajectory, which the manipulator and its controller tries to follow [MoveIt, 2014b]. If the plans are made with differential constraints, then the plans are made to comply with the natural motions of the manipulators mechanical system [LaValle, 2006, p. 713].

6.3 Open Motion Planning Library

MoveIt enables motion planning through a plugin interface, and the default motion planning library is the Open Motion Planning Library (OMPL), which at present includes 20 different sampling-based motion planning algorithms. The characteristic with respect to their applications will be examined. This section, combined with appendix C, gives an introduction to these motion planning algorithms. Five planners is

to be presented in-depth in this section. These are chosen, such that they represent the range of diversity of motion planning algorithms in OMPL. The applicability for Little Helper 4 is assessed based on the theory, after the introduction of each of the five motion planners.

6.3.1 Probabilistic Roadmap

The Probabilistic Roadmap (PRM) algorithm is presented in [Kavraki et al., 1996] as a tool to do motion planning for manipulators with many degrees of freedom (five or more). PRM uses the multi-query method, where the basic idea is to divide the algorithm into two overall steps; a pre-processing phase and query phase. [Kavraki et al., 1996]

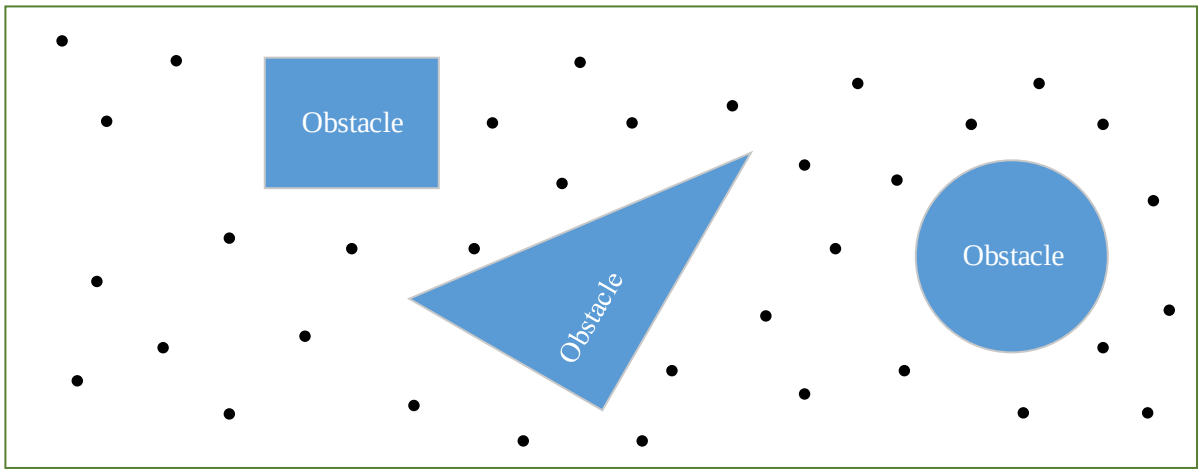


Figure 6.2: Illustrates the generation of random samples in the free configuration space. This is a part of the pre-processing phase.

The pre-processing phase consists of the creation of a roadmap. This is done by attempting to make connections between a set of randomly sampled configurations, in the obstacle free space, by use of a local planner (See figure 6.2 and 6.3). The local planner is typical limited to only include samples within a certain radius or a certain number of nearest neighbours. This is done until the roadmap is dense enough. [Kavraki et al., 1996]

The second phase is the query-phase, where the algorithm answers queries. The process of the algorithm is that a query first attempts to find a path from the initial configuration and the goal configuration, to two samples. Next, a graph search is conducted to find a sequence of edges connecting these two samples in the roadmap. The successive connection will at the end have created the path, as shown in figure 6.3 and 6.4.[Kavraki et al., 1996]

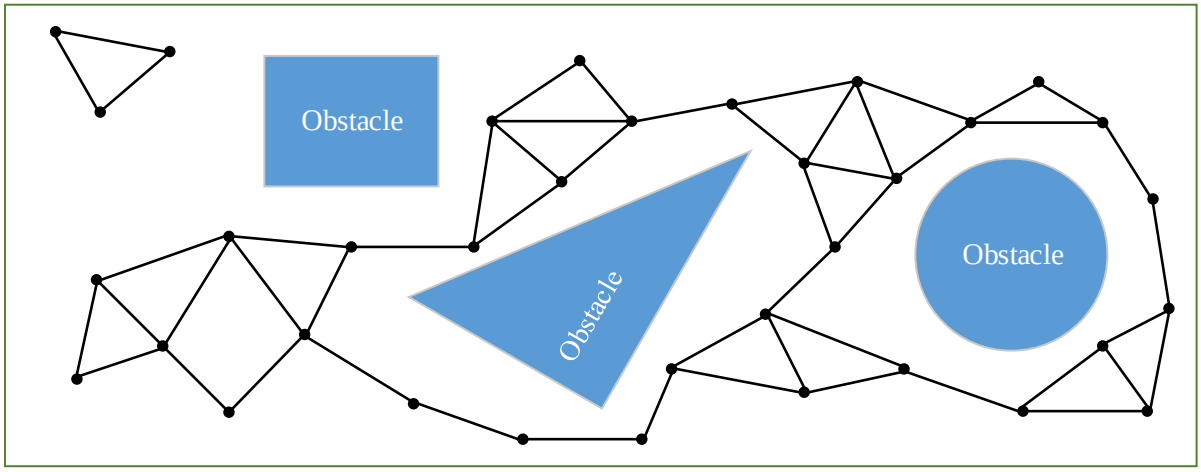


Figure 6.3: A roadmap after a local planner have connected the randomly sampled configurations. This is a part of the pre-processing phase.

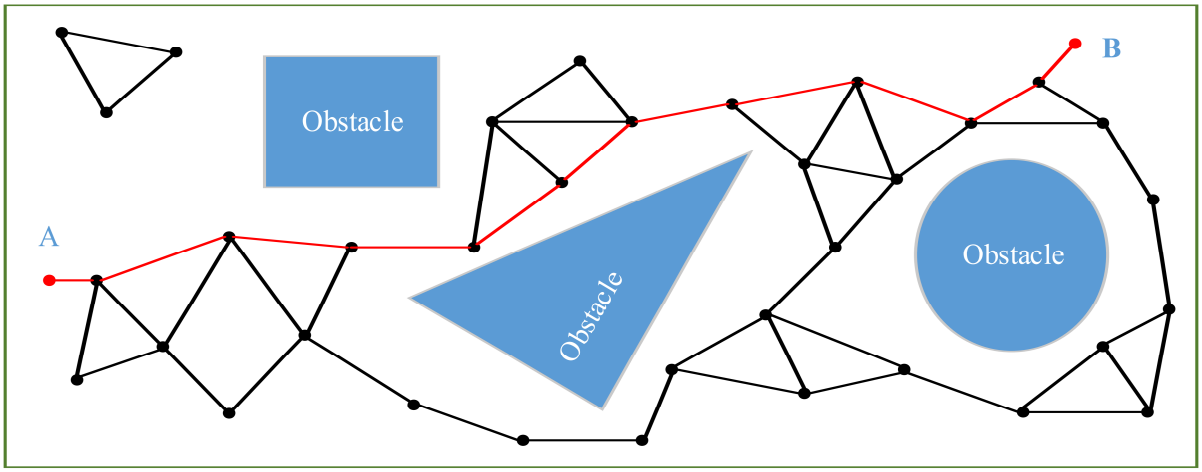


Figure 6.4: A solution to a query, where the initial and goal state is connected to the roadmap and a path is found.

OMPL contains a set of algorithms, which are developed from the original PRM algorithm. These are listed in the following:

- LazyPRM is inherited from the original PRM algorithm. The "Lazy" prefix means that the algorithm postpones the collision checking to reduce the process time [Bohlin and Kavraki, 2000]. The algorithm is further elaborated in appendix C.1.

- PRMstar is similar to the PRM, but the connection radius is depended on the coverage of the space. The algorithm is asymptotically optimal, which means the cost of the solution goes towards optimum as the number of samples increases [Karaman and Frazzoli, 2011]. The algorithm is further elaborated in appendix C.2.
- SPARS provides near-optimal solutions by building sparse sub-graphs from a roadmap created by the PRMstar algorithm. The sparse sub-graphs, with fewer nodes and edges, gives faster solutions to a query and are easier to handle for resource-constrained robots. Small roadmaps with multiple paths are advantageous in dynamic environments if moving obstacles invalidates path segments [Dobson et al., 2013]. The algorithm is further elaborated in appendix C.3.
- SPARS2 is comparable to SPARS in structure, but the memory usage under the construction of the sparse sub-graph is lowered, and the path are increased in quality [Dobson and Bekris, 2013]. The algorithm is further elaborated in appendix C.4.

Applications for the Skill-based System

In general it can be said that the PRM and the closely related planners mainly are suited for static environments, where the algorithms can fully utilise the pre-processing phase [Hsu et al., 1997]. However if a multi-query motion planning algorithm is to be used in a dynamic environment, then the SPARS2 will be the one to choose, since it utilises a sparse roadmap that should be relatively computational inexpensive. In addition, the algorithm is able to use another of the pre-processed queries, if the ongoing is obstructed.

Although probabilistic roadmap algorithms are able to answer multiple queries efficiently, they are not suitable when only a single query is possible. This can for example be in a narrow assembly problem, where it has to be determined whether there at all exist a path to assemble a component in a mechanical part. In that case, it will be more desirable to build only the part of the roadmap that is relevant to the query, like tree-based planners, compared to perform expensive pre-processing to construct a roadmap of the entire configuration space. This means that the choice between using single-query or multi-query motion planning algorithms is task-depended. [Hsu et al., 1997]

6.3.2 Rapidly-exploring Random Trees

The RRT algorithm is a single-query tree algorithm. This makes it suitable for online motion planning with a dynamically changing environment. The algorithm functions by expanding incrementally from a given initial state, thus it does not need to have a certain number of samples given in pre-hand. The algorithm will give a solution, when the tree reaches the goal state or fails if a termination condition is

fulfilled. Like the PRM algorithm, the RRT algorithm is probabilistic complete. [Karaman and Frazzoli, 2011]

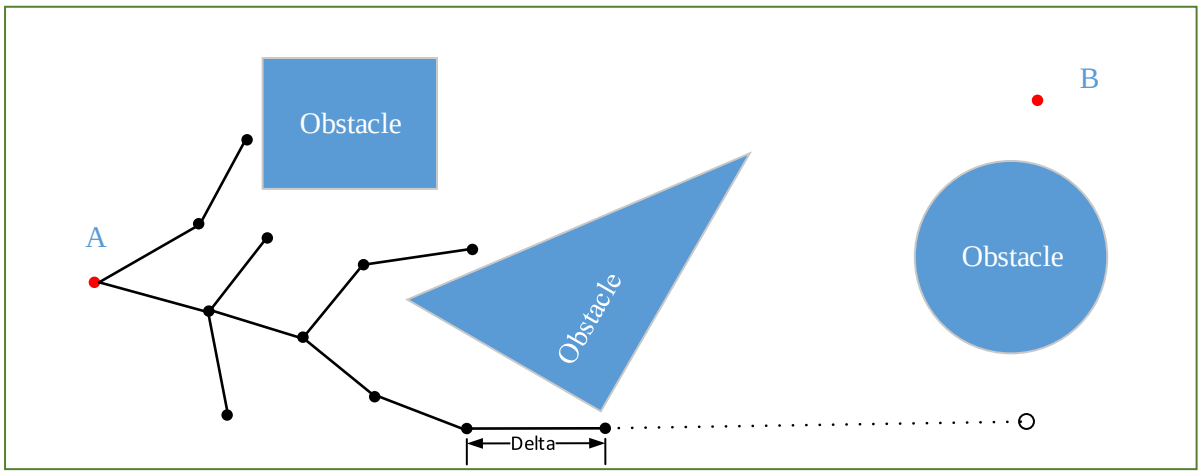


Figure 6.5: The beginning of a RRT, where a tree-based structure starts growing from the initial state A. If the shortest distance between the tree and a new sample is longer than delta, the sample is added to the tree at a length of delta. It looks like the tree is biased towards the goal, but in the original RRT this is not the case.

An example of the incrementing RRT algorithm, at its basic form, is shown in figure 6.5 and 6.6. The algorithm incrementally creates a tree of feasible edges, by randomly sampling new states in the free configuration space. The tree is extended with a fixed length *delta* towards the newly sampled state, by finding the nearest state in the tree. [Karaman and Frazzoli, 2011]

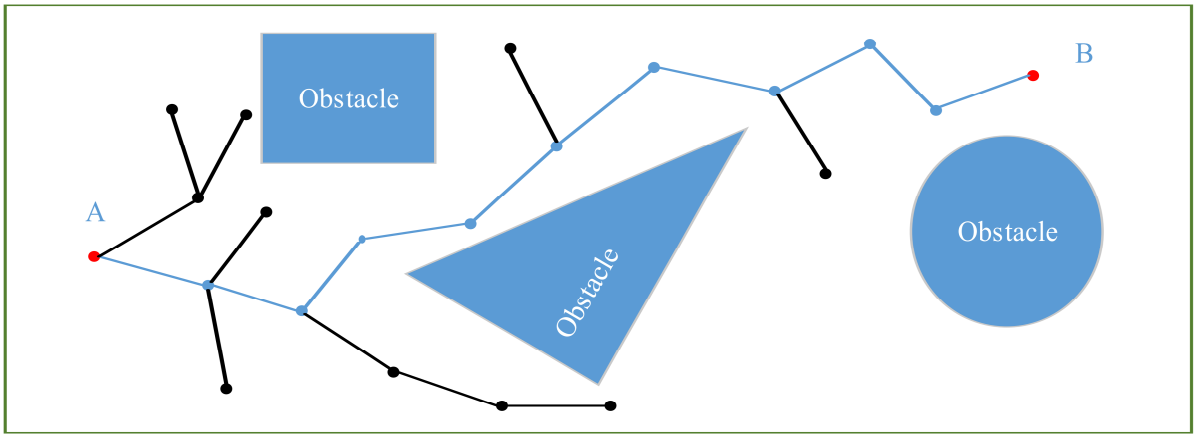


Figure 6.6: The RRT has reach the goal state (B).

The most basic form of the RRT algorithm expands randomly in the configuration space. A tree expansion can however also be biased, by means of Voronoi regions. Voronoi regions guides the tree towards less explored areas of the configuration space, whereas the probability of sampling in a certain Voronoi region is proportional to the size of the region [Kuffner and LaValle, 2000].

Tree-based algorithms can be divided into the following classes of algorithms, depending on the number of trees included in the algorithm; *uni-directional*, *bi-directional*, or *multi-directional*. [LaValle, 2006, p. 219]

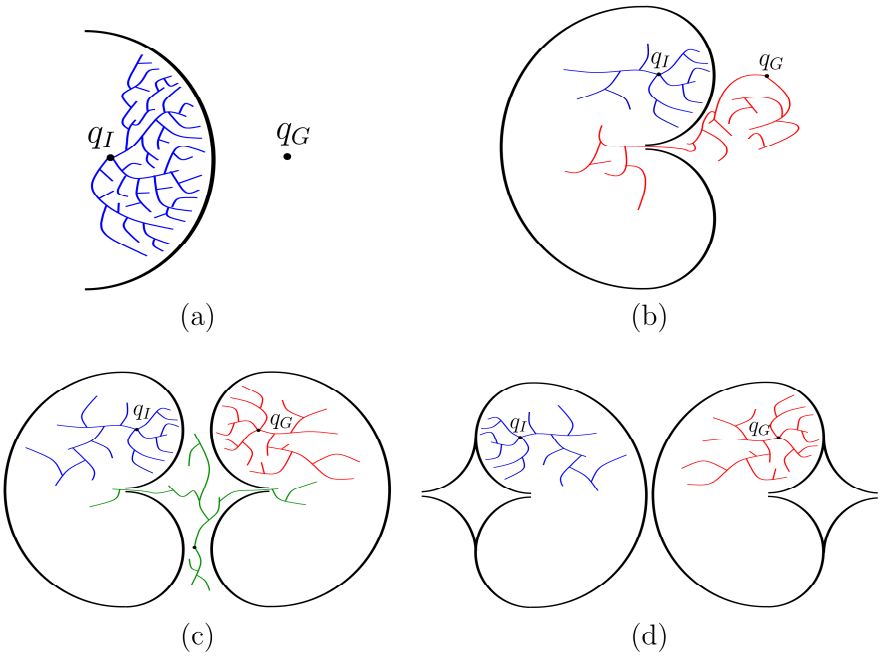


Figure 6.7: (a) shows a cavity with a unidirectional algorithm. (b) shows bi-directional algorithm within a *bug trap*, which is commonly used to illustrate high-dimensional obstacles regions, in the configuration space. (c) shows two bug traps, with a multi-directional motion planner. (d) shows an example of problem, that multi-directional algorithm will have difficulties solving. [LaValle, 2006, p. 219]

Unidirectional motion planning algorithms builds a single tree iteratively, as shown in figure 6.7 (a). Figure 6.7 (a) shows a cavity for a unidirectional tree. The motion planner needs to avoid the cavity, to build a path from the initial state q_I to the goal state q_G. The result may be, that the algorithm explores too many regions inside the cavity, and thereby will not be able to retrieve a solution, within the given solution time. In figure 6.7 (b) a *bug trap* is shown. A bug trap is used to illustrate high-dimensional obstacles regions, in the configuration space. Here will a unidirectional algorithm, shown as the blue tree, experience difficulties leaving the trap. Thus, this illustrates the reasons why unidirectional trees may have difficulties with high dimensional problems in the configuration space. [LaValle, 2006, p. 219]

Bi-directional algorithms are often more preferable, if it is not known whether the initial state, the goal state, or both lies within a complicated region of the configuration space, such as a bug trap. A bi-directional search is iteratively conducted, illustrated in figure 6.7 (b), between a blue tree being built from the initial state q_I, and a red tree being built from the goal state q_G. A path is created, when the two meet. In figure 6.7 (b) it is seen how bi-directional handles complicated configuration spaces better. [LaValle, 2006, p. 219]

If both the initial state q_I and the goal state q_G lies within two different bug traps, then it might make sense to try to grow a tree outside the bug traps, with the aim that this will be able to enter the bug traps in another direction, as seen in figure 6.7 (c). Multi-directional algorithms are more complicated to handle, and are not contained in OMPL. Figure 6.7 (d) illustrates a difficult problem for multi-directional algorithms as well. [LaValle, 2006, p. 219]

The RRT algorithm is the main basis for a set of other algorithms, which are included in OMPL. These are listed below:

- pRRT can do parallel processing on the RRT algorithm, to enhance processing time.
- LazyRRT combines the RRT algorithm with the "Lazy" function. Thus, the algorithm postpones the collision checking. This shall reduce processing time. The algorithm is further elaborated in appendix C.5.
- RRTstar can ensure that the vertices of the RRT algorithm are reached through a minimum-cost path. The algorithm is further elaborated in appendix C.6.
- LBT-RRT algorithm is a combination of RRT and RRTstar, where it can interpolate between the two. This gives a fast convergence, like the RRT, and with high quality, like the RRTstar. The algorithm is further elaborated in appendix C.7.
- T-RRT combines the RRT algorithm with cost-function, to guide the growth of the tree into minimum-cost paths. The algorithm is further elaborated in appendix C.8.
- RRTConnect consists of two parts, a bi-directional RRT algorithm and a Connect heuristic algorithm that extends more aggressively than the original RRT algorithm, and thereby rapidly explores the configuration space before uniformly covering it. It is designed for path planning problems without differential constraints. The algorithm is further elaborated in appendix C.9.

Applications for the Skill-based System

Common for all the algorithms are that they are single-query. As mentioned previously this can be beneficial in situations where only one query is needed or in a dynamic environment. Furthermore, the tree-based algorithms handles differential constrains better than its PRM counterparts do. [Karaman and Frazzoli, 2011]

RRTConnect is bi-directional and designed for high dimensional systems, such as systems composed of multiple manipulators. In addition, the algorithm demands a minimal amount of user-input, which corresponds well to the general idea about the SBS. In contrary to other RRT algorithms, the RRTConnect does not support differential constraints. It is however assessed that if differential constraint becomes a need, then algorithms that are more suitable will be chosen. Thus, it is estimated that RRTConnect is to be selected for a SBS.

6.3.3 Kinodynamic Planning by Interior-Exterior Cell Exploration

Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) is a tree-based planner, which is designed for handling of systems with complex dynamics. KPEICE utilises parallel processing, if multi core CPUs are available. A multiple-level grid-based discretisation is used to estimate the coverage of the configuration space, to help guide the extension of the tree. [Şucan and Kavraki, 2010]

In OMPL the algorithm is simplified by use of only one grid. It is important to set this set grid, and if none is set, then a default grid is chosen. [OMPL Rice University, 2014a]

The OMPL contains a set of algorithms, which are similar to KPIECE. These are listed in the following:

- BKPIECE is bi-directional, but is otherwise similar to the original KPIECE.
- LBKPIECE is in addition to BKPIECE, this algorithm includes lazy collision checking.
- PDST is a tree-based planner especially developed for systems with significant drift, severe under-actuated systems, and in "discrete system changes". Thus, it handles complex differential constraints. The growth of the tree is guided towards less explored areas by a user-specified projection. The algorithm is further elaborated in appendix C.10.
- EST is a single-query bi-directional tree-based planner. The algorithm is able to handle differential constraints. The growth of the two trees are guided towards less explored areas by a user-specified projection. The algorithm is further elaborated in appendix C.11.
- STRIDE is similar to EST and the latest addition to the tree-based planners that are able to handle differential constraints. STRIDE uses a so-called GNAT function to guide the development of the tree, thus STRIDE does not need to rely on a user specified projection grid. The STRIDE

algorithm is especially able to handle many degrees of freedom (10 or more). The algorithm is further elaborated in appendix C.12.

Applications for the Skill-based System

Common for all of these planners, except STRIDE, is that they rely on a projection of the grid being set as input to guide the tree development. If no projection is given, then a default value for the grid is attempted to be used. [OMPL Rice University, 2014a]

STRIDE has the best performance compared to the rest of this group of motion planners, with respect to processing time and number of solved queries for high dimensional problems, such as with multiple manipulators. [Gipson et al., 2013]

This implies that STRIDE will be the one to be selected for a skill-based system, if differential constraints is to be implemented. However, the PDST algorithm's ability to handle under-actuated systems may be useful, if motion planning has to be conducted for the gripper, since the RQ3 gripper is under-actuated.

6.3.4 Single Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking

Single query bi-directional probabilistic roadmap planner with lazy collision checking (SBL) is based upon the PRM algorithm, but this algorithm is single-query in contrast to PRM. As the name implies this algorithm is bi-directional, which means that a roadmap of two trees are rooted at respectively the initial and goal state. In addition, the algorithm applies lazy collision checking. By reducing the computational time, then more complex motion plans can be made, such as motion plans for multiple manipulators or complex environments with high dimensional problems. [Sánchez and Latombe, 2003]

The algorithm can be divided into three steps; a tree expansion, a tree connection, and a path testing.

The tree extend the roadmap by adding new sampling points to one of the two trees. First, a tree has to be selected. Then a sampled vertex in the tree has to be found, from which the new extension shall be made. To help deciding where to further expand the roadmap, the algorithm has a grid data structure implemented. The grid contains information about where previous vertices have been visited. The area in the grid that is least visited, has the greatest chance of being selected. In OMPL the grid is imposed on a projection of the configuration space. This projection needs to be set before using the algorithm, but if it is not set the planner attempts to set a default projection. [OMPL Rice University, 2014a]

The next part is the tree connection. If a sample in one of the trees come within a certain radius of a sample in the other tree, then a connection is made and a possible solution is found. The connection is called a "bridge". Thus, the bridge connects the initial state with the goal state. [Sánchez and Latombe, 2003]

The final step is to test whether the solution path is valid or not. If the path is valid, then the algorithm is done. If a collision is found, then the vertices and edges that are in collision are removed and a gap between the two trees is created. The trees can inherit each other's vertices and edges if a break is made in one tree. Thereafter the algorithm goes back to tree expansion to find a new solution. [Sánchez and Latombe, 2003]

There is one motion planner in OMPL, which resembles SBL:

- The parallel SBL (pSBL) enables the possibility to parallel process parts of the SBL algorithm, such as the collision checking, for faster solution response. [OMPL Rice University, 2014a]

Applications for the Skill-based System

The SBL and pSBL algorithm should benefit compared to the PRM, by also including lazy collision checking and bi-directional growth of the trees, although being single-query can affect the results. The SBL variants handles high dimensional problems, thus the algorithm is applicable for multiple manipulators.

If utilised, then the parallel SBL (pSBL) must be the one of choice, since it should enhance the performance with parallel processing.

6.4 Sub-conclusion

The main purpose of this chapter was to answer **Hypothesis 2** in chapter 3, with respect to a literature study about motion planning algorithms. The literature study was delimited to sampling-based motion planning algorithms, within the Open Motion Planning Library. It was chosen to divide the motion planning algorithms into five main groups, where they were presented and one was elaborated. The theory was used to assess the applicability for use of the motion planning algorithms in the Little Helper SBS.

Chapter 7

Benchmarking of Motion Planning Algorithms

This chapter concerns the benchmarking of the motion planning algorithms, described in chapter 6. The purpose for benchmarking is first clarified, followed by a description of how the benchmarking is conducted. The data from the benchmark is next presented and interpreted. Practical information about how the benchmark is created, is presented in appendix E.

7.1 Purpose

The main purpose of this chapter, in a combination with chapter 6, is to answer **Hypothesise 2** in chapter 3. The motion planning algorithms in OMPL are benchmarked to be able to make a selection of a few, or even a single motion planning algorithm, applicable for Little Helper 4. The motivation for selecting motion planning algorithms, is described in chapter 4, where the main reason originates from the wish to extend the capabilities of Little Helpers, without neglecting ease of usability. This can be achieved by having a few, or even a single motion planning algorithm, with predetermined parameters, capable of solving different problems.

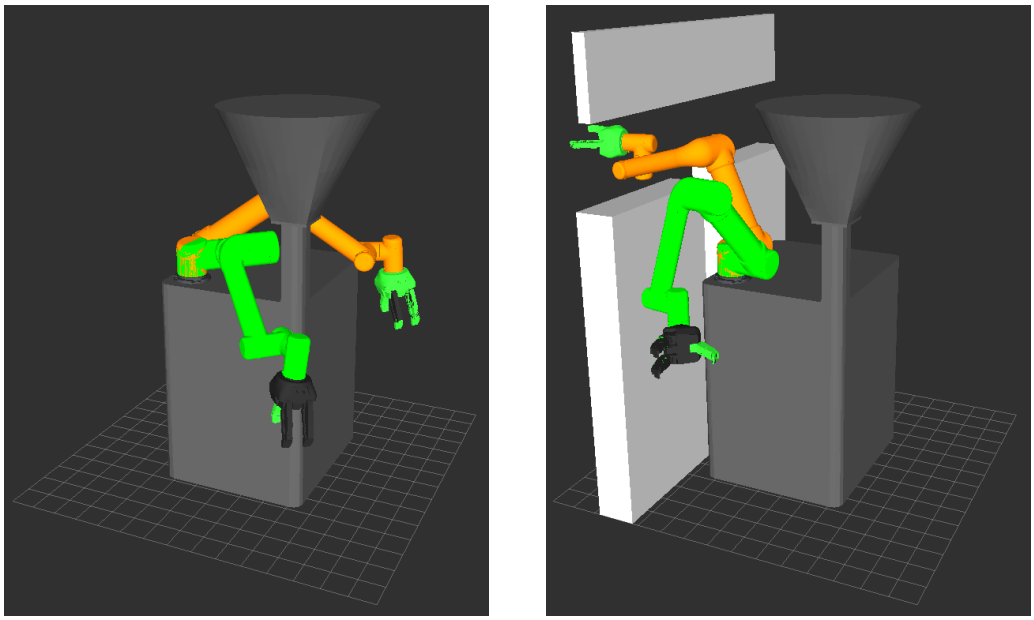
7.2 Design of Benchmarking Experiment

The aim is to find one or, a set of applicable motion planners. It will be easiest if solely one motion planner, without exception, is tested applicable, such that an end-user does not need to choose a motion planning algorithm for any given problem.

It is however challenging, and might not make sense to choose a single algorithm for all problems, because of the variety of problems. [Cohen et al., 2012]

In [Cohen et al., 2012] a generic infrastructure for benchmarking of motion planning algorithms is introduced. This infrastructure is implemented in MoveIt, where the aim is to make it possible for a developer to select the motion planning algorithm, which best suits a particular problem. One single problem cannot be identified for collaborative mobile manipulators, but general problems can be designed, to cover the most common problems. The infrastructure is described in appendix E, where this section describes how the benchmarking set-up is designed, conducted, and the choices made.

The motion planning algorithms are benchmarked in two different scenes. In the first scene, the manipulator has to move around the pan-tilt unit pole, described in appendix I, without collision. This scene is created to test how well motion planning algorithms handles relatively simple environments. The other scene is used to test how well the motion planning algorithms handles narrow passages. This is applicable for problems, where the manipulator places work-pieces into machines, such as a CNC machine. The scenes can be seen in figure 7.1a and 7.1b respectively.



(a) Benchmarking scene around the pan-tilt unit pole.

(b) Benchmarking scene with narrow passage.

Figure 7.1: The two scenes used for benchmarking of the motion planning algorithms. The manipulator has to motion plan around the pan-tilt unit pole in figure 7.1a; this scene is called benchmarking scene a. In figure 7.1b the manipulator has to motion plan into a narrow passage; this scene is called benchmarking scene b.

Most motion planning algorithms utilises parameters, which can be adjusted to alter their performance for a specific problem [Cohen et al., 2012]. To regulate the parameters for every motion planning algorithm in the OMPL is assessed to be a comprehensive task. Even though the parameters are not altered, it is assessed that applicable motion planning algorithms can be selected based on the default parameters. It is assumed that the default parameters are chosen with care by the developers of MoveIt and OMPL, hence the benchmarking of the motion planning algorithms are conducted with their default parameters. After a single or a few motion planning algorithms are chosen, the effort for regulating the motion planning parameters is less comprehensive, this have not been done in this project.

The benchmarking has been conducted with 40 runs for 17 of the 20 OMPL motion planning algorithms[1], with a maximum allowed time of 15 seconds for each run. It is important that the number of runs, also called sample size, is large enough to give a correct insight into the behaviour of the motion planning algorithms. [Walpole et al., 2007] recommends a sample size larger than 30, hence the choice of 40 runs.

7.3 Data Processing and Results

Statistical methods have to be applied to the benchmark data before drawing inferences. The log from the benchmarking contains data for 18 different properties, such as the total time and whether the query were solved. The raw data from the benchmarking is appended on the enclosed CD[2].

The first property evaluated, is whether the queries were solved. A bar chart for the percentage of solved queries is shown in figure 7.2, where the blue bars belongs to **scene a** and the red bars belongs to **scene b**. The chart shows that most of the motion planning algorithms have a solving rate less than 100%. If the percentage of solved queries for each scene are compared, it is seen that some motion planning algorithms fails more frequently for **scene b** compared to **scene a**. It is assessed desirable to choose algorithms that always solve the queries. RRTConnect, LazyPRM, LBKPIECE and BKPIECE fulfils this for both scenes, hence they are preferred if their performance regarding other properties are satisfying.

The next property examined, is the total time used for solving each query. It is chosen to visualise the total time used for solving the motion planning queries with Box-and-Whisker plots, which can be seen in figure 7.3 and 7.4. Box-and-Whisker plots illustrates multiple samples, for which it shows the median, variability, and the degree of symmetry. A description of how the Box-and-Whisker plot visualises the samples follows. The boxes shows the interquartile range of the sample, which is from the 25% to the 75% quantile of the sample. The median is shown within the boxes as a line cutting the boxes into two.

[1] Note that since it has not been possible to implement STRIDE, pSBL, and LBTRRT, then these will not appear in the benchmarking. Further information about the implementation of motion planners can be found in appendix F.

[2] See: <Benchmark/AroundThePole.txt> and <Benchmark/SmallSpaceTest.txt>

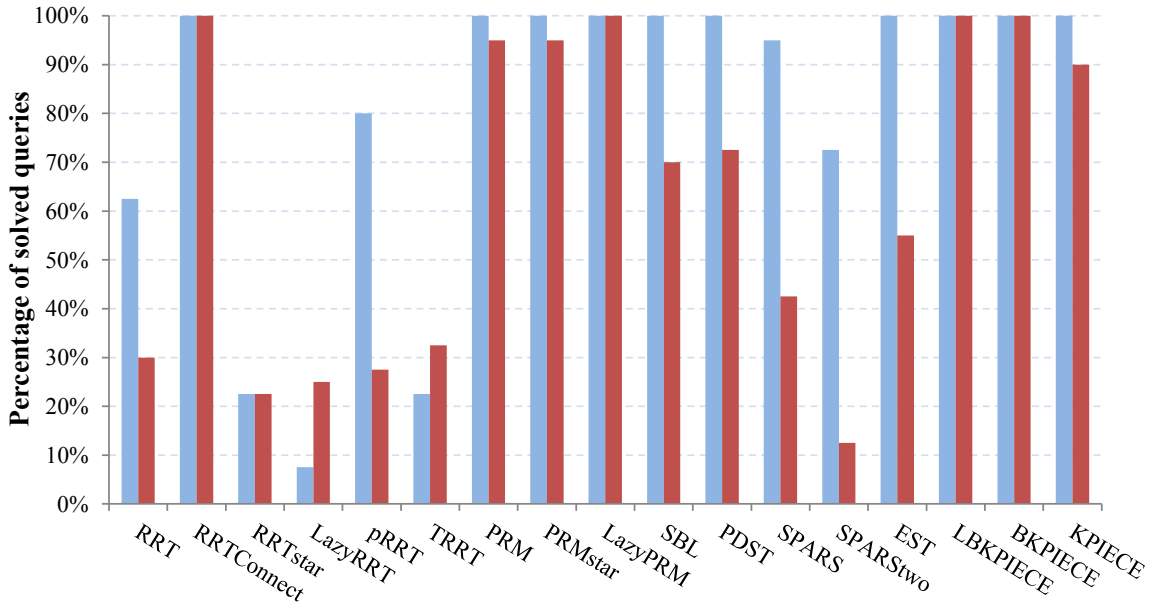


Figure 7.2: Solved motion planning queries for scene a and for scene b.

The range of the whiskers are calculated as a multiple of the interquartile range and visualises extreme observations within the sample. [Walpole et al., 2007, p. 236]. The total time can only be used as a relative comparison between the algorithms in this benchmark, because the time is dependent on the computer the benchmarking was conducted on.

Studying of the two Box-and-Whisker plots in figure 7.3 and 7.4 reveals that the motion planning algorithm LazyPRM is the quickest in both scenes followed by RRTConnect. Furthermore, both algorithms solved all the queries hence they are reliable. The plots shows also that RRTstar uses the maximum allowed time for each run. This is assessed to be because RRTstar is asymptotic optimal. This is expressed by, for each sample connected to the tree, all samples within a given radius is connected to the sample and the shortest path is chosen.

It is questionable whether the motion planning algorithm LazyPRM is as good as the data indicates. LazyPRM has, after the benchmark, been attempted validated by a simulated motion planning query, where the manipulator has to move from the top of the platform and down, along the side, as illustrated in figure 7.5. The fingers of the gripper collides with the Little Helper 4 platform, thus for this relatively simple query, the LazyPRM algorithm failed multiple times. It can therefore be questioned whether the benchmark shows the true behaviour of the LazyPRM algorithm. The same motion planning query has been tested with several other algorithms, including RRTConnect, PRM and PRMstar, which solved the query without collision.

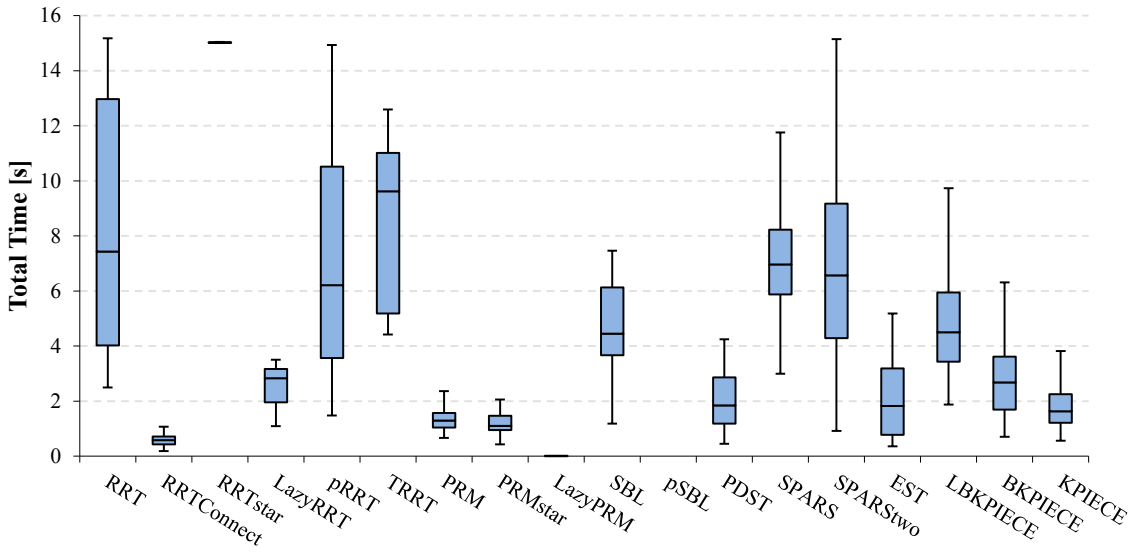


Figure 7.3: The total time used to handle the motion planning queries in scene a. Note that the solution time is relatively in-between, since they are fully depended on the used computer.

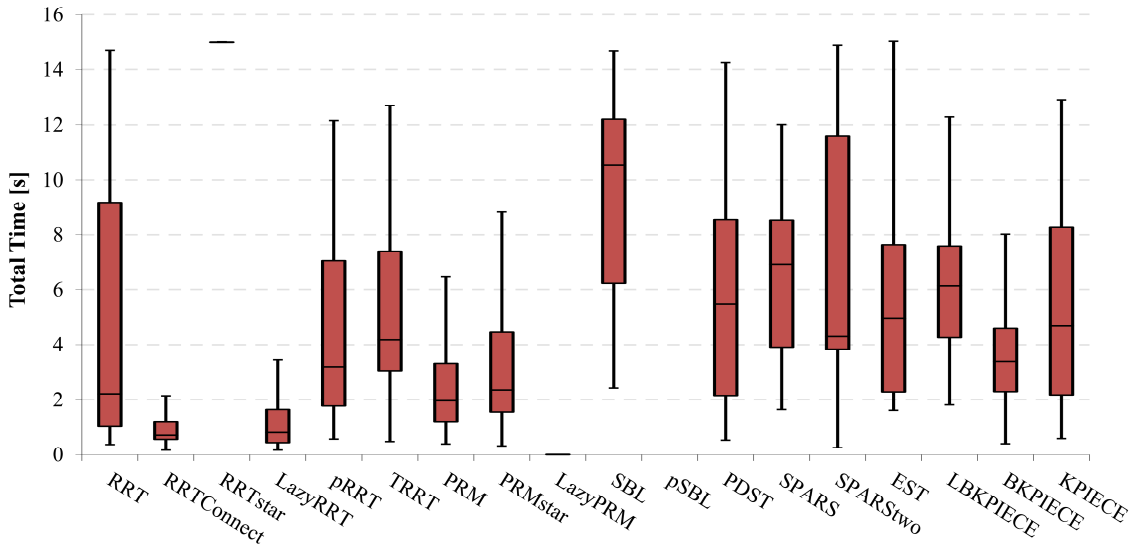


Figure 7.4: Visualises the total time used to handle the motion planning queries in scene b. Note that the solution time is relatively in-between, since they are fully depended on the used computer.

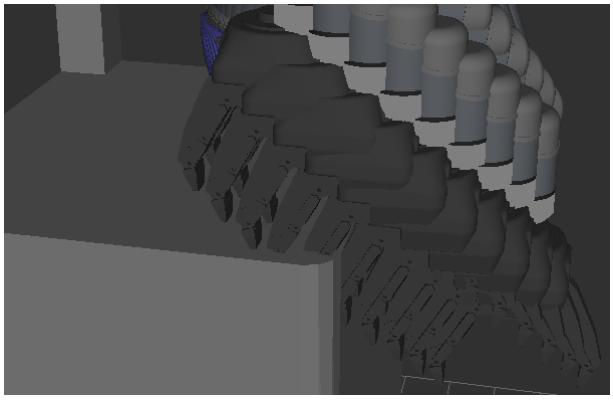


Figure 7.5: Image from simulation of a motion planning query with the LazyPRM algorithm which collides with the Little Helper 4 platform.

The last property tested is the path length. It has been discovered that the solutions sometimes are a long detour. It is undesirable because it is inefficient with respect to energy and time. Furthermore, it is unconvincing from the point of view of a user, especially if a close collaboration between manipulators and users is intended. A shorter path means less detouring, hence this property is worth taking in mind before selection of an algorithm.

The path lengths are visualised with Box-and-Whisker plots in figure 7.6 and 7.7 for **scene a** and **scene b** respectively. The path length is measured as the total movement for all joints in radians. The KPIECE algorithm and its variants produces paths much longer, than the rest of the algorithms, hence these algorithms are undesirable. To make it possible to see the details and compare the other algorithms, the KPIECE algorithms have been cut off.

If the LazyPRM is evaluated in figure 7.6 and 7.7 for the path length, it is seen that its path is shorter and with zero variance compared to the other algorithms. The LazyPRM is not a good choice based on the previous experiences. The second best motion planning algorithm, regarding path length is the RRTstar algorithm. In addition to those two algorithms, most of the algorithms performs almost equally well.

The results of this benchmark does not encompass every motion planning problem a Little Helper can encounter, hence the chosen algorithm might not be the best for motion planning problems, which significantly differs from the two tested. This can for example be motion planning with multiple manipulators. In addition, Little Helper's vary in manipulator type, hence the mechanical configurations differs. Thus, the result will solely indicate a solution for Little Helper 4. If it is desired to have a more general formalism, within motion planning for all Little Helpers, then a comprehensive benchmarking has to be conducted.

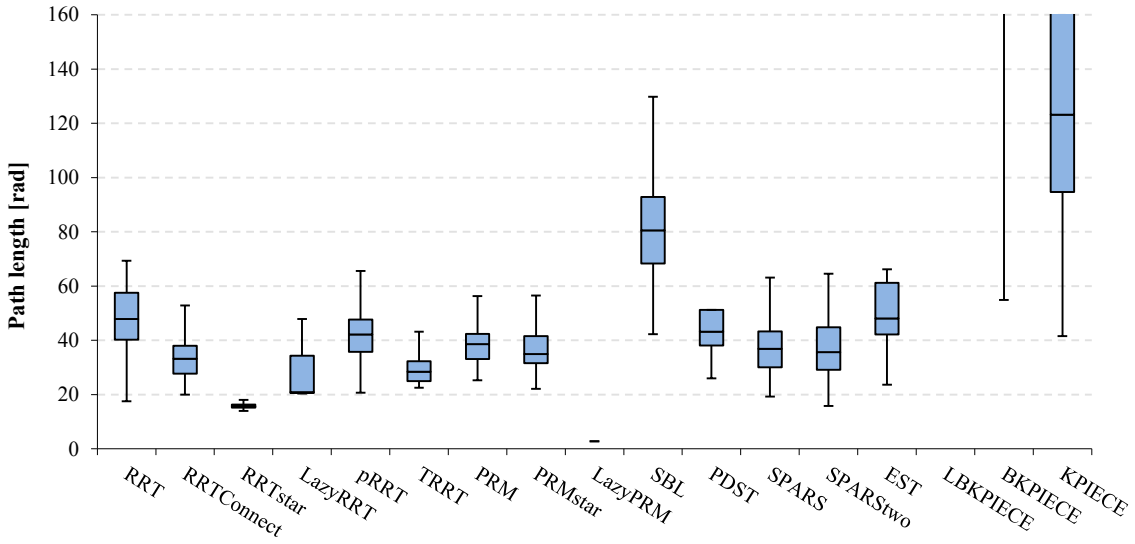


Figure 7.6: The path lengths for the motion planning queries in scene a. The KPIECE algorithms are cut-off to make it possible to evaluate the details of the other algorithms.

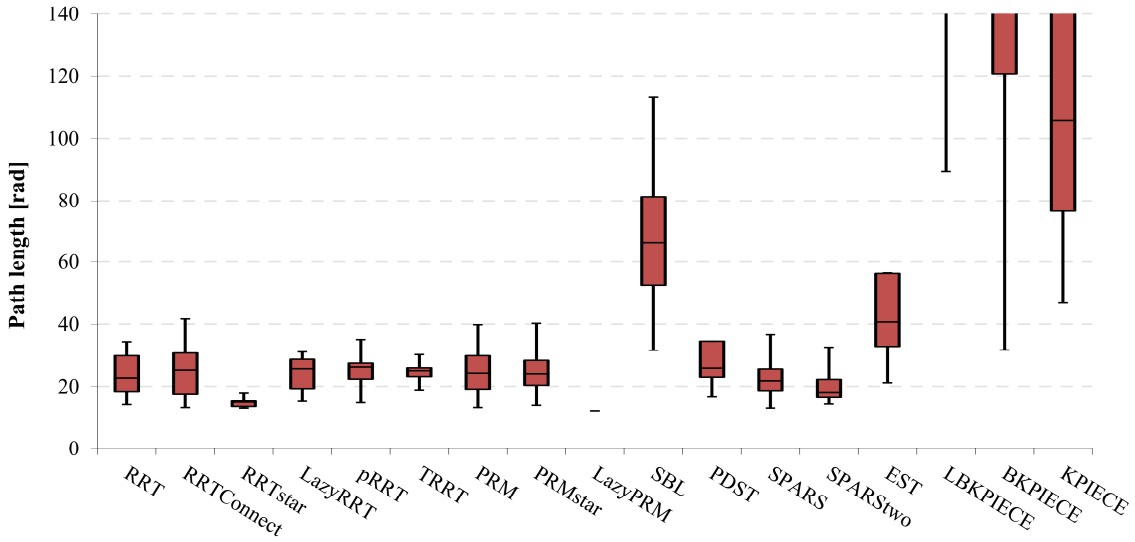


Figure 7.7: The path lengths for the motion planning queries in scene b. The KPIECE algorithms are cut-off to make it possible to evaluate the details of the other algorithms.

7.4 Result Analysis

Based on the results presented by the benchmarking in this chapter, and the study conducted about motion planning algorithms in chapter 6, a single or a few motion planning algorithms have to be selected. The selection will be based on three parameters: reliability, solution time, and path length.

Reliability

It is preferable to select a reliable algorithm, hence the selection narrows down to those algorithms which have a success rate of 100% in figure 7.2. Four algorithms complies:

- RRTConnect
- LazyPRM
- LBKPIECE
- BKPIECE

It is observed that three (RRTConnect, LBKPIECE, and BKPIECE) out of the four motion planner algorithms that performs best, concerning finding a solution, are bi-directional. This corresponds with the theory about bi-directional, stated in section 6.3.2. Additionally it can be seen in figure 7.2 that the PRM variants in general performs fairly, although LazyPRM obviously is the best. This is likewise assessed to be reasonable, since PRM and its variants covers the entire configuration space. The experiences with LazyPRM regarding collision have to be taken into account.

Solution Time

The solution time is also of concern during motion planning calculations. If the four algorithms, mentioned above, are compared with respect to the total time used for solving a motion planning query, seen in figure 7.3 and 7.4, it is observed that the LazyPRM algorithm outperforms the others, where the second best algorithm is the RRTConnect algorithm.

Again, it is questionable whether the results for LazyPRM is valid. Comparing the results within the PRM variants in figure 7.3 and 7.4, with the study conducted in 6.3.1, it is reasonable that the LazyPRM is faster than its siblings are, since the collision checking is postponed. A solution-time close to zero, with no notable variance, is however assessed to be unlikely. By combining the observations made in the benchmarking, with the motion planning study, it is assessed that the algorithm is not functioning correctly in MoveIt at present.

The second best algorithm, concerning minimal time consumption, is the RRTConnect algorithm. This corresponds well to the motion planning study in chapter 6, being a fast tree-based planner, since it

extends more aggressively, than the original RRT algorithm, which results in a rapidly exploration of the configuration space.

Path Length

Thirdly, the motion planner's path lengths are compared. If LazyPRM is ignored, then the result indicates that the RRTConnect algorithm performs best, with the shortest path, as seen in figure 7.6 and 7.7.

The KPEICE, and its siblings, behaves poorly concerning the path length. Based on the motion planning study in chapter 6, it is assessed to be because a default grid is projected on the configurations space. Thus, nothing guides the trees of the algorithms, and thereby the result may be relatively large path lengths.

Thus, the immediate choice for Little Helper 4 is RRTConnect, based on the performance criteria and with the reservations in mind, given throughout this chapter.

7.5 Sub-conclusion

The purpose of this chapter was to answer **Hypothesis 2**, stated in chapter 3. This was done by conducting a benchmark on the selection of motion planning algorithms available and functioning in OMPL. To this, the method utilised for the benchmarking, was introduced. Next, the design of the experiment was enlightened, including reasoning for the choices made, before conducting the benchmark. Here two scenes was chosen to replicate some of the environments for Little Helper 4. Next, the data was presented in a statistical manner, to make the result clear. It was realised that LazyPRM in general outperformed most of its competitors, but after a more thorough examination of LazyPRM, it was assessed that the results of LazyPRM might be invalid. Additionally an assessment on the validity of the results, based on the two scenes, was given.

Finally an analysis of the results was conducted, on the four best performing motion planners, where the literature study from chapter 6 was utilised to verify the credibility of the results. Here it was again confirmed that the results of LazyPRM was invalid, thus these was ignored. The benchmarking ended up selecting RRTConnect as the most applicable motion planning algorithms, based on the two scenes conducted with Little Helper 4.

Chapter 8

Implementation of Motion Planning in the Little Helper 4 Software Framework

This chapter concerns the implementation of motion planning in the Little Helper 4 software framework, where the implementation is based on the short-term concept proposal, described in section 4.3, and the software selection in chapter 5. The implementation is a proof of concept for the short-term concept proposal. The implementation is presented throughout this chapter, which is supported by appendix A, regarding the set-up of MoveIt, and appendix B, regarding modification of the UR5 proxy and driver. The implementation is the foundation for the additional work, described in chapter 9 and 10.

8.1 Purpose

The purpose of this chapter is to answer **Hypothesise 3**, in chapter 3. The basis for the implementation of motion planning in the Little Helper 4 software framework is the short-term proposal, conceived in section 4.3. The implemented motion planning software is based on the selection, described in chapter 5. The implementation is verified based on a system test, where Little Helper 4 is provided with a sequence of skills, via teaching, which must be executed. The skills shall include a motion planning request. The implemented motion planning software tool, in the software framework of Little Helper 4, is verified with a sequence of executed skills, without collision.

8.2 Motion Planning in the Little Helper 4 Software Framework

Motion planning is implemented in the Little Helper 4 software framework, between the skill level, and proxy level. The structure of the implementation is shown in figure 8.1. The motion planning functionalities are in the figure illustrated with *Motion planning level*. The motion planning software tool of choice in the motion planning level is *MoveIt*, based on the software selection described in chapter 5. The implementation of motion planning for the UR5 manipulator has not affected the implementation of the RQ3 gripper. The changes to the original Little Helper 4 software framework are related to three different levels; the *skill level*, the *motion planning level*, and the *proxy level*. The changes to each of the three levels are described in the following.

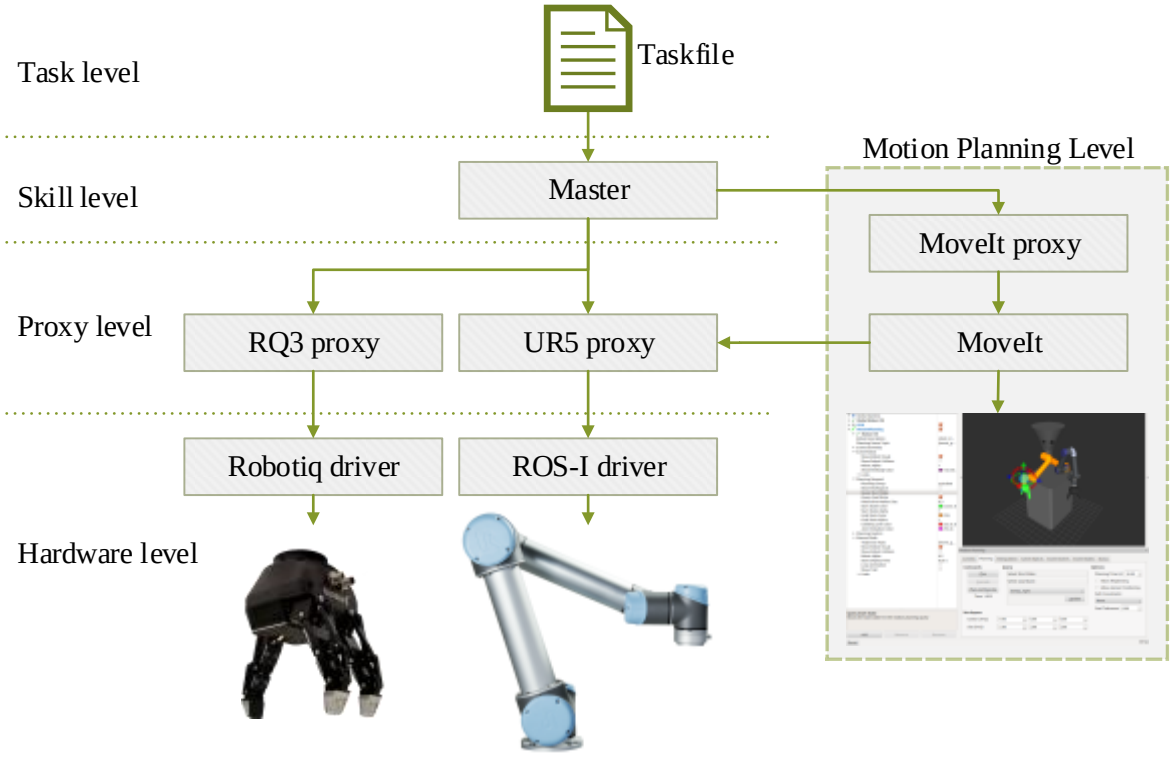


Figure 8.1: The implemented motion planning in the Little Helper 4 software framework.

Skill Level

The changes in the skill level concerns device primitives and addition of parameters. Device primitives invoking motion, are changed from *move* to *motionplan*. The parameters for motion planning is added to the task file during teaching of skills, which contains a motion planning device primitive. The parameters are "read" from the task file during execution.

Motion Planning Level

The structural change to the overall framework, compared to the original Little Helper 4 software framework, is limited to the addition of the motion planning level, as shown in figure 8.1. Since motion planning is solely conducted for the articulated manipulator, then a motion planning handler, proposed in the short-term proposal in chapter 4.3, becomes redundant, hence it is not implemented.

The motion planning level consist of a motion planning proxy (MoveIt proxy), the MoveIt program, and a visualisation tool (Rviz). The MoveIt proxy handles the connection from the skill level to MoveIt. The proxy translates motion planning requests, such that it follows the syntax of MoveIt.

The output from MoveIt is a trajectory. MoveIt utilises the ROS visualisation tool Rviz, to show a graphical representation of the manipulator, the environment, and planned paths [Hershberger et al., 2014]. A trajectory, from MoveIt, can be executed either by use of a simulated robot controller or a real manipulator. The simulated controller can visualise the position and velocity of a trajectory [Edwards, 2014].

Proxy Level

The proxy level concerns hardware specific proxies and drivers. The UR5 proxy and driver, from the original Little Helper 4 software framework, has been exchanged, to be able to cope with trajectories. The driver originates from the ROS-Industrial Universal Robots package[1]. The changes necessary to make the driver compatible with the Little Helper 4 software framework are described in appendix B.

8.2.1 MoveIt Proxy - Motion Planning Level

The MoveIt proxy translates a motion planning request send from a skill to MoveIt. A request consists of a query and a set of parameters. A query consist only of a goal state, since the motion planning is conducted with the actual position of the manipulator, as the initial state. The proxy only handles

[1]The ROS-Industrial Universal Robots package: `https://github.com/ros-industrial/universal_robot`

Cartesian pose as input for goal state, thus not joint configurations. This is because skills in the Little Helper 4 software framework are based on Cartesian poses, as an effort to make the motion planning request generic [VT3-2013, 2013].

The motion planning parameters from the task file are passed along, with the planning request to MoveIt. This is supplemented with parameters specified in the MoveIt proxy. The parameters includes specification of tolerances for goal position and goal orientation, which are specified to 10mm and 0.01rad respectively. The tolerances are specified relative coarse, compared to an industrial task, which should have position tolerances of ±1mm [Hvilshøj et al., 2012]. This is done throughout the development phase, to reduce the planning time. No hurdles have been encountered, obstructing a lowering of the tolerances. All parameters regarding motion planning, which are specified through the MoveIt proxy and from task files, are described in appendix A.

Not all parameters are specified through the task file, as proposed in section 4.3. Parameters, assessed to be changed often, during the development phase, are specified in the task file. This is to ease the development phase and to prove that motion planning can be done, based on parameters from a task file. The remaining parameters are specified in the MoveIt proxy, which can be added to the task file if desired.

Process constraints, such as limited orientations of the tool, are suggested in the short-term concept proposal, in chapter 4.3. No process constraints are defined as a part of a planning request at present. It is also proposed to categories the motion planning algorithms, such that they can be chosen to appropriate tasks, in section 4.3.2. However, since solely one motion planning algorithm was chosen in chapter 7, based on the two tested motion planning problems, then this is not necessary. More motion planning problems may need to be benchmarked, in order to provide a variety of motion planners.

8.2.2 MoveIt - Motion Planning Level

Motion planning is handled by MoveIt based on a query, parameters, and an environment representation. The used motion planning algorithms are based on OMPL, described in chapter 6.

The practical implementation of MoveIt in the Little Helper 4 software framework, is conducted according to the encapsulations shown in figure 8.2. The encapsulations are conceived based on the architecture of MoveIt, described in section 5.4 on page 43. The encapsulations are illustrated with five main parts; an environment representation, a collision checking, a motion planning, a post processing, and an execution. Environment, motion planning, and execution are described in the following sections, whereas post processing is described in appendix D and collision checking is briefly described in section 5.4.

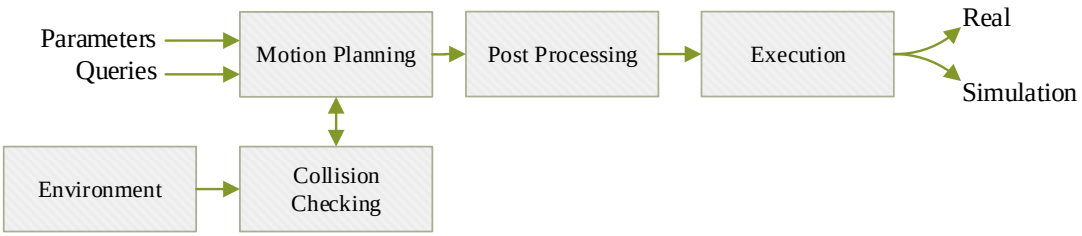


Figure 8.2: Encapsulations used to described the implementation of MoveIt in the Little Helper 4 software framework.

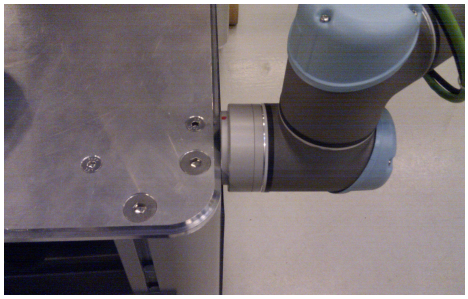
Creation of a Static Environment

The environment representation is the foundation for collision checking; hence the environment shall include all objects, which the manipulator is not allowed to collide with. The environment for this implementation is composed of model-based representations of the components of the Little Helper 4. This includes the UR5 manipulator, the RQ3 gripper, and the platform. A general representation of the environment is described in appendix G.

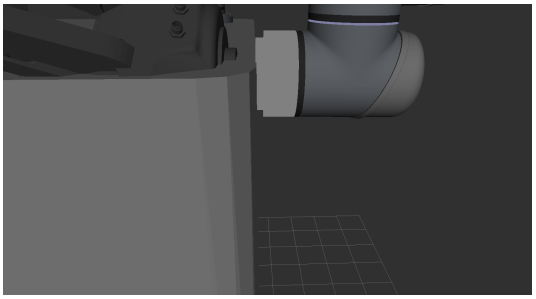
Objects being manipulated must be taken into account, to ensure a collision free trajectory, during execution. An object can be assigned to the manipulator in MoveIt, and thereby be considered during the motion planning [MoveIt, 2014b]. Inclusion of the object being manipulated will also remove the need for approach and leave points, during execution of pick and place skills, as stated in section 4.3. This has not been done in this project, and objects are accounted for by maintaining approach and leave points during execution of pick and place skills.

Calibration of a Static Environment

The model of the platform is calibrated relative to the UR5 manipulator, based on measurements of the translation of the real UR5 manipulator. The UR5 manipulator is presumed to be perpendicular to the top of the platform. The rotation is determined from calibration, of where the real UR5 manipulator is placed in a position, as shown in figure 8.3a. The model of the manipulator and platform is adjusted, to obtain the same relative placement, as shown in figure 8.3b. This is repeated until the error is negligible. The needed precision of the calibration is dependent of the size of the collision model; the bigger the collision model, the coarser calibration is allowed. To improve the calibration, the platform could have been designed with well-defined haptic calibration points. These points could be usable for a haptic calibration, as described in section 4.3. This can also be used to calibrate the manipulator relative to the platform.



(a) A calibration point for the real manipulator.



(b) Corresponding calibration point for the model.

Figure 8.3: Calibration of the UR5 manipulator relative to the platform.

Motion Planning - Path Calculation

Motion planning is conducted, based on a planning request from the MoveIt proxy. Five motion planning algorithms are manually implemented in MoveIt, in addition to the 12 pre-implemented algorithms. The manual implemented algorithms are:

- LazyPRM
- PDST
- SPARS
- SPARStwo
- pRRT

The process of implementing them into MoveIt is described in appendix F. The motion planning is conducted, accordingly to the description in chapter 6. The motion planning software tool of MoveIt is able to conduct post processing in multiple manners; a potential post processing regarding creation of a trajectory based on the path is described in appendix D, whereas post processing of a path in OMPL is described in appendix A.

Execution of Trajectories

MoveIt compares the actual execution time, for each manipulator configuration of a trajectory, to the desired time of each configuration, to ensure that the manipulator follows the trajectory. If the execution time deviates from the specified time, then MoveIt returns an error, which do not affect further execution. The error can be created to be used as stopping criterion for the manipulator, but this has not been done for this implementation. The time constraint of a trajectory affects the velocity and acceleration of the manipulator. Timing becomes important in a changing environment, for example in an environment with

two manipulators, as described in chapter 9. The entire trajectory is send via a ROS actionlib. Feedback is send to MoveIt, containing the current joint configuration of the manipulator, by either the simulated or the real manipulator. A trajectory can be simulated before execution on a real manipulator.

Execution on a Simulated Robot Controller

The response of the simulated controller is not the same, as of the real controller of the UR5 manipulator. This is because the two controllers do not contain the same control algorithms and models of the manipulator. The simulated robot controller is a part of the "industrial_robot_simulator package[2]. The simulated motion is based on a linear interpolation of the position and time, for each part of the trajectories to be executed, thus the simulation neglects all kinematics[3] and dynamics of the manipulator, which can cause deviation between the simulated and real trajectory. The potential deviation from the real response has no effect during this development. This is because the simulated response has only been used for visual verification of the planned trajectory and functionality test of the system.

The simulation enables for test of planned trajectories, without being connected to the real manipulator. It is thereby possible to test the functionalities of the motion planning ability of the system, without the real hardware.

Execution on the UR5 manipulator

Execution on the real manipulator is conducted by transferring the calculated trajectory to the proxy, which handles the communication to the driver. The driver creates a cubic interpolation, between the points of the trajectory, to obtain a smoother path, but this also deviates from the intended path. This deviation is neglected due to the size difference, between the real objects and corresponding meshes used during collision checking. The driver publishes the joint states of the manipulator, which MoveIt uses to visualise the real manipulator and to check the timing of the movement. The driver is based on the ROS-Industrial Universal Robots driver[4]. The changes to the driver, and the creation of the UR5 proxy, are described in appendix B.

[2]The package can be found at: `https://github.com/ros-industrial/industrial_core`
[3]Note that the visualisation tool (Rviz) utilises kinematics.
[4]The package can be found at: `https://github.com/ros-industrial/universal_robot`

8.3 Test and Verification

The implementation of motion planning in the skill-based system (SBS) has been verified by a system test. Prior to this test, several test of sub-parts of the system are conducted. This is done to determine different parameters.

The system test consist of a teaching phase and execution phase. Both phases are included in the verification, because the teaching phase is a central part of a SBS. The test was created, such that a collision would occur if the test was conducted with a basic move command - that is a movement without motion planning. The test is shown in figure 8.4, where an object is picked, from the Little Helper 4 platform, and placed next to the platform on a chair. A sequence from the test is shown in figure 8.5.

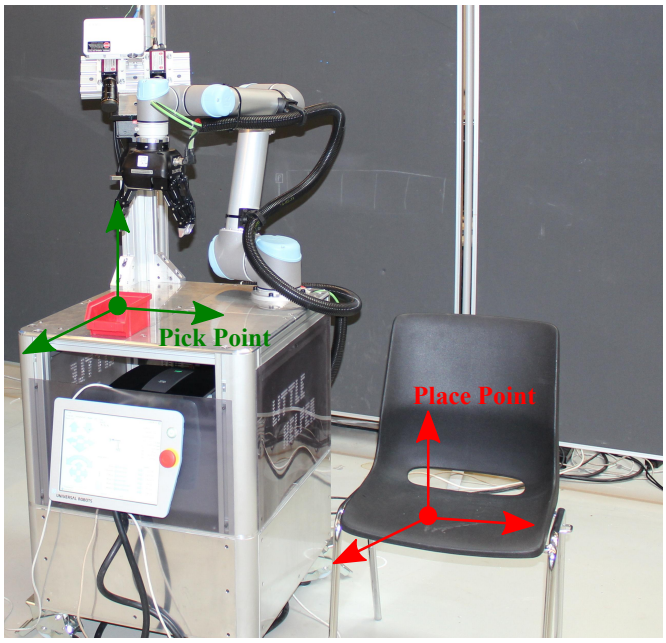


Figure 8.4: Schematic overview of the verification test. The red box is picked from the platform and placed on to the chair.

The Little Helper 4 software framework was taught a pick skill and a place skill during the teaching phase, to create a task file. The task file was executed through the software framework, which invoked multiple motion planning request. The task file was executed several times, to ensure the result was representative. The motion planning was successfully executed through the framework and the manipulator followed collision free trajectories. A video of the test is appended on the enclosed CD[5]. During the testing phase, the chair, which marked the place point, was pushed over by the manipulator. The chair was not modelled in the environment, and therefore could not be accounted for by the motion planner,

[5]See <Video/LH4 motion planning.mp4>

during the calculation of the trajectory. This underlines the importance of that a sufficient environmental representation, not only being limited to a model of the Little Helper 4 mobile manipulator. An improvement of the representation of the environment can be, by use of sensors, which is described in chapter 10.

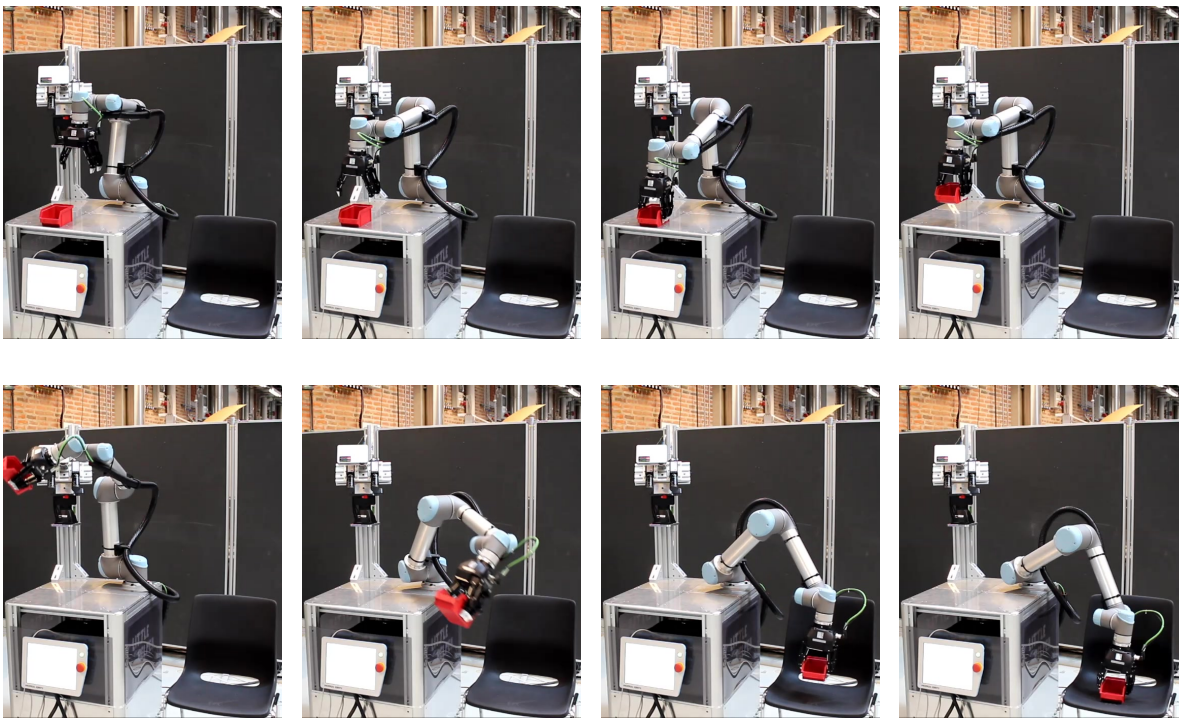


Figure 8.5: Sequence from the verification test. The red box is picked from the platform and placed on to the chair.

Error handling was not a concern during the development of the framework. This causes the system to continue, even though a motion planning attempt fails. An attempt can fail due to the complexity of the needed trajectory, collision or "unlucky" planning attempts; both the motion planning algorithms and kinematics solver creates numerical approximations, which results in small changes can occur in the output between different attempts.

8.4 Sub-conclusion

The Little Helper 4 software framework was developed, based on the short-term proposal, described in chapter 4 and to answer **Hypothesise 3** in chapter 3. Motion planning was implemented as a proof of concept. The developed framework was based on the original Little Helper 4 software framework. The Little Helper 4 software framework contains all functionalities from the original software framework.

The software framework was tested to work with motion planning, by execution of trajectories, without collision.

For the implementation to include all aspects from the short-term proposal, some still needs to be included, as for example error handling, calibration, and handling of constraints. A sensor-based environment representation are neither implemented in the Little Helper 4 software framework, but the implementation in MoveIt is described in chapter 10. The capabilities, which was not implemented, are refinements, and not directly connected to proving the concept.

Part III

Additional Work

Chapter 9

Motion Planning for Multiple Manipulators

This chapter concerns the development and experiences gained from performing motion planning with two manipulators, with overlapping work envelopes. This is carried out by adding Little Helper 3 to the environment described in chapter 8. In this chapter, the described set-up concerns creation of a dual manipulator model, motion planning, and communication to the respective manipulators. The implementation is verified with a test.

9.1 Purpose

This chapter is created to answer **Hypothesise 4**, in chapter 3. The purpose is to implement motion planning for a dual manipulator set-up, which in this instance includes Little Helper 3 and Little Helper 4. This is to clarify possibilities and difficulties for motion planning with two manipulators, and ultimately to be able to implement it in the skill-based system (SBS), which is outside the scope of this project.

The mechanical development of mobile manipulators have been focused towards a human-like design, regarding number of articulated manipulators and degrees of freedom [Ott et al., 2006]. This is assessed to be due to the flexibility and versatility of the human body. One way to obtain a more human-like design is utilisation of two manipulators. Two manipulators can be used to conduct two individual operations, in an overlapping work envelope or be used to conduct tasks requiring bimanual capabilities. A demonstration task of individual tasks with overlapping work envelops, conducted in cooperation between Little Helper 3 and Little Helper 4, is described in appendix J. These considerations are the foundation for the following work, regarding motion planning for Little Helper 3 and Little Helper 4. The work is solely focused on motion planning and not cooperation between the mobile manipulators.

9.2 Design of Experiment

The differences between motion planning for one manipulator, described in chapter 8, and multiple manipulators, seen from a system integrations point of view, are the creation of an appropriate environment representation and the communication to and from the controllers. Using the terms and encapsulations from chapter 8, the structure, shown in figure 9.1 is created. *MoveIt* contains the same functionalities, as described in section 8.2.2. The differences is the environment and the interface, which is elaborated in the following.

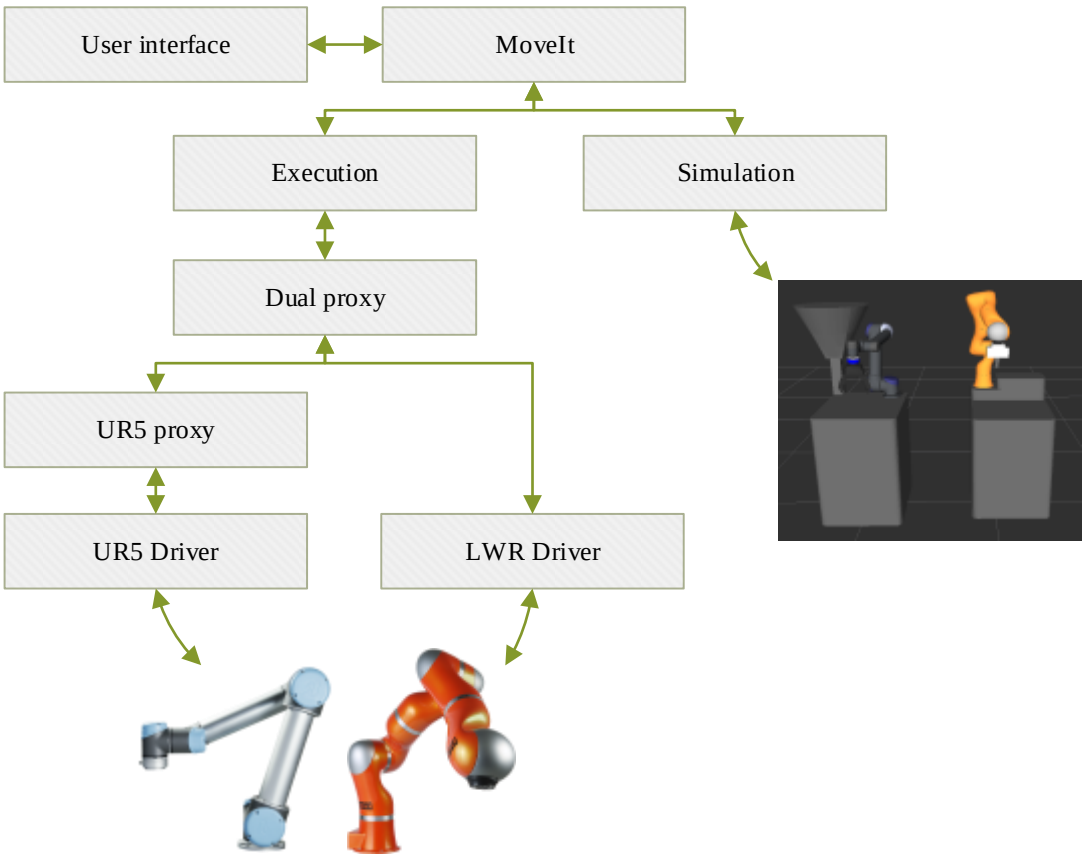


Figure 9.1: Structural overview of the software framework regarding motion planning for two manipulators.

The hardware utilised for the dual manipulator set-up is Little Helper 3 and Little Helper 4. Each consists of a platform and an articulated manipulator. Little Helper 3 consists of a KUKA LWR manipulator (LWR). The hardware of Little Helper 3 is further introduced in appendix G.3. Like with Little Helper 4, the platform of Little Helper 3 is solely used statically in this test, thus the platform is treated as a fixed joint.

9.2.1 Environment for Two Manipulators

The environment for the dual manipulator set-up consist only of a manipulator model and not a scene. The model is based on a merging of individual models, for each of the manipulators. The creation of the model is described in appendix G.

The manipulators are calibrated relative to their platform, like described for Little Helper 4 in chapter 8. The platforms are calibrated relative to each other, by measuring their relative position. No changes occurs to the environment, beside the movement of the manipulators, and the environment is therefore considered static.

9.2.2 Motion Planning with Two Manipulators in MoveIt

Motion planning is based on planning groups within MoveIt. A planning group is a collection of joints or links, which is part of the same kinematic chain. Three planning groups are used in the dual manipulator set-up; one for each of the individual manipulators, and one for both manipulators at once. This makes it possible to conduct motion planning for each of the individual manipulators and for both manipulators at once.

9.2.3 Communication

A *dual proxy* handles the distribution of the trajectories from MoveIt, to both the UR5 and LWR manipulator. A trajectory is distributed to one of the manipulators, or both of them, depending of the planning request.

The utilised UR5 proxy and driver, are the ones introduced in section 8.2.2, and elaborated in appendix B. The driver for the LWR manipulator is created by *The Robotics and Automation Group* at *Aalborg University*, to obey the syntax of MoveIt. The LWR driver does not obey time constraints, which are specified in a trajectory. This means that the LWR manipulator executes the path of the trajectory with a predefined velocity. This can cause timing issues, if the paths of the two manipulators are intersecting. Due to this shortage, no considerations have been made regarding timing, between the two controllers, to ensure that the trajectories are executed simultaneously.

9.3 Proposal for Interface to the Little Helper 4 Software Framework

The interface to the system, during execution of motion planning, for two manipulators, is directly through the GUI tool Rviz. This interface is utilised, since the purpose is to implement the functionalities within motion planning for multiple manipulators. Implementation of the dual manipulator set-up

into the software framework of Little Helper 4 will require changes to the following parts of the framework, shown in figure 8.1.

- **MoveIt proxy**
  - Addition of another goal query, which will be of the same structure as of the one for the UR5 manipulator.
- **Skill level**
  - Addition of skills supporting dual manipulator movements. This can for example be pick with two manipulators.
  - Addition of a teaching functionality for the LWR manipulator. This can be a hybrid of existing individual teaching functionalities for both the LWR and UR5, to suit the Little Helper 4 software framework.
  - Addition of a new teaching abilities and skills will require a change in the existing functionalities, regarding write and read of tasks.

The work with creation of a motion planning set-up has not revealed any complications, for obstructing the implementation of a dual manipulator set-up, in the Little Helper 4 software framework.

9.4 Test and Verification

The implementation of motion planning for multiple manipulators is verified, based on a functionality test. The test is composed of a set of joint configurations for both the manipulators. The configurations are executed, with movement of the individual manipulators and both at once. The configurations are placed, such that the direct paths between them are intersecting. This is done to ensure that the test cannot be conducted successfully without motion planning. A sequence from the test is shown in figure 9.2.

The functionality of dual manipulator set-up was successfully tested. This was even though the LWR manipulator could potentially deviate from the specified trajectory, and thereby result in a failed test. A video of the test is appended on the enclosed CD[1].

[1] See <Video/Dual motion planning.mp4>

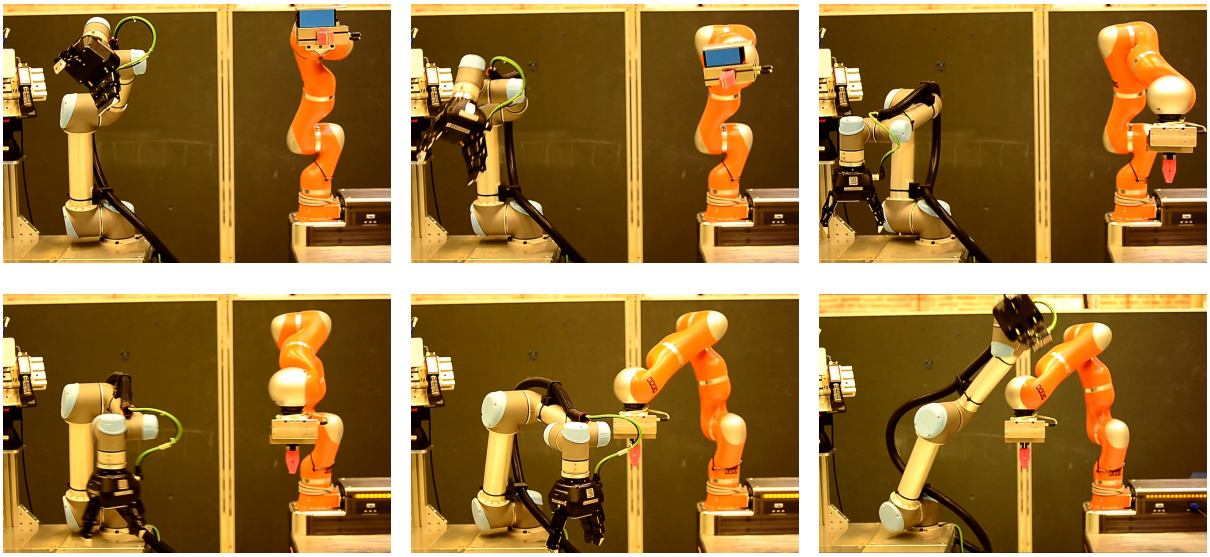


Figure 9.2: Series of dual manipulator motion planning of Little Helper 4 (left) and Little Helper 3 (right).

9.5 Sub-conclusion

An environment including both the Little Helper 3 and Little Helper 4 was created, as the basis for motion planning for multiple manipulators, to answer **Hypothesise 4** in chapter 3. A dual proxy was used to distribute the trajectories to the manipulators. To create the connection to the LWR manipulator, a driver supporting a specified interface was developed for the project group. The LWR driver does not support the time specification, associated with a position in a trajectory. The velocity can therefore deviate from the intended velocity and can cause a collision between the manipulators, if paths of the manipulators are intersecting.

No barriers were encountered regarding addition of the Little Helper 3 to the Little Helper 4 software framework, which also was proposed. This would enable motion planning for multiple manipulators through the SBS. Motion planning for a dual manipulator set-up, consisting of a UR5 and a LWR manipulator was successfully tested to verify the motion planning functionality.

Chapter 10

Motion Planning in a Sensor-Based Environment

This chapter concerns the development and experiences gained from introducing sensor-based information into the environment of Little Helper 4. The implementing of a RGB-D sensor into MoveIt is presented in appendix H.

10.1 Purpose

This chapter shall answer **Hypothesis 5** in chapter 3 about the desire for motion planning in a sensor-based environment. Thus, this chapter shall enlighten the possibilities and difficulties, encountered during implementation of a scene with sensor input. The practical implementation is elaborated in appendix H.

A step towards truly collaborative mobile manipulation is the integration of sensor-based motion planning. This reduces the necessity of a static model and enables motion planning to compensate for a changing environment. A changing environment is for example moveable obstacles, like pallets or the presence of humans, within the work envelope of a mobile manipulator. The safety requirements, because of humans in the work envelope, induce increased requirements regarding reliability and update frequency of the sensor input and trajectories, compared to movable obstacles. No special precautions regarding human safety have been taken into account throughout this work. This choice is enhanced by the fact that the UR5 manipulator is allowed to work without any safety precautions.

10.2 Design of Experiment

The experiment consist of a fixed RGB-D sensor, pointed towards the top of the platform. The entire top of the platform is inside the work envelope of the UR5 manipulator. The sensor input is utilised to create a scene, which creates the environment representation. This is further used for collision checking. To acquire the 3D data for the scene, an off-the-shelf RGB-D Primesense Carmine 1.09 sensor and a mid-range laptop is utilised. The RGB-D sensor is mounted on the pan-tilt unit pole, mentioned in appendix I. The RGB-D sensor is placed approximately 400 mm above the top of the platform. The range of the Primesense Carmine 1.09 is 350-1400mm, further specifications are to be found in appendix H.

10.2.1 Calibrating the RGB-D Sensor

To be able to transform, and thereby project the 3D data correctly, a virtual sensor has to be placed in the model-based representation of the environment. This is further elaborated in appendix H.

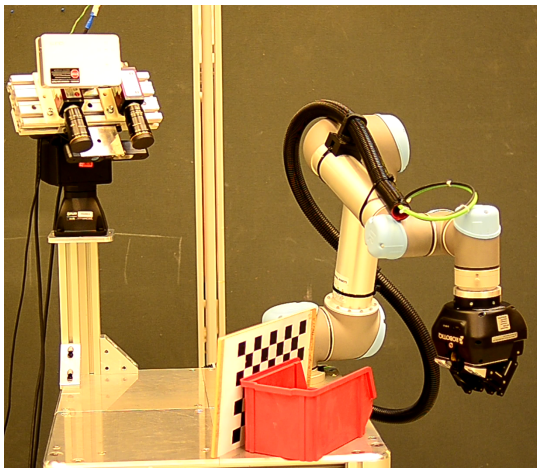
The need for placing the RGB-D sensor in the model-based environment, such that it corresponds to the real set-up, calls for a calibration. The calibration is done by localising the RGB-D sensor, in relation to the base frame of the UR5 manipulator. It is done by having the real manipulator holding a QR code in front of the sensor. By use of a calibration program, it is possible to get the position and orientation of the real RGB-D sensor, and place the virtual RGB-D sensor within an uncertainty of ±4mm [Andersen et al., 2013]. The accuracy is assessed applicable for the proof of concept set-up. The calibration is further elaborated in appendix H.3.

10.2.2 Test Set-Up

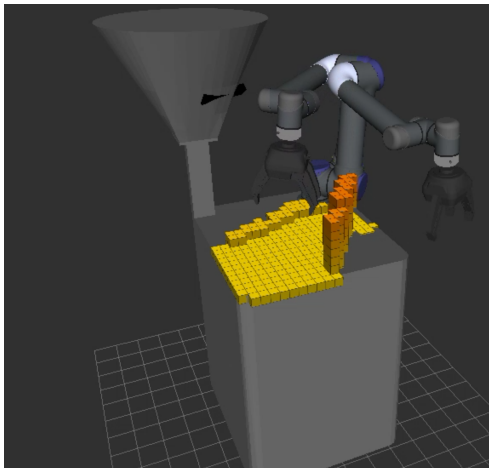
It is chosen to test the motion planning in a sensor-based environment, with an obstacle placed in the field of view of the RGB-D sensor, as shown in figure 10.1a and visualised through Rviz in figure 10.1b. The motion planning algorithm then has to create a plan, such that the manipulator does not collide with the obstacle or the model of the Little Helper 4 manipulator and platform. At present the implemented proxy and driver for the UR5 manipulator are not capable of cancelling a trajectory, thus the obstacle has to be stationary during the motion planning. Furthermore, it has not been confirmed nor denied, if this is possible, through the current configuration of MoveIt.

10.2.3 Self-filtering of the Equipment

MoveIt utilises OctoMap library to filter the 3D data and to do self-filtering. The OctoMap library is described in appendix H. Self-filtering is done for the model-based representation of the environment,



(a) The real set-up. A red box supporting the plate, which is shadowed out on in the simulation, can be seen.



(b) The two manipulators symbolises the initial state and the goal state, whereas the coloured boxes are voxels (volumetric pixels) created by OctoMap. The boxes represents obstacles.

Figure 10.1: A comparison between the real set-up and the simulated model, with a RGB-D sensor input.

such that MoveIt does not interprets for example the manipulator as an obstacle.

As mentioned in chapter 8, all model-based objects have a visualisation part and a collision part. Besides utilising the collision part for collision checking, then it is likewise utilised for the self-filtering. Thus, OctoMap ignores areas in the model-based environment, where the collision part of the respective objectives are located.

The used collision meshes for the UR5 manipulator is relatively large, compared to the actual size of the manipulator. This means the manipulator is self-filtered even though a cable-pipe is attached to the manipulator, as long as the cable-pipe is kept along the manipulator. The cable-pipe is shown in figure 10.1a and the self-filtering is shown in figure 10.1b.

Challenges have emerged when self-filtering is done for the RQ3 gripper. An assessed combination of relatively smaller collision meshes, and a relatively complex geometry, compared to the UR5 manipulator, means that it is challenging to do self-filtering. The failing, regarding complete self-filtering of the RQ3 gripper may also be due to a too inaccurate sensor calibration. Additionally the issue can be that the fingers are not positioned alike in the model and on the real gripper, the gripper has in the environment been treated as a static model. Due to the assessment regarding the calibration, a temporary solution is chosen: To test the sensed environment with a closed gripper, where the collision mesh is changed to a solid box, shielding the outer geometry of the gripper. The result is that the whole set-up becomes self-filtered, but the gripper will not be able to grasp during testing in a sensor-based environment.

Both the RGB-D sensor and the platform are calibrated relative to the manipulator. This entails that the calibration error between the RGB-D sensor and the platform is the difference between both errors. This error could be the reason for the poor self-filtering of the platform, as shown in figure 10.1b. The calibration error between the manipulator and the platform has not been determined.

10.2.4 Shadowing

Since the 3D data solely relies on one RGB-D sensor at present, then shadowing will occur. This is seen by comparing figure 10.1a and figure 10.1b, where it can be seen that the red box does not appear in the sensor-based representation of the environment. The result can be that the UR5 manipulator collides with the box. The shadowing occurs for example behind the UR5 manipulator and RQ3 gripper, since the RGB-D sensor obviously cannot see thorough the equipment. To remove the chance for shadowing, more sensors can added to the sensor-based representation of the environment.

10.3 Test and Verification

The test for verification is composed of two manipulator configurations. The initial and goal configuration are placed on opposite sides of the obstacle. The initial configuration is placed furthest away from the sensor, to avoid shadowing from manipulator and gripper, in the area of the obstacle. This necessity underline the advantage of multiple sensors. The process of calculating a motion plan between two sides of the obstacle was conducted multiple times, without collision.

Issues with self-filtering the RQ3 gripper disappeared, when making a larger collision mesh. In the long term this is obviously not applicable, thus more care has to be taken, when calibrating the set-up.

Issues with shadowing occurred occasionally, where Voxels could not be cleared. The solution was to remove the obstacle and manipulator away from the *field of view* of the sensor, such that the whole view could be updated.

The functionality of motion planning in a sensor-based environment was tested. An obstacle was detected by use of a RGB-D sensor and a collision free trajectory avoiding the obstacle was executed. A sequence of the test is shown in figure 10.2, whereas the video of the test is appended on the enclosed CD. [1]

[1] See <Video/MoveIt with sensor input.mp4>

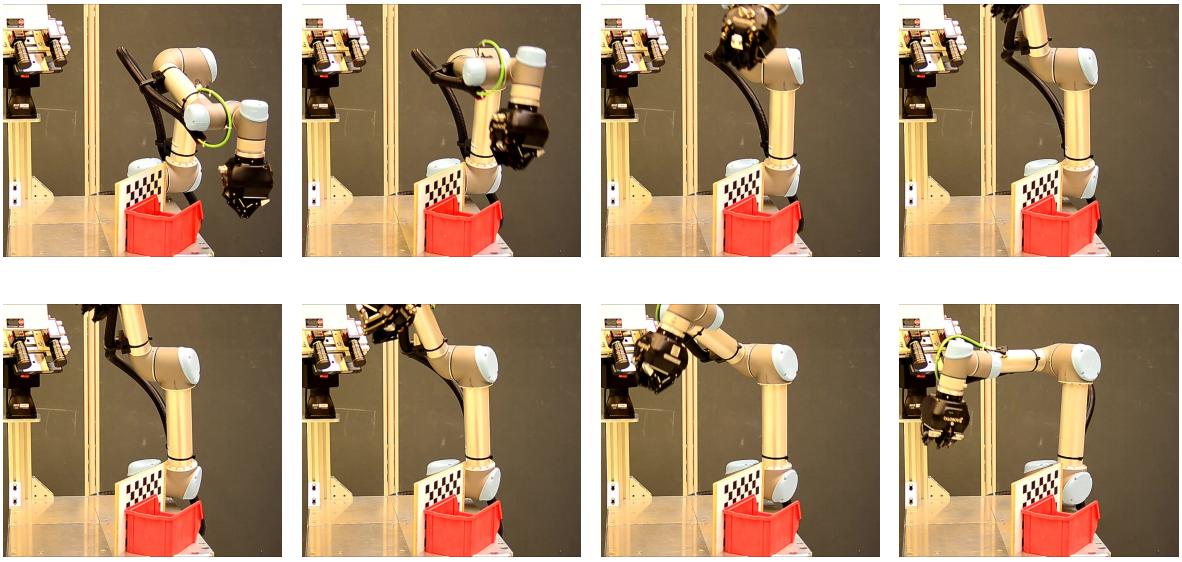


Figure 10.2: A sequence showing execution of a trajectory, based on motion planning based on sensor input.

10.4 Sub-conclusion

Motion planning through MoveIt, with sensor input to create an environment representation has been conducted, to answer **Hypothesise 5** in chapter 3. The basis for the environment was the model-based representation of the Little Helper 4 environment, conceived in chapter 8. The RGB-D sensor was calibrated and the input filtered, to obtain a representative representation, of the sensor-based environment. The difficulties have been exposed, and solutions to these, have been proposed during the design of this experiment. An obvious improvement to motion planning, based on sensor input, is the addition of multiple sensors, to reduce the effect of shadowing. The motion planning should be able to respond to changes in the environment, which is still to be implemented. Motion planning was successfully conducted based on a sensor-based representation of the environment.

Part IV

Conclusion

Chapter 11

Discussion

This chapter concerns a discussion aimed at providing a critical evaluation on the choices made throughout the project.

The Concept Proposals

Two concept proposals were chosen to be conceived, prior to the integration of motion planning in the skill-based system (SBS). The process of conceiving the proposals was based on brainstorming sessions and exemplification by written literature, if this was possible to obtain. The concept proposals were discussed and verified in overall terms, with the main developers of the Little Helper SBS, the *Robotics and Automation Group* at *Aalborg University*, which are assessed to have extensive experiences within the SBS. It can however be argued that the developers needed a more comprehensive understanding of each specific functionality within the concept proposals, in order to make sufficient judgement.

Parts of the long-term concept proposals have not been supported. This mainly concerns the proposed handlers, because the applicability of the handlers have not been verified by a proof of concept implementation, or by examples within written literature. The functionality of a handler can be compared to the functionality of MoveIt; creating an architecture, which combines different software tools and enables for plug-ins. Based on this assumption is it assessed that handlers with the desired functionalities can be developed.

Selection of Software

During the selection of a software solution for motion planning, a screening between three solutions were conducted. It can be discussed, whether the chosen software solution is in fact the most applicable solution for the Little Helper SBS, since the selection was based on a screening of only three candidates. It can be argued that an in-depth analysis of which software solutions that are most applicable, based on a demand analysis for Little Helpers, would have had been preferable. The result of the selection is mainly relying on a hunch about which solution that would be most beneficial. The choice of conducting a screening was mainly due to that the specific motion planning tool should not be the focus in this project, but instead the remedy towards an implementation of motion planning in a SBS. Thus, the actual software solution was immaterial to the project.

Selection of Motion Planning Algorithm

The selection of a motion planning algorithm to be utilised, in the Little Helper 4 software framework, is based on the benchmark and literature study. The validity of the benchmarking results can be discussed, which will be elaborated in the following. Based on the literature study, it is assessed that for example the results of the benchmarking are depended on the set-up of the system and the given motion planning problem. Thus the results does not encompass every motion planning problem, every Little Helper will encounter. This means that it can be argued that the chosen algorithm might not be advantageous for motion planning problems, which significantly deviates from the two tested. Thus to retrieve indicative results from the benchmarking, it can be argued that more effort had to be taken into account, when designing the motion planning problems.

Benchmarking was not conducted for the KUKA LWR (LWR) manipulator. It can be argued that this had to be done, since the mechanical set-up is different from that of the Universal Robots UR5 (UR5) manipulator. Motion planning with both manipulators at the same time will, in addition, provide a high degrees of freedom motion planning problem, which may also provide different benchmarking results.

The specific motion planning algorithms of concern was limited to algorithms implemented in Open Motion Planning Library (OMPL). Since the literature study and benchmarking was delimited to the motion planning algorithms in OMPL, then it can be argued that the literature study had to be extended with other algorithms, to determine if the algorithms of OMPL was sufficient. The amount of algorithms was however assessed to be comprehensive enough to solely rely on OMPL.

During the benchmarking, it was chosen to use the default parameter values in OMPL for the motion planning algorithms. It can be argued that the motion planning algorithms does not show their true

performance, with the default parameter values. Some algorithms may perform better than the selected motion planning algorithm, if correct parameter values were selected in respect to a given motion planning problem. It is however assessed that this would demand an extensive factorial experiment, which was out of the scope at these initial development stages, within implementing motion planning in the Little Helper SBS. The default parameter values were used based on the assumption that the developers of OMPL have given a qualified estimate.

Manipulator Kinematics

It was chosen to implement both the UR5 and the LWR manipulator into MoveIt. The Kinematics and Dynamics Library (KDL) handled kinematics calculations, which is the standard numerical kinematics solver in MoveIt. Kinematics can also be solved with IKfast, which is an analytical solver for kinematics. It is supposed to be faster and more reliable than its numerical counterpart is. Attempts for utilising IKfast were conducted, but not succeed during this project. It can be argued that more effort should have been offered in making IKfast working with the UR5 manipulator. It was however not a main concern of this project, thus the numerical solver was used.

It has been experienced that MoveIt had difficulties in solving a motion planning problem with the true joint limits of the UR5 manipulator, when using a Cartesian pose goal. This has been experienced neither with reduced joint limits nor with the goal given, as a corresponding joint configuration. It is assessed that the problem could be related to the kinematics solver, either because it cannot solve the kinematics request or provides a poor solution for the goal state, with respect to the initial state. A poor solution for the goal state can result in motion planning algorithms having difficulties in covering the size of the configuration space with the true joint limits. It can be discussed whether reducing the joint-limits is desirable, but it is assessed admissible for the proof of concept implementation.

Practical Verifications

During the practical verification, work pieces were not included in the environment. The reason for not handling work pieces was that these are mainly aimed for grasping techniques, where the focus in this project was limited to motion planning with articulated manipulators. This entailed problems, when handling work pieces, since the motion planning algorithm was not able to account for work pieces and thereby resulting in collision with the environment. Thus, it can be argued that the work pieces had to be included in the simulations, even if grasping techniques was not the scope of the project.

The practical verification of two manipulators was conducted by letting the UR5 and LWR manipulator motion plan within each other's work envelopes. It can discussed whether this verification is comprehen-

sive enough, to show the true potential within motion planning for dual manipulators. However, since it was not possible to execute trajectories by use of the LWR, then it was assessed redundant to attempt to utilise the two manipulators for for example a bimanual assembly task.

During the execution of motion planning, within a sensor-based environment, issues emerged because of coarse calibrations and shadowing. It was chosen to utilise a quick and relatively coarse method for calibrating the utilised sensor, since it at first was assessed to be sufficient to prove the concept. The shadowing occurred because only one RGB-D sensor was utilised. It can be discussed if it had been advantageous to introduce multiple sensors to avoid shadowing, but implementation of one sensor proved the basic concept of motion planning in a sensor-based environment. In addition, more effort could have been invested in enabling the possibility to alter a trajectory "on-the-fly", such that the system can react based on a dynamic environment and thereby increase the capability to collaborate with humans. This would put requirements to different parts of the software framework, for example the timing of the proxy/driver, filtering of the sensor input and the calculation time for the motion planning algorithm.

Error handling was chosen to be delimited during the project. This was based on the belief that error handling was first necessary, when the system was developed and functioning. During the practical implementation, the assumption regarding error handling was proven wrong, because the execution of planning request from a task continued even though a former request had failed. Thereby the need for error handling occurred during the development phase.

Chapter 12

Conclusion

The conclusion seeks to verify that the hypothesis of chapter 3 are fulfilled, based on the work described in this report.

This project concerned the conceptual implementation of motion planning in the skill-based system (SBS) of Little Helper 4. The project had its origin in a long-term concept proposal, concerning how to handle motion planning in a SBS of the future. A short-term concept proposal was conceived, based on the long-term concept proposal, to propose how motion planning could be implemented in the current SBS. An implementation to verify the short-term concept proposal was to be carried out. This gave rise to conduct a selection of a motion planning software tools that were assessed to be applicable for the SBS of Little Helper 4. Based on the selection, MoveIt was the motion planning software tool of choice. It was chosen to work with the Open Motion Planning Library (OMPL), which is the default motion planning library in MoveIt. OMPL provided a variety of motion planning algorithms, which a literature study was based upon. A benchmark was conducted, where the results was compared to the literature study. This provided a motion planning algorithm to be used, for the implementation of motion planning in the SBS of Little Helper 4. The implementation of motion planning in the SBS of Little Helper 4 was conducted, to realise some of the concepts from the short-term proposal.

The work was further extended by introducing motion planning for multiple manipulators in a set-up consisting of both Little Helper 3 and Little Helper 4, and by introducing sensor-based environment representation into MoveIt.

The Concept Proposals

Priori to the implementation of motion planning, in the software framework of Little Helper 4, brainstorming sessions, about how to incorporate motion planning, were conducted. As a result the long-term concept proposal was proposed, where limitations regarding development and incorporation were not taken into account.

The long-term concept proposal paved the way for how to incorporate motion planning in a SBS of the future. This was done by introducing motion planning on multiple equipment, and to cope with these via a motion planning handler. This entailed the need for an environment handler, to cope with the environment representation, for multiple motion planning software tools. It was further proposed how to encompass motion planning enhancement, by proposing learning and a common database for the Little Helpers.

The aim of the short-term concept proposal was focused on a feasible incorporation of motion planning in a present SBS, and thereby maintaining the overall structure. It was deduced to maintain the motion planning handler in the short-term concept proposal, to encompass multiple types of motion planning software tools. This entailed an incorporation of an environment representation in the SBS. Proposals for how to cope with user interaction, error handling, and calibration was likewise given.

The concept proposals were, where it was possible, supported by exemplification from written literature, to support the applicability. Both concept proposals were presented and discussed with parts of the *Robotics and Automation Group* at *Aalborg University*, to overall verify the feasibility. Hereby it was concluded that the concept proposals in general were feasible, within their respective time frames, and that **Hypothesise 1** from chapter 3 was answered.

Choosing of Motion Planning Algorithms

Selection of a motion planning algorithm was done by conducting a literature study of motion planning algorithms, within the OMPL. This was done with focus on their applicability for the Little Helper 4 software framework. The motion planning algorithms were further benchmarked, to test their applicability in practise. The benchmarking was done for Little Helper 4, in two different scenarios. The data from the benchmarking was used to select algorithms for further investigation. The selected algorithms were analysed, by comparing the results and the literature study. RRTConnect was the motion planning algorithm of choice, still having in mind that the results were based on two dedicated motion planning problems. It was hereby assessed that **Hypothesise 2** was answered.

Practical Implementation

The practical implementation was conducted in three phases; implementation of motion planning in the Little Helper 4 software framework, motion planning for multiple manipulators, and motion planning in a sensor-based environment.

Motion planning was implemented in the Little Helper 4 software framework, based on the short-term concept proposal. The implementation included creation of a proxy and driver for the Universal Robots UR5 (UR5) manipulator and a MoveIt proxy. The implementation was tested, based on execution of a task, taught through the user interface of the Little Helper 4 software framework. The implementation proved the main concepts of the short-term concept proposal, by a successfully execution of the task, and thereby answering **Hypothesise 3**. Error handling, and the capability to obey process constraints during motion planning, was not implemented, which still leaves some parts of the short-term concept proposal unanswered.

The utilisation of motion planning in the Little Helper 4 software framework additional paved the way for motion planning with multiple manipulators. An environment representation, containing both the Little Helper 3 and Little Helper 4, was created, including corresponding planning groups for the two manipulators. A driver for the KUKA LWR (LWR) manipulator, the UR5 proxy/driver, and a dual proxy were used to interface MoveIt with the hardware. The provided driver for the LWR was not able to execute a trajectory, but instead executed the corresponding path, with a predefined velocity. Tests were successfully conducted, by means of motion planning with two manipulators, with overlapping work envelopes, despite the lacking ability to execute trajectories for the LWR manipulator. This was the basis to answer **Hypothesise 4**. In addition to the hypothesis, a proposal for implementation of the dual manipulator set-up in the SBS of Little Helper 4 was outlined.

Motion planning, based on an environment representation with both model and sensor input, has been conducted, by use of MoveIt. This was done by introducing RGB-D sensor data into MoveIt. Challenges with self-filtering and shadowing was clarified, and temporary solutions was created, to obtain the basis for sensor-based motion planning to be implemented in the future. Tests were conducted, where an obstacle was placed in the field of view of the RGB-D sensor, and the manipulator was able to avoid the obstacle, by use of motion planning. Hereby it can be concluded that **Hypothesise 5** was answered, as the final hypothesis of this report.

Based on the work conducted during this project, a conceptual implementation and verification of motion planning in a skill-based system has been presented.

Bibliography

Andersen, R., Damgaard, J., Madsen, O., and Moeslund, T. (2013). Fast calibration of industrial mobile robots to workstations using QR codes. In *Robotics (ISR), 2013 44th International Symposium on*, pages 1–6.

Angerer, S., Strassmair, C., Staehr, M., Roettenbacher, M., and Robertson, N. (2012). Give me a hand - The potential of mobile assistive robots in automotive logistics and assembly applications. In *Technologies for Practical Robot Applications (TePRA), 2012 IEEE International Conference on*, pages 111–116.

Bøgh, S., Nielsen, O. S., Pedersen, M. R., and Krüger, V. (2012). Does your robot have skills? In *Proceedings of the 43rd International Symposium on Robotics*.

Bohlin, R. and Kavraki, E. (2000). Path planning using lazy PRM. In *Robotics and Automation, 2000. ICRA '00. IEEE International Conference on*, pages 521–528 vol.1.

Burns, B. and Brock, O. (2006). Sampling-based motion planning using uncertain knowledge. Technical report, University of Massachusetts Amherst.

Carøe, C., Hvilshøj, M., and Schou, C. (2012). Intuitive Programming of AIMM Robot. Master's thesis, Aalborg University Department of Mechanical and Manufacturing Engineering.

Chitta, S. (2013). MoveIt! `http://www.willowgarage.com/blog/2013/05/06/moveit` [Accessed 26 May 2014].

Cohen, B., Sucan, I., and Chitta, S. (2012). A generic infrastructure for benchmarking motion planners. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 589–595.

Coleman, D. (2013). Benchmarking. `http://moveit.ros.org/wiki/Benchmarking` [Accessed 21 April 2014].

Coppelia Robotics (2014). V-REP. http://www.coppeliarobotics.com [Accessed 21 May 2014].

Dobson, A. and Bekris, K. (2013). Improving sparse roadmap spanners. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4106–4111.

Dobson, A., Krontiris, A., and Bekris, K. (2013). Sparse Roadmap Spanners. In *Algorithmic Foundations of Robotics X*, volume 86 of *Springer Tracts in Advanced Robotics*, pages 279–296.

Edwards, S. (2014). industrial_robot_simulator. http://wiki.ros.org/industrial_robot_simulator [Accessed 10 May 2014].

Edwards, S., Glaser, S., and Hawkins, K. (2013). Universal Robots - ROS Wiki. http://wiki.ros.org/universal_robot [Accessed 24 May 2014].

Ellekilde, L.-P. and Jorgensen, J. A. (2010). RobWork: A Flexible Toolbox for Robotics Research and Education. *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1 –7.

Ellekilde, L.-P. and Jorgensen, J. A. (2014). RobWork. http://www.robwork.dk/ [Accessed 16 May 2014].

Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W. (2012). An evaluation of the RGB-D SLAM system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696.

Gipson, B., Moll, M., and Kavraki, L. (2013). Resolution Independent Density Estimation for motion planning in high-dimensional spaces. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2437–2443.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2014). RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Experimental Robotics*, volume 79 of *Springer Tracts in Advanced Robotics*, pages 477–491.

Hermann, A., Xue, Z., Ruhl, S., and Dillmann, R. (2011). Hardware and software architecture of a bimanual mobile manipulator for industrial application. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2282–2288.

Hershberger, D., Gossow, D., and Faust, J. (2014). Rviz. http://wiki.ros.org/rviz [Accessed 9 May 2014].

Hsu, D., Latombe, J.-C., and Motwani, R. (1997). Path planning in expansive configuration spaces. In *Robotics and Automation*, volume 3 of *1997 IEEE International Conference on*, pages 2719–2726.

Hvilshøj, M., Bøgh, S., Skov, O., and Madsen, O. (2012). Autonomous industrial mobile manipulator (AIMM): past, present and future. *Industrial Robot: An International Journal*, 39:120–135.

Hvilshoej, M., Boegh, S., Madsen, O., and Kristiansen, M. (2010). Calibration Techniques for Industrial Mobile Manipulators: Theoretical configurations and Best practices. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–7.

Jaillet, L., Cortés, J., and Siméon, T. (2010). Sampling-Based Path Planning on Configuration-Space Costmaps. *IEEE Transactions on Robotics*, 26:635–646.

Kallmann, M. and Jiang, X. (2010). A Motion Planning Framework for Skill Coordination and Learning. In *Motion Planning for Humanoid Robots*, pages 277–306.

Karaman, S. and Frazzoli, E. (2011). Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*, 30:846–894.

Kavraki, L. E., Švestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580.

Kuffner, J. and LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings.*, volume 2 of *ICRA '00. IEEE International Conference on*, pages 995–1001.

KUKA (2012). KUKA LWR. `http://www.kukaconnect.com/wp-content/uploads/2012/07/KUKA_LBR4plus_ENLISCH.pdf` [Accessed 10 May 2014].

Kunze, L., Roehm, T., and Beetz, M. (2011). Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5589–5595.

Ladd, A. M. and Kavraki, L. E. (2005). Motion planning in the presence of drift, underactuation and discrete system changes. In *Proceedings of Robotics: Science and Systems*, pages 233–241.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Li, T.-Y. and Shie, Y.-C. (2002). An incremental learning approach to motion planning with roadmap management. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 4, pages 3411–3416.

Luna, R., Sucan, I., Moll, M., and Kavraki, L. (2013). Anytime solution optimization for sampling-based motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5068–5074.

Machine Vision (2013). Little Helper. http://www.machinevision.dk/wordpress/little-helper/ [Accessed May 2014].

May, S., Werner, B., Surmann, H., and Pervolz, K. (2006). 3D time-of-flight cameras for mobile robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 790–795.

METAL SUPPLY (2010). KUKA Roboter is presenting a new robot generation at Automatica 2010. http://www.metal-supply.com/announcement/view/7193/kuka_roboter_is_presenting_a_new_robot_generation_at_automatica_2010#.U3djnvl_t8E [Accessed 17 May 2014].

Miller, A. and Allen, P. (2004). Graspit! A versatile simulator for robotic grasping. *Robotics Automation Magazine, IEEE*, 11(4):110–122.

MoveIt (2013). Kinematics/IKFast - MoveIt. http://moveit.ros.org/wiki/Kinematics/IKFast [Accessed 12 May 2014].

MoveIt (2014a). 3D Sensors. http://moveit.ros.org/wiki/3D_Sensors [Accessed 12 May 2014].

MoveIt (2014b). Concepts. http://moveit.ros.org/documentation/concepts/ [Accessed 13 May 2014].

Muijzer, F. (2014). Development of an automated exercise Detection and Evaluation system using the Kinect depth camera. Master's thesis, Electrical Engineering, Mathematics and Computer Science, University of Twente.

Nicolas Lauzier (2013). Stack for robotiq grippers users. https://github.com/ros-industrial/robotiq [Accessed 26 November 2013].

Niemueller, T. D., Lakemeyer, G., and Srinivasa, S. (2012). A Generic Robot Database and its Application in Fault Analysis and Performance Evaluation. In *IEEE International Conference on Intelligent Robots and Systems*, pages 364–369.

OMPL Rice University (2014a). Available Planners. http://ompl.kavrakilab.org/planners.html [Accessed 19 May 2014].

OMPL Rice University (2014b). ompl::geometric::PathSimplifier. http://ompl.kavrakilab.org/classompl_1_1geometric_1_1PathSimplifier.html [Accessed 03 April 2014].

Ott, C., Eiberger, O., Friedl, W., Bauml, B., Hillenbrand, U., Borst, C., Albu-Schaffer, A., Brunner, B., Hirschmuller, H., Kielhofer, S., Konietschke, R., Suppa, M., Wimbock, T., Zacharias, F., and Hirzinger, G. (2006). A Humanoid Two-Arm System for Dexterous Manipulation. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 276–283.

Oyama, A., Chitta, S., Conley, K., Bradski, G., Konolige, K., and Cousins, S. (2009). Come on in, our community is wide open for Robotics research! https://www.willowgarage.com/sites/default/files/RSJ2009_AkihisaOyama_ComeOnIn_En.pdf [Accessed 5 May 2014].

Pan, J., Chitta, S., and Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3859–3866.

Pedersen, M. R. (2011). Integration of the KUKA Light Weight Robot in a mobile manipulator. Master's thesis, Aalborg University Department of Mechanical and Manufacturing Engineering.

Pedersen, M. R., Nalpantidis, L., Bobick, A., and Krüger, V. (2013). On the Integration of Hardware-Abstracted Robot Skills for use in Industrial Scenarios. *Second international workshop on Cognitive Robotics Systems*.

Raveh, B., Enosh, A., and Halperin, D. (2011). A Little More, a Lot Better: Improving Path Quality by a Path-Merging Algorithm. *IEEE Transactions on Robotics*, 27(2):365–371.

Robotiq (2013). Robotiq 3-Finger Adaptive Robot Gripper Instruction Manual. http://support.robotiq.com [Accessed 04 May 2014].

ROS (2013). Documentation - ROS Wiki. http://wiki.ros.org/ [Accessed 8 May 2014].

Salzman, O. and Halperin, D. (2013). Asymptotically near-optimal RRT for fast, high-quality, motion planning. *CoRR*.

Saxena, A., Wong, L., Quigley, M., and Ng, A. (2011). A Vision-Based System for Grasping Novel Objects in Cluttered Environments. In *Robotics Research*, volume 66 of *Springer Tracts in Advanced Robotics*, pages 337–348.

Schou, C., Damgaard, J., Bogh, S., and Madsen, O. (2013). Human-robot interface for instructing industrial tasks using kinesthetic teaching. *Robotics (ISR), 2013 44th International Symposium on*, pages 1–6.

Sánchez, G. and Latombe, J.-C. (2003). A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. In *Robotics Research*, volume 6 of *Springer Tracts in Advanced Robotics*, pages 403–417.

Srinivasa, S., Berenson, D., Cakmak, M., Collet Romea, A., Dogar, M., Dragan, A., Knepper, R. A., Niemueller, T. D., Strabala, K., Vandeweghe, J. M., and Ziegler, J. (2012). HERB 2.0: Lessons Learned from Developing a Mobile Manipulator for the Home. In *Proceedings of the IEEE*, volume 100, pages 2410–2428.

Şucan, I. and Kavraki, L. (2010). Kinodynamic Motion Planning by Interior-Exterior Cell Exploration. In *Algorithmic Foundation of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 449–464.

Universal Robots (2013a). Force control introduced by Universal Robots. http://www.universal-robots.com/GB/Universal_Robots/News.aspx?Action=1&NewsId=382&PID=14345 [Accessed 25 Nov 2013].

Universal Robots (2013b). Technical Specifications. http://media1.limitless.dk/UR_Tech_Spec/UR5_GB.pdf [Accessed 14 February 2014].

Universal Robots (2013c). The URScript Programming Language. http://ur-update.dk/URsupport/Manuals/ScriptManual/Release1.8Beta/scriptmanual_en.pdf [Accessed 23 February 2014].

Universal Robots (2013d). Universal Robots - Products. http://www.universal-robots.dk/DK/Presse/Multimedia/Products.aspx [Accessed 01 April 2014].

Universal Robots (2013e). Universal Robots Support. http://support.universal-robots.com/ [Accessed 09 April 2014 - Confidential.].

Venator, E., Lee, G., and Newman, W. (2013). Hardware and software architecture of ABBY: An industrial mobile manipulator. In *Automation Science and Engineering (CASE), 2013 IEEE International Conference on*, pages 324–329.

VT3-2013 (2013). Creation of an Autonomous Industrial Mobile Manipulator. Technical report, Aalborg Universitet.

Walpole, R. E., Myers, R. H., Myers, S. L., and Ye, K. (2007). *Probability and Statistics for Engineers and Scientists*. Pearson Prentice Hall, 8 ed. edition.

Willow Garage (2012). PR2 User Manual. http://pr2support.willowgarage.com/wiki/PR2%20Manual?action=AttachFile&do=get&target=pr2_manual_r321.pdf [Accessed 5 May 2014].

Zacharias, F., Leidner, D., Schmidt, F., Borst, C., and Hirzinger, G. (2010). Exploiting structure in two-armed manipulation tasks for humanoid robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5446–5452.

Part V

Appendix

Appendix A

Motion Planning in the Little Helper 4 Software Framework

This appendix is a supplement to some of the statements from chapter 8. The appendix elaborates parameters, which is a part of the implemented motion planning.

A.1 MoveIt Proxy

The MoveIt proxy translates the output from a skill, to input for MoveIt. The output from a skill, regarding a query, is defined as a Cartesian pose. The rotation of the TCP specified in the Cartesian pose is transformed from fixed XYZ angles[1] to Quaternions. The motion planning goal, specified in XYZ translation and Quaternions, are passed to MoveIt along with the parameters described in the following.

Parameters editable through the task file are specified below:

- **Planning Time:** The allowable planning time for the motion planning, without post-processing.
- **Planning Algorithm:** The used planning algorithm. The available planners are determined from the benchmarking results from chapter 7.
- **Planning Reference Frame:** Planning based on the base coordinate frame ("absolute" movement) or TCP coordinate frame ("relative" movement).

Parameters editable through the task file are parameters, which should be easy to change, either as a part of the teaching phase of a task or as part of the system development. Making parameters editable through

[1]The UR5 manipulator specifies TCP rotation by use of fixed XYZ angles and are therefore stored in the task file during the teaching phase. The teaching phase are shortly described in section 2.2 on page 15.

a task file, is faster to change than changing the parameter in a program, which needs to be compiled before execution.

Parameters, which are not editable through the task file, are specified directly in the MoveIt proxy and listed below:

- **Goal Orientation Tolerance:** Orientation tolerance of the TCP when planning to a pose: 0.01rad
- **Goal Position Tolerance:** Position tolerance of the TCP when planning to a pose, radius of sphere the TCP have to be within: 10mm
- **Number of Planning Attempts:** The number of times MoveIt are conduction motion planning attempts, to aid a successful planned trajectory: 3

The values of the tolerances are discussed and justified in chapter 8.

A.2 MoveIt

The structure of MoveIt is shown in figure 8.2. A kinematics solver is used to translate the input query to a joint configuration. The collision checking is assessed to be conducted in Cartesian space, because the environment is specified in Cartesian space. Therefore are the potential joint configurations of the manipulator validated in Cartesian space.

All information regarding a model of the manipulator, calibration, and kinematics are described in a unified robot description format (URDF) file and a semantic robot description format (SRDF) file. The URDF and SRDF files for the Little Helper 4 software framework are described in appendix G.

A.2.1 Kinematics solver

The specification of the kinematics solvers used for the individual planning groups are specified in a *kinematics file*[2]. The standard solver in MoveIt is the numerical kinematics and dynamics library (KDL)[3]. The KDL solver utilises kinematics information from the URDF file to determine a joint configuration based on the received Cartesian pose. Attempts to implement an analytical solver have been conducted. IKFast is an analytical solver, where kinematics expressions should be generated [MoveIt, 2013]. The analytical solver reduces the calculation time, allowing for a faster motion planning.

The IKFast expressions have been generated for the original URDF containing the kinematics for the UR5 manipulator. The URDF file has been change to make the model, based on the URDF file, corre-

[2]The file is appended on the enclosed CD, see <Source code/lh4_moveit_config/config/kinematics.yaml>

[3]More information about the used KDL package at: `http://www.orocos.org/kdl`

spond to the real manipulator. The changes, regarding the manipulator, are limited to rotating joint 2, 180° and addition of a TCP link. The TCP link offset the tool centre point from the joint between the two outermost links to the flange of the manipulator. The TCP link and additional changes to the URDF file are described in appendix G. The IKFast expressions have not been possible to generate subsequent to the changes.

The kinematics file also contains specific parameters, related to the kinematics solver, as for example the number of attempts to solve a given configuration and the allowable time for each attempt.

A.2.2 Motion Planning

The motion planning is started with the specified parameters and the chosen motion planning algorithm. Collision checking is conducted either during the motion planning or afterwards, depending on the chosen algorithm, corresponding to the description in chapter 6. The collision checking is done, based on the environment and the disabling of collision between some links, as specified in the SRDF file, described in appendix G.

The trajectory post-processing operations, conducted by the motion planning algorithms, are described in appendix D. The post-processing consist of a path simplification and a path merging.

Longest Valid Segment

During conduction of motion planning queries, it has been experienced that the parameter *longest valid segment* has influence on whether a query succeeds or fails. The parameter is utilised by OMPL for discrete validation of a path, and describes the distance between the states, which are to be validated. An example of how a solution for a query can be found, without noticing a collision, is shown in figure A.1. As default the parameter is set to 0.05%, and after it has been changed to 0.001%, more queries have succeeded. The parameter is specified in the "ompl_planning.yaml" file[4], which also contains a specification of the planning algorithms.

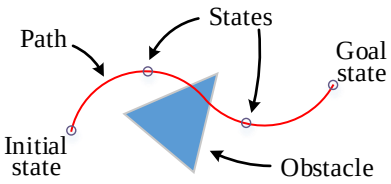


Figure A.1: Illustrates how a large longest valid segment parameter makes a path in collision succeed. With a smaller value, the collision would have been found, and another path would have been taken.

[4]The file is appended on the enclosed CD, see <Source code/lh4_moveit_config/config/ompl_planning.yaml>

A.2.3 Processing from a Path to a Trajectory

The output from the used OMPL motion planning algorithm can be either a path or a trajectory. If the output is a path, then MoveIt uses trajectory processing to generate appropriate trajectories, based on the path and the limits for each joint. The limits includes maximum velocity and acceleration, which are read from a "joint_limits.yaml" file[5]. [MoveIt, 2014b]

A.3 Execution of Trajectories

Execution of trajectories from MoveIt is visualised by use of the GUI Rviz. The execution can conducted either by a simulated robot controller or on the real hardware. Both executions make use of the same interface to and from MoveIt. The interface from MoveIt makes use of the ROS FollowJointTrajectory message type. An extract of the message type is shown in table B.3 and described in appendix B.2.1. The feedback to MoveIt concerns the joint configuration of the manipulator. The feedback follows the syntax of Joint State message[6]. A joint state message contains information of the name of the joints for identification, the position, velocity and effort (torque for rotational joints) for each joint.

[5]The file is appended on the enclosed CD, see <Source code/lh4_moveit_config/config/joint_limits.yaml>
[6]Joint states are a submessage group to "sensor_msgs"

Appendix B

Creation of Proxy and Driver for Universal Robots UR5 Manipulator

This appendix is used to document the work conducted, to make the Little Helper 4 software framework, created by [VT3-2013, 2013], compatible with motion planning. The motion planning is conducted by use of MoveIt and the communication is handled by the middleware ROS.

Making the Little Helper 4 software framework compatible with MoveIt, consist of two primary tasks: Handling trajectory commands from MoveIt and supporting the existing functionalities of the original Little Helper 4 software framework. To satisfy these demands, three different approaches to create a proxy/driver set-up are proposed. The approaches primarily concerns how to handle the connection between MoveIt and the UR5 manipulator. The other functionalities already have been developed in the proxy/driver from the original Little Helper 4 software framework [VT3-2013, 2013]. The structure of the software framework is shown in figure 8.1. This structure entails that the approaches concerns a proxy, a driver and perhaps a UR script:

1. Extension of the proxy/driver from the original Little Helper 4 software framework, to support the MoveIt trajectory movement commands.
2. Creation of a C-API proxy, which is equivalent to the development of the proxy/driver, from the original Little Helper 4 software framework, including support of the MoveIt trajectory movement commands. This requires the development of a controller for the manipulator.
3. Use of the ROS-Industrial Universal Robots (ROS-Industrial UR) package, and extension with the functionalities of the original Little Helper 4 software framework.

B.1 Selection of Proxy/Driver Approach

The selection of the proxy/driver approach is based on the availability and the functionalities of the approaches. Approach 1 contains the functionalities of the original Little Helper 4 software framework, but lacks the ability to handle communication to and from MoveIt (joint states and trajectories respectively), including the ability to execute a trajectory. Approach 2 concerns creating a driver/proxy from scratch, based on the UR C-API interface, which is a low level language, compared to the URScript used in approach 1 and 3 The use of the C-API interface requires development of a new controller for the UR5 manipulator [Universal Robots, 2013e]. Approach 3 is capable of handling the communication between MoveIt and the UR5 manipulator, allowing for execution of trajectories [Edwards et al., 2013]. Therefore the work related to this approach, concerns implementation of the functionalities of the Little Helper 4 framework.

Approach 3 is chosen due to the implemented functionalities in the ROS-Industrial UR package, and because the additionally needed functionalities have once been implemented in a proxy/driver by the project group. The experiences from creating the original Little Helper 4 software framework can therefore be reused. Approach 3 does also comply with the desires, stated in [VT3-2013, 2013], to be able to use existing ROS packages, with minor or none modifications in the software framework.

B.2 Additions to the Little Helper 4 Software Framework

The key components of the chosen ROS-Industrial UR package are an URScript program and a ROS node created in the programming language Python. URScript is a scripting language, which can be executed on Universal Robots manipulators, using build-in functionalities. The URScript of the package is executed on the manipulator, while being controlled by the ROS node. This structure corresponds to the structure of the proxy/driver of the original Little Helper 4 software framework [VT3-2013, 2013], described in section 2.2. To obey the demand of maintaining the functionalities of the original Little Helper 4 software framework, the functionalities of the package needs to be expanded, regarding both the URScript and the python program. In addition, a proxy node must be developed to handle the communication between the software framework and the driver. The proxy should be connected to both the master node and MoveIt, as shown in figure B.1.

The communication between the driver and the proxy is handled by two actionlibs, one standard ROS FollowJointTrajectory (FJT) actionlib, and one custom actionlib, created to suit the needs of the original Little Helper 4 software framework. The actionlibs are described in section B.2.1. The needs from the original framework consist of the following abilities:

- Linear movement of the TCP with joint and Cartesian input.
- Joint movement with joint and Cartesian input.
- Use of the force functionalities embedded in the UR5 manipulator.
- Be able to return the Cartesian position of the TCP to the proxy.

Additionally demands, as a result of contributions to the ACAT project, concerns the possibility to offset the tool centre point and the mass and centre of gravity for the tool, described in appendix I. The developed proxy is controlled either by the master node, handling commands directly, or controlled by an execution request from the MoveIt motion planner. A request from MoveIt is initiated by the master node.

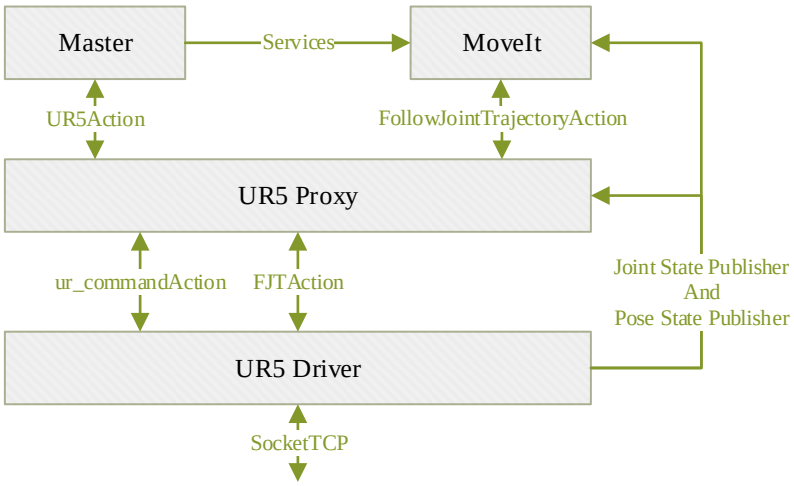


Figure B.1: The detailed framework of the UR5 manipulator branch.

B.2.1 UR5 Proxy

A proxy is used to handle the communication between the generic commands from the master to the hardware specific commands used by the driver [VT3-2013, 2013]. The functionality is expanded to include commands from MoveIt also. The original ROS-Industrial Universal Robots package is capable of handling the generic commands from MoveIt directly, but the added functionalities needs to be handled by the proxy.

All communication to and from the proxy is handled by ROS actionlibs. An overview of the actionlib connections is shown in figure B.1. The master is connected to the proxy through an *ur5* actionlib. MoveIt is connected to the proxy through a *FollowJointTrajectory* (*FJT*) actionlib. The actionlibs are

described in the following. The proxy translates the input to two different actionlibs; another FJT actionlib and a custom *urcommands* actionlib. All commands regarding movement of the manipulator, to the driver, is handled by the FJT actionlib; commands from the master is translated into a FJT action and send along with an urcommands action. Commands from MoveIt are send without translation (beside changed from one actionlib to another), along with an urcommands action, to verify the execution. If the master sends a goal regarding no-movement commands (force, set-tcp and get-data) then only the urcommands actionlib is used.

ur5 actionlib

The ur5 actionlib, handling the communication between the master and proxy, is described in table B.1. Because the driver handles all functionalities, regarding check of fulfilment of manipulator goals, there have not been a need for feedback.

Action	Name	Type	Description
Goal	movetype	string	The desired movetype: linear, joint or servo. Also used to specify *set TCP*, *force* and *get data*.
	inputtype	string	The inputtype, joint or Cartesian. Also used to specify *set TCP*, *force* and *get data*.
	coord	float64[6]	Move coordinate (Cartesian or joint) and TCP.
	speed	float64	Velocity of leading joint or TCP and mass of tool.
	acc	float64	Acceleration of leading joint or TCP.
	selection	int8[6]	Specifies if the manipulator is compliant in the corresponding degree of freedom.
	wrench	float64[6]	Obtain or account for the specified force, depending of the selection for the degree of freedom and TCP centre of gravity (X, Y, Z).
	limits	float64[6]	Allowed TCP velocity or TCP position deviation from the specified path, depending of the selection.
Result	resPos	float64[6]	Result Cartesian position of the manipulator.
	resJoint	float64[6]	Result joint configuration of the manipulator.

Table B.1: Overview of the ur5 actionlib, used to communicate between the master and the UR5 proxy.

urcommands actionlib

The urcommands actionlib, handles the communication between the UR5 proxy and driver, and is described in table B.2. Because the driver handles all functionalities, regarding check of fulfilment of manipulator goals, there have not been a need for feedback during execution.

Action	Name	Type	Description
Goal	tcppose	float64[6]	The Cartesian position of the TCP of the tool.
	mass	float64	Mass of the tool.
	tcpcog	float64[3]	Centre of gravity of the tool.
	selection	int8[6]	Specifies if the manipulator is compliant in the corresponding degree of freedom.
	wrench	float64[6]	Obtain or account for the specified force, depending of the selection for the degree of freedom and TCP centre of gravity.
	limits	float64[6]	Allowed TCP velocity or TCP position deviation from the specified path, depending of the selection.
Result	resPos	float64[6]	Result Cartesian position of the manipulator.
	resJoint	float64[6]	Result joint configuration of the manipulator.

Table B.2: Overview of the urcommands actionlib, used to communicate between the UR5 proxy and UR5 driver.

FollowJointTrajectory actionlib

The FollowJointTrajectory (FJT) is a standard message type, which is part of the ROS control_msgs package. It consist of goals, feedback, and results related to movement. The most used topics will be described in the following[1] and shown in table B.3. The FJT actionlib is primarily used to create a trajectory of a number of points, which consist of a set of manipulator configurations, velocities, and accelerations for each joint. Furthermore, is the total time from start specified, to ensure the manipulator is at a desired position at a desired time, which can be important, especially in a changing environment. A trajectory can consist of an arbitrary large number of configurations, to be executed by the manipulator (limited by the memory of the executing PC).

[1]The full list of topics from the actionlib can be found on: `http://docs.ros.org/api/control_msgs/html/action/FollowJointTrajectory.html`

Action	Name	Type	Description
Goal	trajectory	trajectory_msgs/ JointTrajectory	The trajectory to be executed by the manipulator. Part of FollowJointTrajectoryActionGoal action.
	points	trajectory_msgs/ JointTrajectoryPoint[]	The points to be executed of the manipulator. Goal as a part of "trajectory".
	positions	float64[]	The joint position of one trajectory point. Goal as a part of "points".
	velocities	float64[]	The joint velocity of one trajectory point. Goal as a part of "points".
	accelerations	float64[]	The joint acceleration of one trajectory point. Goal as a part of "points".
	time_from_start	duration	The allowable execution time for a trajectory point. Goal as a part of "points".
Result	status	actionlib_msgs/GoalStatus	The action status of the actionlib.

Table B.3: A extract of the FollowJointTrajectory actionlib. FollowJointTrajectory is part of the contol_msgs package [ROS, 2013].

B.2.2 ROS-Industrial Universal Robots driver

The driver is based on the ROS-Industrial Universal Robots package, where all core functionalities are maintained and used. The main changes to the package are addition of further functionalities and another actionlib server, as mentioned in section B.2.

The main structure of the driver is described in the following, to establish a baseline for the development of additional functionalities. An URScript script is send from the driver by use of at socket connection to the UR controller. The script is executed on the controller, listening for incoming commands from the driver. The driver listen for incoming trajectories via FJT actionlib.

Between each configuration of a received trajectory (including the start/current configuration of the manipulator), the driver conducts a cubic interpolation, to ensure a smooth movement between the configurations. Then each of the interpolated configurations from the trajectory are send to the manipulator, one by one. Each configuration of the trajectory is executed as individual joint movement commands.

Within the time frame of the trajectory, the interpolated configurations are sent successive. If the manipulator does not reach the final position of the trajectory, within the time frame, then the final configuration is send. This approach can cause the driver to skip some of the interpolated configurations. The ac-

tual configuration of the manipulator is compared to the desired, and the status of the trajectory is first changed to succeeded after the two configurations are within a specified tolerance. After successfully executing a command, then the driver returns to the listening state, ready for a new command.

Feedback from the driver, containing the current configuration of the manipulator, follows the syntax of Joint State message[2]. A joint state message contains information of the name of the joints for identification, the position, velocity, and effort (torque for rotational joints) for each joint.

The functionalities, described in section B.2, have been added to the original functionalities of the ROS-Industrial UR driver. The movement functionalities are primarily modifications of existing functions from the ROS-Industrial UR package. This has been done to maintain the ROS-Industrial approved structure, and to stay true to the developed package. No-movement commands have been created to adapt to the structure of the driver. An additional actionlib has been implemented, which handles specification of what command to be executed. The execution of no-movement commands are handled without the use of the FJT actionlib, using only the urcommands actionlib. When the no-movement commands have been executed, the status is also set to succeeded. The Cartesian position of the manipulator is published to a topic, enabling the framework to subscribe and continuous receive the position status.

Sum-up of the added functionalities, to the ROS-Industrial UR package:

- Send move, both linear and joint with joint and Cartesian input.
- Offset TCP, including centre of gravity and mass of the tool and placement of the tool centre point.
- Start and end force, including control of all related parameters.

The ROS packages described in this section is appended on the enclosed CD[3]. The packages have been created using ROS Hydro Medusa, compiled using catkin on Ubuntu.

[2]Joint states are a submessage group to "sensor_msgs", more info at `http://docs.ros.org/api/sensor_msgs/html/msg/JointState.html`

[3]See <Source code/ur5_rosi_proxy> and <Source code/ur_driver>

Appendix C

OMPL Sample-based Motion Planner Algorithms

This appendix introduces the motion planning algorithms in the Open Motion Planning Library (OMPL). It serves as a supplement to the descriptions in chapter 6, underlining the structure and functionalities of the motion planning algorithms.

C.1 Lazy Probabilistic Roadmap

The aim of the lazy probabilistic roadmap (LazyPRM) is to be faster than PRM, but otherwise having similar capabilities as the original PRM algorithm. Its aim is to find the shortest path in an initial roadmap, which is generated by randomly distributed configurations. If a path cannot be found between the initial and goal configuration, then the algorithm enhances the roadmap by adding more configurations to the roadmap. In contrast to the original PRM, the LazyPRM does not make use of collision checking in the preprocessing phase, to make sure that the samples in the roadmap are collision-free. Thus, the LazyPRM does not create a roadmap of only feasible paths. Instead, the algorithm builds a roadmap of assumed feasible paths. The idea is to evaluate the feasibility of the roadmap after a shortest path is found. If a collision is found in the path, the vertices and edges, which are in collision, are removed. Afterwards a new shortest path is calculated and checked for collision. This continues until a feasible path is found. This high level description of how the LazyPRM works is illustrated in figure C.1. By minimising the use of a local planner, and thereby the usage of the collision checker, then the computationally demands decreases and a feasible motion plan can be made more quickly. The LazyPRM algorithm is mainly made for single query tasks. [Bohlin and Kavraki, 2000]

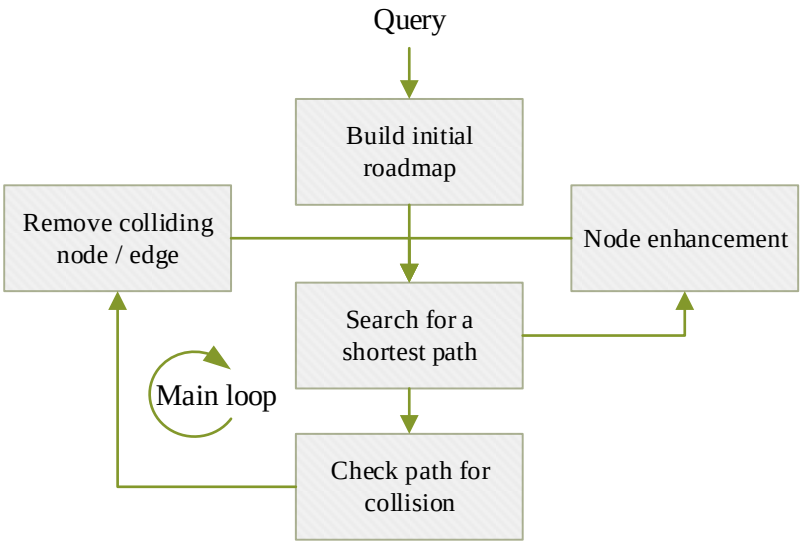


Figure C.1: High level description of the LazyPRM planning algorithm. [Bohlin and Kavraki, 2000]

C.2 PRMstar

The PRMstar algorithm is made to give asymptotically optimal solutions. This means if the number of samples goes towards infinity, then the solution found is the optimal solution. The standard PRM algorithm utilises a fixed radius to select which samples the current sample shall be connected to. PRMstar utilises the number of samples to define the radius. The result is that the connection radius decreases as the number of sampled vertices increases. This furthermore decreases the number of connections the algorithm attempts to make from a roadmap to vertices, as the number of samples increases. [Karaman and Frazzoli, 2011]

C.3 Sparse Roadmap Spanner

Sparse Roadmap Spanner (SPARS) constructs a sparser good-quality roadmap, by relaxing optimality guarantees by using graph spanners, for answering shortest-path queries. The algorithm is partly based on PRMstar, which provides asymptotically optimal solutions. The difference is that in addition to building the relatively dense PRMstar graph, and then a sparse spanner graph is built as well. The spanner is a sub-graph of the PRMstar graph. Vertices from the dense graph is included in the spanner if they are useful for coverage or if the vertices improves the path quality relative to paths of the dense graph. The reason for using a spanner is that this reduces the number of edges. [Dobson et al., 2013]

The motivation for using a sparse roadmap spanner and thereby reducing the roadmap size, while keeping

a good quality, is to comply with applications like resource-constrained manipulators, which can better communicate, store and query smaller roadmaps that are computed offline.

Sparse roadmaps with multiple good quality paths is beneficial in dynamic environments, where some edges might be invalidated by changes in the environment. [Dobson et al., 2013]

C.4 Sparse Roadmap Spanner 2

Sparse Roadmap Spanner 2 (SPARS2) is an updated alternative to the original SPARS algorithm. The aim for SPARS2 is to create a compact graph: This is done based on a roadmap and a spanner subgraph. The used roadmap is less dense than the PRMstar used as part of SPARS. The returned subgraph spanner is similar to that of the SPARS, but slightly denser. The computed paths is competitive to those of PRMstar and better than those of SPARS. The SPARS2 algorithm uses significantly less computational memory than the SPARS algorithm and has improved online capabilities compared to SPARS and PRMstar. [Dobson and Bekris, 2013]

C.5 LazyRRT

The algorithm of LazyRRT is similar to that of the original RRT. The difference is that, when moving towards a new configuration state, then the LazyRRT does not check for collision to make sure the path is valid. Thus, the algorithm attempts to find a path without checking for collision. When a path is found, it is checked for collision. If collision is found in the path, then the invalid vertices and edges are removed and the search process is continued. By decreasing the usage of the collision checker, then the computationally demands are decreased. [Karaman and Frazzoli, 2011]

C.6 RRTstar

The RRTstar algorithm is similar to the original RRT in that it first attempts to connect a new sampled vertex to the nearest vertex in the tree structure. The new vertex is added to the tree structure, if the connection attempt is successful. RRTstar differs, in that every time a new vertex is added to the tree structure, then connections are attempted from all other vertices within a given radius in the tree structure. Next, the algorithm removes the connections that are not part of a shortest path from the root of the tree to the new vertex. This avoids formation of cycles, as in roadmaps, and maintains the tree structure. [Karaman and Frazzoli, 2011]

C.7 LBT-RRT

Lower Bound Tree-RTT (LBT-RRT) is a single-query algorithm that has the properties of being *asymptotically near-optimal*. Being asymptotically near-optimal means that the solution of the algorithm is within an approximation factor of $1 + \epsilon$ of the optimal solution. For the LBT-RRT, this enables the possibility of interpolating between the RRT and the RRTstar. Thus when the approximation factor equals one, then LBT-RRT behaves like a RRTstar. When the LBT-RRT approximation factor is unbounded, then the LBT-RRT behaves like the original RRT. The aim is to have an approximation factor in-between. This combines the desirable properties of the RRT and the RRTstar. The result is LBT-RRT converges fast like RRT, and with a high-quality solution comparable to the RRTstar. [Salzman and Halperin, 2013]

C.8 Transition-based RRT

The transition-based RRT combines the original RRT algorithm with costmaps of the configuration space, thus the algorithm is divided into two stages; the RTT stage and the Transition stage.

The first stage is to utilise the RRT algorithm previously mentioned, however before inserting a newly sampled vertex, the second stage begins. The goal of the transition stage is to filter those configurations that do not comply with the cost-function. The filtering relies on the Metropolis criterion[1]. [Jaillet et al., 2010]

The implementation of a user-given cost-function in the RRT algorithm, as an additional input to RRT, enables the possibility to both produce a path that are feasible and has good quality with respect to the cost-function. [Jaillet et al., 2010]

C.9 RRTConnect

RRTConnect is based on the RRT algorithm, but in contrast, it is designed for path planning problems without differential constraints. The algorithm consists of two parts; the RRT algorithm, that is similar to the one of the original RRT, but the algorithm is however bi-directional. The second part is the Connect heuristic algorithm that is implemented instead of the incremental extender of the original RRT. The Connect heuristic iterates until the counter tree is found, or an obstacle is reached. The heuristic approach allows for rapid convergence to a solution. [Kuffner and LaValle, 2000]

[1] Commonly utilised in stochastic optimisation methods.

C.10 Path-Directed Subdivision Tree

Path-directed subdivision tree (PDST) is a tree-based planner, especially developed for systems with significant drift, severe under-actuated systems, and in "discrete system changes". Drift occurs in dynamically systems when the system cannot instantaneously stop. Under-actuation occurs when the dimension of the control space is less than the dimension of the state space. Discrete system changes occurs in hybrid dynamical systems, where hybrid dynamical systems are dynamic systems that exhibits both continuous and discrete dynamic behaviour, thus this causes discontinuous dynamic constraints or state variables. The PDST motion planner utilises a projection of a grid, to bias the tree to less explored areas. [Ladd and Kavraki, 2005]

C.11 Expansive Spaces Trees

Expansive spaces trees (EST) is a bi-directional single-query algorithm that is able to handle differential constraints. The algorithm builds the part of the roadmap that is connected to either the initial configuration or the goal configuration. [Hsu et al., 1997]

Thus, the idea behind EST is to have an initial configuration and a goal configuration. The algorithm then samples vertices at random in the configuration space, but only retains the vertices that are either connected to the initial or goal configuration. The result is two trees being built with root in the initial or goal configuration respectively. The two trees will keep growing until visibility regions of the two trees intersect each other. By visibility is meant if a straight-line path, joining the two trees lies entirely in the obstacle free space. [Hsu et al., 1997]

C.12 Search Tree with Resolution Independent Density Estimation

Search Tree with Resolution Independent Density Estimation (STRIDE) algorithm is a tree-based planner that is similar to EST, PDST, and KPIECE. The main difference is that these makes use projections to estimate the sampling density of the configuration space, whereas the STRIDE algorithm uses a data structure to produce density estimates directly in the configurations space. The effects is that the planner is guided towards unexplored regions of the configurations space. Experiments have shown that the algorithm has beneficial performance especially in high dimensional problems (10 or higher), but also within classical 6-dimensional problems. [Gipson et al., 2013]

Appendix D

OMPL Trajectory Post-Processing

The chapter introduces two post-processing operations in OMPL, PathSimplifier and PathHybridization. The post-processing is not within the scope of this project, which means that the set-up of the post-processing has been the same during the whole project and is treated as a "black box".

The idea behind the two post-processing operations are to optimise by shorten and smoothing the trajectories provided by the chosen motion planning algorithm. Thus, the quality of the resulting trajectory is also depended on the post-processing.

D.1 PathSimplifier

The pathSimplifier attempts to simplify the path, as the name implies. It consists of four functions of interest; *collapseCloseVertices*, *reduceVertices*, *shortcutPath*, and *smoothBSpline*. [OMPL Rice University, 2014b]

CollapseCloseVertices

The function tries to remove vertices that are close to each other, while still keeping the path valid with respect to the collision checker. This result in the function attempting to make short-cuts iteratively between non-consecutive states, thus states that are not directly followed by each other, but that are close along the path. If the function is successful, then the function removes any in-between vertices. [OMPL Rice University, 2014b]

ReduceVertices

The function tries to reduce the amount of vertices by doing short-cutting along the path, while still keeping the path valid with respect to the collision checker. The short-cutting is done by attempting to connect non-consecutive samples iteratively, meaning samples that are not directly followed by each other. If a connection is successful, then the in-between samples will be removed. [OMPL Rice University, 2014b]

ShortcutPath

The shortcutPath function also attempts to shorten the path while keeping it valid. It does so by attempting to connect between random points along the given path. Thus, the function does not only sample the vertices produced by the motion planner, but also intermediate points on the path. If needed, new vertices are created to shorten the path. [OMPL Rice University, 2014b]

SmoothBSpline

After the removal of vertices, this function attempts to smooth the path, while keeping the path valid. The smoothing is done by applying a B-spline on the path. The B-spline is first applied to a given number of maximal steps. If no progress is detected, meaning the states is updated less than a given minimum change, then the amount of steps on which the B-spline is applied, is reduced. The path of the B-spline is subdivided and the states along it is updated to improve the smoothness. [OMPL Rice University, 2014b]

D.2 PathHybridization

The PathHybridization class utilises the theory published in [Raveh et al., 2011]. The idea is that although motion planners often can be effective in finding collision-free paths, then the produced paths are often of poor quality, regarding standard quality measurements such as, path length, clearance, smoothness or energy. [Raveh et al., 2011]

With hybridization, the approach is to merge multiple paths into one high-quality hybrid path. The making of a hybrid path is based on observations that the quality of certain sub-parts of a solution may be of higher quality, than the entire path itself. Thus, a combination of these high-quality sub-paths will return a whole path of high-quality. The basic idea is illustrated in figure D.1. [Raveh et al., 2011]

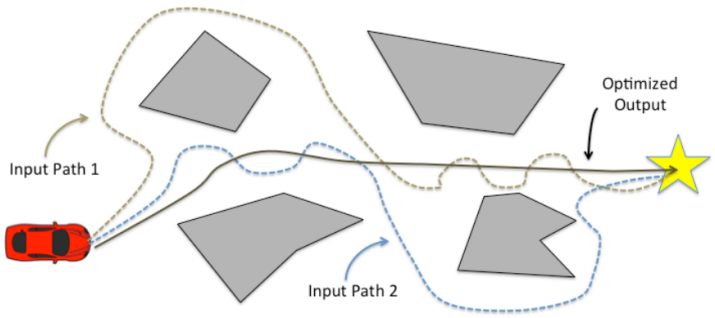


Figure D.1: The basic idea behind hybridization, where two sub-paths of high-quality are merged into one whole path of high-quality.[Luna et al., 2013]

Appendix E

Benchmarking in MoveIt

This appendix concerns the workflow for benchmarking in MoveIt. Next a description of the scenes, which have been used for benchmarking, is given.

E.1 Work flow of the Benchmarking

In figure E.1 the workflow for conducting a benchmark is illustrated. The Rviz motion planning plug-in is first started. This plug-in makes it possible to load scenes, in which manipulator movements can be performed in, by motion planning. These scenes are created by a *.scene* text file and can contain geometries, such as rectangles, balls, and cylinders. The *.scene* text file created for the benchmarking, is described in appendix E.2. After a scene is loaded, a query can be set by moving the initial and goal state of the manipulator. The scene and query is saved with a unique name to the Warehouse database. The Warehouse is utilised by the benchmark configuration file. The configuration file is used to set-up the benchmark, such as choosing the scene and query to use, number of runs, and the different motion planning algorithms to use. When the configuration file is set-up, then the benchmark can be conducted. The data from the benchmarking is saved to a log file, where different properties for each run are saved. The log files for containing the result is appended on the enclosed CD[1]. These data are then processed, to visualise and compare the results. Based on the results, a selection of motion planning algorithms with the desired characteristics can be made.

[1] See: <Benchmark/AroundThePole.txt> and <Benchmark/SmallSpaceTest.txt>

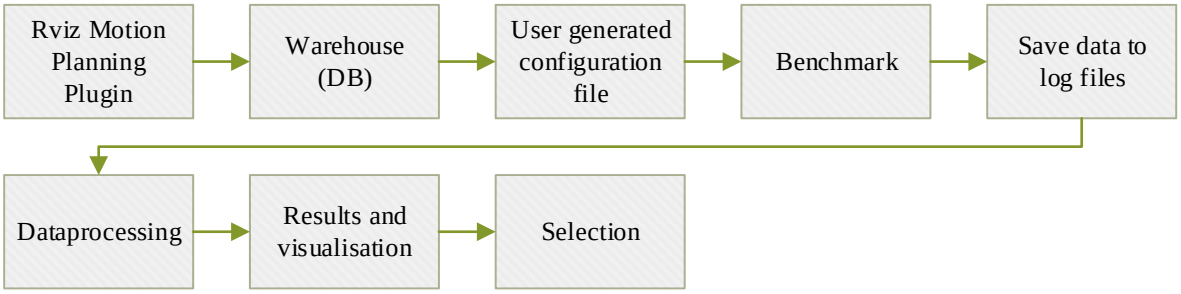


Figure E.1: The work flow of benchmarking. Inspired from [Coleman, 2013].

E.2 Scenes for Benchmarking

There has been conducted two benchmarks; one where the manipulator must move around the pan-tilt unit pole, and one where it has to move into a narrow passage. The first benchmark does only require itself (Little Helper 4), hence a scene is not needed. The second benchmark needs a scene for representation of a narrow passage. MoveIt scenes can be created in various ways, and the narrow passage scene has been created by manually creating a *.scene* file. A part of the *.scene* file is seen in table E.1, with a description for each parameter[2]. The entire *.scene* file can be found on the enclosed CD [3].

Parameter	**Description**
SmallSpaceBench	The scene name
* Wall1	Defines an object with the name Wall1
1	Number of shapes
box	The shape type
0.60 0.15 1.20	Shape measures; for a box it are its side-lengths
0.40 0.50 0.60	Translation (x, y, z) from origo
0 0 0 1	Orientation in quaternions
0.75 0.75 0.75 1	Color in rgba.

Table E.1: Parameters, including description, of the .scene file.

[2]Further information about the format can be found at http://moveit.ros.org/wiki/Scene_Format.
[3]See: <Benchmark/SmallRoomBench.scene>

Appendix F

Implementation of Motion Planning Algorithms in MoveIt

This appendix outlines the changes made to the MoveIt source code, to implement five additional motion planners, which was not contained per default. The changes concerns implementation of a number of planning algorithms into the OMPL planning files.

MoveIt, used throughout this project been embedded as a part of the Little Helper 4 ROS workspace. That is the MoveIt source code is cloned into the used workspace. This has been done for two main reasons; the MoveIt functionalities are built as a part of the workspace, thus no additional installation is necessary and it gives the ability to change the code to obtain the desired functionalities, as described in the following.

MoveIt includes 12 functional motion planning algorithms, based on OMPL, per default. The full list of all implemented algorithms can be seen on page 137. In addition to these, are another five implemented, which are listed below:

- LazyPRM
- PDST
- SPARS
- SPARStwo
- pRRT

The implementation is based on the existing implementation of the original 12 algorithms. The header files for each planner is present in OMPL, thus the implementation is limited to the planning_context

```
#include <ompl/geometric/planners/rrt/pRRT.h>
#include <ompl/geometric/planners/pdst/PDST.h>
#include <ompl/geometric/planners/prm/LazyPRM.h>
#include <ompl/geometric/planners/prm/SPARS.h>
#include <ompl/geometric/planners/prm/SPARStwo.h>
```

Table F.1: Inclusion of algorithm header files in the planning_context _manager.cpp file.

```
registerPlannerAllocator("geometric::pRRT", boost::bind(&allocatePlanner<og::pRRT>, _1, _2, _3));
registerPlannerAllocator("geometric::LazyPRM", boost::bind(&allocatePlanner<og::LazyPRM>, _1, _2, _3));
registerPlannerAllocator("geometric::PDST", boost::bind(&allocatePlanner<og::PDST>, _1, _2, _3));
registerPlannerAllocator("geometric::SPARS", boost::bind(&allocatePlanner<og::SPARS>, _1, _2, _3));
registerPlannerAllocator("geometric::SPARStwo", boost::bind(&allocatePlanner<og::SPARStwo>, _1, _2, _3));
```

Table F.2: Creation of allocators in the planning_context _manager.cpp file.

_manager.cpp file[1]. The file is written in C++.

The first step of the implementation, is inclusion of the algorithm header files, as shown in table F.1.

Each motion planning algorithm is created as a allocator (memory model), to be called through the ompl_planning.yaml file. The implementation of each planner is created with a planner ID and a reference to a configured planner. The planner ID is corresponding to the ID parameter, specified in the planning file. The configured planner is created by use of the boost::bind function, with placeholder arguments (_1, _2, _3). This means that the placeholder arguments are replaced with the input arguments, when the function is called. The implementations are listed in table F.2.

[1]The modified file is appended on the enclosed CD, see <Source code/planning_context_manager.cpp>

The total list of successfully implemented motion planners in MoveIt, both the pre-implemented and the ones described above, is shown below:

- RRT
- pRRT
- RRTConnect
- LazyRRT
- TRRT
- EST
- SBL
- KPIECE
- BKPIECE
- LBKPIECE
- RRTstar
- PRM
- PRMstar
- LazyPRM
- PDST
- SPARS
- SPARStwo

Three algorithms failed to be implement in MoveIt: STRIDE, pSBL, and LBT-RRT. The three planners current development status are "Experimental" [OMPL Rice University, 2014a]. Due to the amount of successfully implemented motion planning algorithms, no further investigation of why the implementation failed, was conducted.

Appendix G

Modelling Little Helpers in MoveIt

This appendix concerns the creation of the models of Little Helper 3 and Little Helper 4, utilised during the motion planning. The specification of geometric and kinematics information about the models are described. This is followed by an elaboration of the modelling of Little Helper 4. Then a brief introduction to concerned hardware on Little Helper 3 is presented, since the implementation of Little Helper 3 is lastly described.

G.1 Creation of Models of the Environment

The mesh models and kinematic information, utilised by MoveIt for the kinematic solver, collision checker, and self-filtering, are defined in a unified robot description format (URDF) file and a semantic robot description format (SRDF) file. The general structure and content of the two file types are described in the following subsections.

G.1.1 Unified Robot Description Format

A *URDF file* is a schematic description of a real robotic set-up. A URDF file consist of descriptions of the links and joints present in the robotic set-up. The URDF file is created in the XML format.

A *link* consist of a visual representation and collision representation. These can either be included as CAD files or directly specified as geometric shapes. The visual representation is used during visualisation of the robotic set-up, whereas the collision is used during collision checking. Mass, centre of gravity, and moment of inertia can furthermore be specified for each link. [ROS, 2013]

A *joint* specifies a parent and child link, along with a joint type. The rotation, translation, or surface normal axis, of each joint, is specified, along with a specification of the transformation from the joint

frame relative to the parent link frame. Each rotational joint is specified with limits for position and velocity. Joints can also be specified with dynamic effects, such as friction and damping, which have not been covered in this work. [ROS, 2013]

G.1.2 Semantic Robot Description Format

A *SRDF file* is a supplement to the URDF file, containing information about planning groups, states, and collision disabling. Planning groups are groupings of links or joints, for which MoveIt can create a motion plan. A planning group can be a kinematic chain containing the links from base link to TCP link for a manipulator. States are predefined joint configurations for a planning group, for example a home position. Collision can be disabled for individual pairs of links, for example adjacent links, which are allowed to have contact due to their connection [ROS, 2013]. A SRDF file can be created by used of the MoveIt tool *Setup Assistant*, specifying for example planning groups and states for kinematic chains. A SRDF file is created from a URDF file.

G.2 Environmental representation of Little Helper 4 in MoveIt

The model for Little Helper 4 consist of three main parts; the UR5 manipulator, the RQ3 gripper, and the platform. Meshes for the UR5 manipulator and RQ3 gripper are provided through the corresponding ROS packages[1]. Meshes are used to specify both a visualisation and collision part for each object. The meshes for the platform are based on a simplified CAD model. Simplified meshes are used to reduce the calculation time during collision checking [Pan et al., 2012]. The CAD model for the platform originates from the designing phase [VT3-2013, 2013], but is drawn to match the size of the actual platform. The platform is subsequent equipped with a pole for a pan-tilt unit, which is added to the environment as a part of the Little Helper 4 platform. The pan-tilt unit can be used to manipulate the mounted vision system, but is fixed through all work conducted in this project. The design of the pan-tilt unit is described in appendix I.

The following objects are included in the URDF file of Little Helper 4[2] [3]. Each of the objects are included as links or groups of links:

[1] The packages can be found at: `https://github.com/ros-industrial/universal_robot` and `https://github.com/ros-industrial/robotiq`

[2] The file is appended on the enclosed CD, see <Source code/lh4moveit/urdf/ur5_rq3_lh4.urdf>

[3] A graphical representation of the URDF file and the links and joints relative transformation and rotation is appended on the enclosed CD, see <URDF overview/LH4 URDF.pdf>

- Little Helper 4 platform
- UR5 manipulator
- RQ3 gripper
- TCP calibration
- RGB-D sensor calibration
- RGB-D sensor

The RGB-D sensor and calibration links are described in appendix H. The TCP calibration link is included to move the tool centre point, to the flange of the UR5 manipulator. The tool centre point is originally placed in the joint, between the two of the outermost links of the UR5 manipulator. This addition does also enable for specification of the TCP according to the used tool, by changing TCP calibration link accordingly. This can be changed without any precautions, when a numerical kinematics solver is used. If an analytical solver is used, the kinematics expressions must be created based on the changed kinematics of the manipulator.

The base link of the UR5 manipulator is placed relative to the Little Helper 4 platform link. The RQ3 gripper links and TCP calibration link are placed relative to the last link of the UR5 manipulator. The RGB-D sensor calibration link is placed relative to the UR5 manipulator, since the sensor link is calibrated relative to it, as described in appendix H. The collision representation for the palm of the RQ3 gripper was changed according to the description in chapter 10.

The set-up regarding the Little Helper 3, including the LWR manipulator, is described in section G.3. The environment for the dual manipulator motion planning set-up, described in chapter 9, is created as a combination of the URDF files for both Little Helper 3 and Little Helper 4[4] [5].

The SRDF file[6], based on the URDF file, has been modified subsequent to the creation from Setup Assistant. The changes include allowing of collision between the palm and fingers of the RQ3 gripper, because of the extended collision representation of the palm. Different group states have furthermore been added for the UR5 manipulator. These was used during the test phase of the system.

[4]The file is appended on the enclosed CD, see <Source code/kuka_lwr/robots/dual.urdf>

[5]A graphical representation of the URDF file and the links and joints relative transformation and rotation is appended on the enclosed CD, see <URDF overview/Dual-LH4_LH3 URDF.pdf>

[6]The files are appended on the enclosed CD, see <Source code/lh4_moveit_config/config/lh4.srdf> for the Little Helper 4 set-up and <Source code/dual_moveit_config/config/little_helper.srdf> for the dual manipulator set-up

G.3 KUKA LWR 4+ Manipulator

Little Helper 3 has been implemented in MoveIt, thus the concerned hardware of Little Helper is presented. Little Helper 3 consists of multiple types of hardware, such as a Neobotix MP-L655 mobile platform, a Schunck WSG 50 gripper, and a KUKA LWR 4+ (LWR) manipulator.

In this project the LWR manipulator, seen in figure G.1, is the equipment of concern on the Little Helper 3. This is because it is utilised to prove the capabilities of motion planning with multiple manipulators, as described in chapter 9.

Figure G.1: The KUKA LWR 4+. [METAL SUPPLY, 2010]

The LWR is a lightweight articulated manipulator, with a 7 degrees-of-freedom and in-line wrist. The LWR is approved for 7 kg payload, with an approximately weight of 16 kg, excluding the controller. [KUKA, 2012]

The LWR is, as the Universal Robots UR5, able to measure the force in each joint. Unlike the UR5 manipulator, which bases the force on current measurements in each motor, then the LWR bases the force on torque sensors in each joint. [KUKA, 2012]

The maximum work envelope of the LWR manipulator is 790 mm in radius and 1178.5 mm in height, from the base and up. [KUKA, 2012]

G.4 Environmental representation of Little Helper 3 in MoveIt

The process of implementing Little Helper 3 into MoveIt is identical to the implementation of Little Helper 4. Some challenges did however emerge, when implementing Little Helper 3, which this section mainly concerns.

To be able to implement Little Helper 3 into MoveIt, a package of the system, containing a URDF file and multiple CAD files for the respective parts, was needed. *The Robotics and Automation Group* at *Aalborg University* has previously implemented Little Helper 3 into other software applications, needing URDF and CAD files. Thus, a URDF file and the respective CAD models was available.

In theory, the process of implementing Little Helper 3 into MoveIt is the same, as described for Little Helper 4, but in practise, this did not work at the beginning.

A URDF file can in addition to XML format be in a XACRO format. In theory, the two formats are the same, except from the added possibility to include multiple URDF.XACRO files into one main URDF.XACRO file. Little Helper 3 is original created with URDF.XACRO files, where each equipment has its own URDF.XACRO, and the combined Little Helper 3 is created by a main URDF.XACRO file. MoveIt did fail, when loading URDF.XACRO files of Little Helper 3. MoveIt should be able to cope with URDF.XACRO files, but when creating the URDF.XACRO files into one URDF file instead, then the failing ceased. The conversion from a URDF.XACRO file to a URDF file was done by the command given in table G.1.

```
$ rosrun xacro xacro.py little_helper.urdf.xacro > litte_helper_expanded.urdf
```

Table G.1: Command for extending a URDF.XACRO file into a URDF file in XML format.

After changing the format type from URDF.XACRO to URDF, it was possible to load the file into MoveIt, without failing at start-up. The next challenge that emerged was that nothing visual appeared in Rviz. After comparing the COLLADA files of the LWR, with the COLLADA files of for example the UR5 manipulator, it became clear that the COLLADA files from Little Helper 3 included less libraries. After implementing additional libraries into each COLLADA files of the LWR, such as "<library_lights>" and "<library_cameras>", then the LWR manipulator appeared in MoveIt.

This was done for all COLLADA parts, to have the full Little Helper 3 visual represented in Rviz. It has been chosen to solely implement the LWR manipulator and the Schunk WSG 50 gripper, whereas the platform is a coarse model, made by the project group, with the outer dimensions of Little Helper 3[7].

[7]The COLLADA files are appended on the enclosed CD, see <Source code/kuka_lwr/kuka_lwr_mesh/meshes/kuka_lwr>

Appendix H

Implementation of a RGB-D Sensor in MoveIt

This appendix first outlines the gained knowledge of how to integrate a RGB-D sensor into MoveIt. Next, the hardware specification of the RGB-D sensor is presented. Finally a brief introduction, to how the sensor calibration is made, is given.

H.1 Utilisation of Sensors in MoveIt

To be able to have a dynamic environment representation in MoveIt, there is developed a plug-in named "octomap updater plug-ins"[1]. This includes the OctoMap library, which by means of octrees[2] is able to represent the surroundings of a manipulator. This enables the possibility to represent an octree by means of a sensor, which shall be able to produce 3D data. [MoveIt, 2014a]

An example of sensor input can be seen in figure 10.1b on page 89. OctoMap does not provide a driver for the RGB-D sensor; hence, a driver is needed to publish the raw 3D data to OctoMap. For this the OpenNI2 driver[3] is utilised, which is a common driver for multiple RGB-D sensors.

As part of the implementation, MoveIt needs the configuration settings for the RGB-D sensor, stated in table H.1, in a *sensors_rgbd.yaml* file[4]. An explanation of the different parameters, is listed at [MoveIt, 2014a].

[1] This plug-in is in the software named *PointCloudOctomapUpdater*, as shown in table H.1.

[2] Octrees are used to partition a 3D space, by recursively subdividing it into eight octants.

[3] The package for the driver can be found at: `https://github.com/ros-drivers/openni2_camera` and the package for launch of the driver is found at: `https://github.com/ros-drivers/openni2_launch`.

[4] The file is appended on the enclosed CD, see <Source code/lh4_moveit_config/config/sensors_rgbd.yaml>

```
sensors:
  - sensor_plugin: occupancy_map_monitor/PointCloudOctomapUpdater
    point_cloud_topic: /camera/depth_registered/points
    max_range: 0.8
    padding_offset: 0
    padding_scale: 0.1
    point_subsample: 1
    filtered_cloud_topic: output_cloud
```

Table H.1: The settings needed for implementing the RGB-D sensor, when using *PointCloudOctomapUpdater* [MoveIt, 2014a]. The settings has been modified to suit the needs of the Little Helper 4 set-up.

The sensor has been implemented in the URDF file of the set-up, such that MoveIt is able to project the sensed data relatively to the sensor. The name of the sensor is identical to the naming provided by the driver of the RGB-D sensor. This ensures the correct transformation is done between the 3D data and the rest of the environment. The overall structure of the URDF file, including the links related to the sensor is shown in figure H.1. MoveIt will not show any 3D sensor data from OctoMap until the transformation tree is set correctly. The driver provides "camera_link" in this case. In figure H.1 it is seen that an additional transformation is provided, named "camera_link_calibration". The transformation is made because MoveIt projects the 3D data along the X-axis, whereas the result from the calibration software package projects it along the Z-axis. The calibration is described in section H.3.

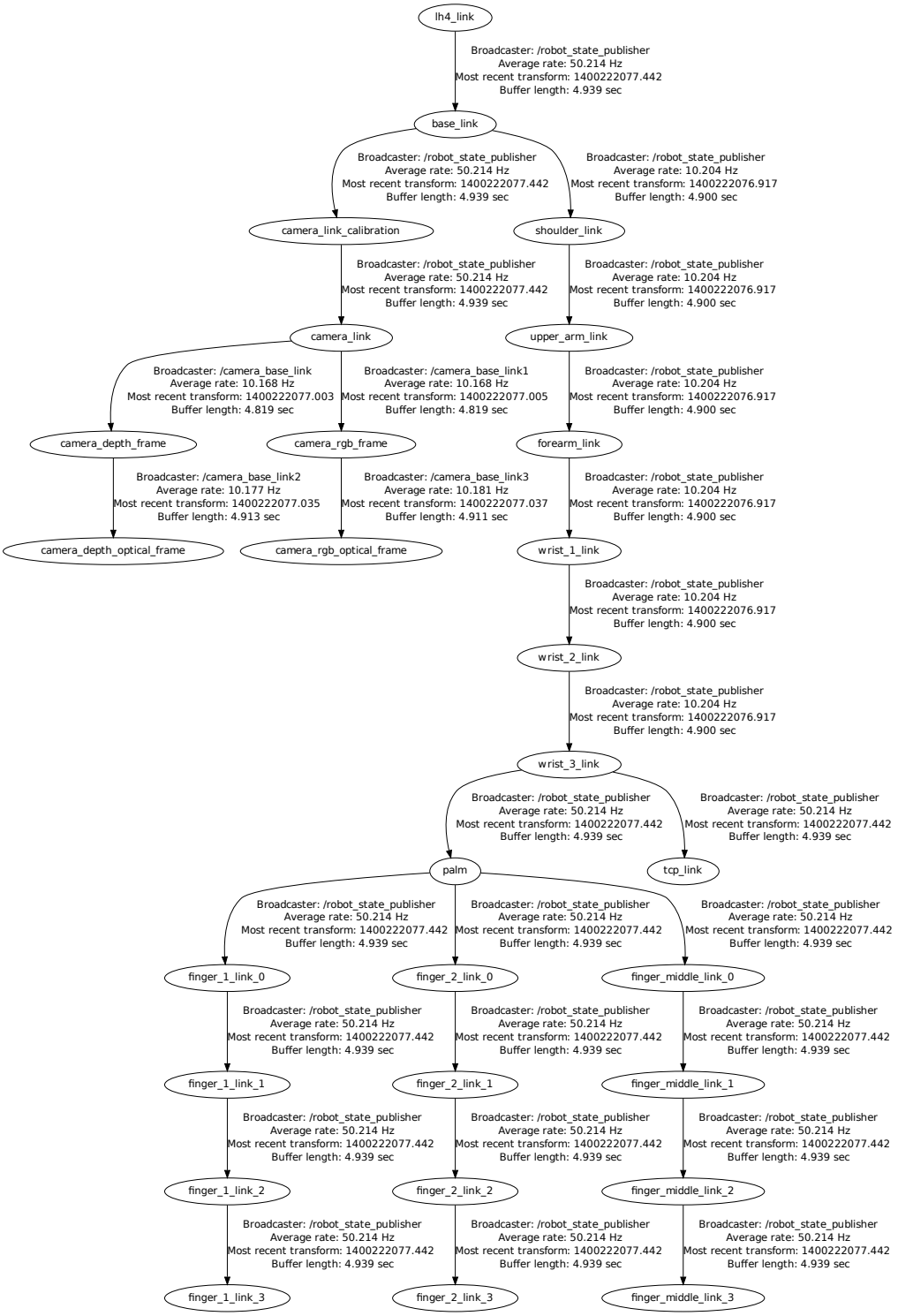


Figure H.1: An overview of the links of the environment of Little Helper 4, including links related to implementation of sensors.

H.2 Hardware Specifications

Hardware specifications for three RGB-D sensors are given in table H.2. The sensor is to be placed on the pole for the pan-tilt unit, approximately 400 mm above the platform. This placement entails that the PrimeSense Carmine 1.09 sensor is chosen to be used throughout this project, due to its range.

Attribute	**Microsoft Kinect for Windows**	**PrimeSense Carmine 1.08**	**PrimeSense Carmine 1.09 (short range)**
Range	400-3000mm	$800 - 3500mm$	$350 - 1400mm$
Introduction price	250$	200$	200$
Resolution / frame rate RGB	1280x960 / 12 fps 640x480 / 30 fps	1280x960	1280x960
Resolution / frame rate depth	640x480 / 30 fps	640x480 / 30fps 320x210 / 60fps	640x480 / 30fps 640x480 / 30fps
Field of view	43° vertical 57° horizontal	45° vertical 58° horizontal	45° vertical 58° horizontal
Dimensions	305x75x60mm	180x35x50mm	180x35x50mm

Table H.2: Selection of common RGB-D sensors [Muijzer, 2014]. In this project, the PrimeSense Carmine 1.09 is chosen.

H.3 Sensor Calibration

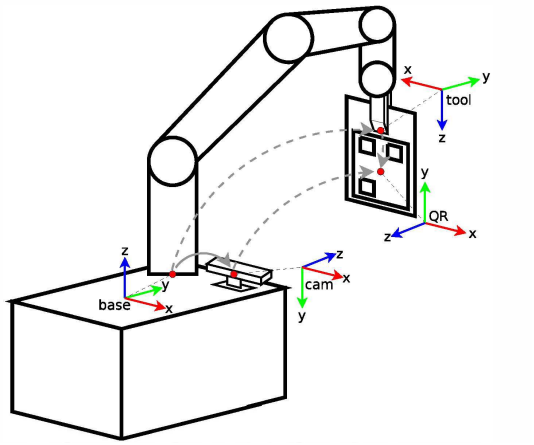
Calibration of the RGB-D sensor is needed, to estimate its position and orientation, relative to the environment model. It is necessary if the 3D data information, which the RGB-D sensor provides, is to be projected correctly in the environment. The calibration of the sensor is conducted by use of a method developed by [Andersen et al., 2013]. The aim for the method is to do fast and easy calibration, by means of a Quick Responds (QR) code. [Andersen et al., 2013]

The basic idea is that a QR code is placed with a known position in relation to the TCP of the manipulator. This is in practice done by conducting a well-defined grasp on a plate with a QR code on. The software recognise the corners of the QR code. After the QR code is correctly, detected, all 3D points inside the code are extracted. A plane is fitted to the extracted 3D points, from which a translation and an orientation of the coordinate system of the QR code can be produced. [Andersen et al., 2013]

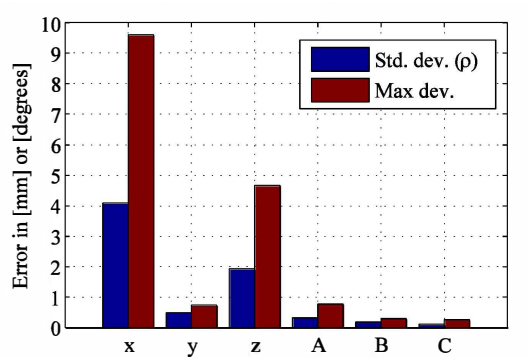
Based on subsidiary transformation from [Andersen et al., 2013], the following transformation between the QR code and the base of the manipulator is given:

$$ {}^{B}_{S}T = {}^{B}_{T}T \cdot {}^{T}_{QR}T \cdot \left({}^{S}_{QR}T \right)^{-1} \tag{H.1} $$

${}^{B}_{T}T$ is the transformation between the base of the manipulator and the tool, holding the QR code, which is predefined during previous calibrations. ${}^{T}_{QR}T$ is the transformation between the tool and the QR code, which is known from the inverse kinematics of the manipulator. $\left({}^{S}_{QR}T \right)^{-1}$ is the inverse transformation between the sensor and the QR code, which is based upon the translation and orientation found by measuring the QR code. Multiplying these, results in the desired transformation between the base of the manipulator and the sensor. The involved coordinate systems are seen in figure H.2a.



(a) The coordinate systems involved to calibrate the the sensor with respect to the base of the manipulator. [Andersen et al., 2013]



(b) The normalised errors in estimating the position and orientation of the sensor, where the x, y, z is translation in mm and A, B, C is the Euler angles in degrees. [Andersen et al., 2013]

In figure H.2b the normalised errors are seen for the respective translations and orientations. It is seen that overall translational precision is approximately $\pm 4mm$. [Andersen et al., 2013]

Appendix I

Contributions to the ACAT Project

This chapter treats the work done to fulfil goals for the ACAT project, stated by the Robotics and Automation group at Aalborg University, which has gradual begun to include Little Helper 4. It shall be clarified that most of the work done for ACAT during this semester, is not directly related to the scope of this project, which is why the documentation is found in this appendix.

The ACAT project is an EU project consisting of participants from institutes and universities from Germany, Lithuania, Slovenia, and Denmark. The goal of the ACAT project is to enable systems, like robots, to understand and use information which intentionally was made for humans[1].

The basis for the following contributions to ACAT, is the Little Helper 4 software framework created in [VT3-2013, 2013]. The framework and functionalities are described in section 2.2.

I.1 Read and Instantiate Specified Task File

The goal is to make the Little Helper 4 software framework able to instantiate task files, which are not instantiated. A task file, which is not instantiated, is in this context a task file containing a number of ordered skills, without values for the parameters. Algorithm 1 was used during the development of the functionalities:

The implementation of the pseudo code, in the software framework for Little Helper 4, is done as an addition to the existing terminal user interface (TUI), keeping all prior functionalities in the framework. This feature reads the skills in the specified task file, one by one, and asks the user to teach to needed parameters, and/or coordinates for each skill. This information is written to a temporary file, which

[1] More information can be found at http://www.acat-project.eu/.

Data: Load the "specified" task file
while *i < total number of skills* **do**
    Read the skill type number i, from "taskfile";
    Teach the skill type number i, to "taskfile1";
    i = i + 1;
end
Add skill type 0 Finished to "taskfile1";
Delete "taskfile";
Rename "taskfile1" to "taskfile";

Algorithm 1: Pseudo code for reading and instantiating of a specified task file.

replaces the original file after completion. This implementation does only concern the master, which is why this functionality is unaffected of the changes to the software framework, which is described in appendix 8.

I.2 Framework Compatibility Among Little Helper 4 and Little Helper 3

To make the Little Helper 4 software framework comply with the Little Helper 3 software framework, including GUI and teach functions, a set of additional functionalities is needed in the Little Helper 4 software framework. The two implemented functionalities, "Get Data" and "Set TCP" are described in the following.

I.2.1 Get Data

A "Get Data" function is needed to return the position of the TCP in both Cartesian and joint values. This need is fulfilled by adding a new set of keywords to the movetype- and inputtype-actionlib topics, ”get” and ”data”. These keywords are read by the proxy/driver and an "empty" function call with only the keywords are send to the URScript on the manipulator. The URScript returns the position of the manipulator, which is passed on to the master. This functionality concerns primary the proxy/driver and the URScript, why it must be considered during the further development of the ROS-Industrial Universal Robots package, described in appendix B.

I.2.2 Set Tool Centre Point and Payload for the Tool

A "Set TCP" function is needed to specify an offset TCP, relative to the original TCP of the UR5 manipulator. Furthermore, the possibility to set the mass and the centre of gravity is added to the "Set

TCP" function. Along with the keywords "set" and "tcp", the parameters are send from the master, to the proxy/driver via an actionlib. The proxy/driver sends the parameters to the manipulator, which enters a function that sets the parameters and returns a "succeeded" status to the proxy/driver. The proxy/driver changes the status of the actionlib to "succeeded". This functionality concerns primary the proxy/driver and the URScript, why it must be considered during the further development of the ROS-Industrial Universal Robots package, described in appendix B.

I.3 Pole for Pan-Tilt Unit

Part of the ACAT project requires the use of a vision system. This is to be mounted on a FLIR PTU-D48 E pan-tilt unit. To obtain a desired distance from the vision system, to the top plate of the platform, the pan-tilt unit is mounted on a pole. The pole is designed to be a bolt-on solution, to avoid reduction of the functionality of Little Helper 4 without the pole. The pole is bolted directly on one of the platform's top plates. The finished pole, mounted on the platform is shown in figure I.1.

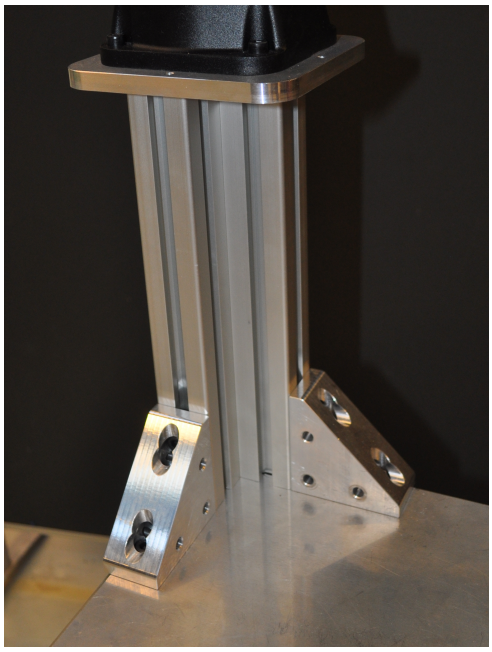


Figure I.1: Mounted pole for pan-tilt unit.

Appendix J

Multiple Manipulator Collaboration at MPCP

In the beginning of this project, the *World Conference on Mass Customization, Personalization, and Co-Creation 2014* (MPCP)[1], was held at Aalborg University, where the project group was asked to exhibit the Little Helpers in collaboration with the *Automation and Robotics group*.

At the exhibition, the Little Helper 3 and 4 were programmed to play a game of Tic Tac Toe, as seen in figure J.1. It was the first time the two Little Helpers operated in the same work envelope, thus it was required to implement a function, which could ensure collision free motion between the two. The information was shared between the two by publishing to a topic, which the other could get information from by a function call. The information published to the topic was a boolean, which was true if the Little Helper was at a safe position, and the other was free to move. This implementation only allowed one Little Helper to move at a time.

It was possible to share the information between the two, by connecting Little Helper 4 to the *ROS Master* of Little Helper 3 over TCP/IP.

[1] For more information about MPCP see `www.mcpc2014.aau.dk`

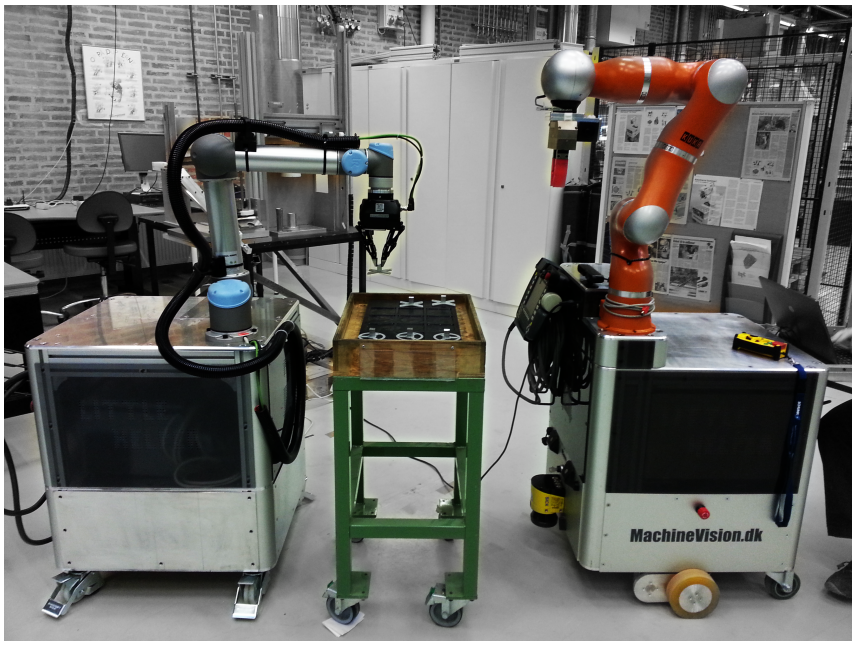


Figure J.1: The Little Helpers at the MPCP conference.

This project concerns incorporation of motion planning in a Skill-Based System (SBS). The vision with SBS is outlined first, whereupon the structure of it is explained. For this structure a long-term and short-term proposal is made for incorporation of motion planning. Among three, the motion planning software MoveIt is chosen for implementation.
A motion planning study is conducted for the sampling-based algorithms of the Open Motion Planning Library, which MoveIt utilises. A benchmark is made to select applicable motion planners in coherence with the theory examined in the motion planning study. After the selection, motion planning is implemented in the SBS to prove some of the concepts in the short-term proposal. Furthermore, motion planning with multiple manipulators and motion planning in a sensor-based environment is probed.

AALBORG UNIVERSITY
STUDENT REPORT