TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANTTI RUOKONEN

ENVIRONMENT- AND TASK-DRIVEN TOOL FOR SELECTING INDUSTRIAL ROBOTS

Master of Science Thesis

Examiners: Prof. Jose Martinez Lastra and Dr. Andrei Lobov
The examiners and the topic approved in the council meeting of the Automation, Mechanical and Materials Engineering Faculty on 6^{th} April 2016

ABSTRACT

ANTTI RUOKONEN: Environment- and task-driven tool for selecting industrial robots
Tampere University of Technology
Master of Science Thesis, 84 pages, 1 Appendix page
August 2016
Master’s Degree Programme in Automation Technology
Major: Factory Automation
Examiners: Professor Jose Martinez Lastra and Doctor Andrei Lobov

Keywords: robot selection, robot modeling, robot environment modeling, inverse kinematics, collision avoidance, kinematic roadmap, MATLAB, robotic toolbox, industrial robots

The problem of the research is to find a better solution for environment- and task-driven industrial robot selection process. Currently there are no tools or methods for the robot selection problem when considering an environment and a robot task. The goal was to find a solution for an industrial robot selection that takes the environment and the task into account and therefore make the robot selection process more simple and efficient.

This thesis solves an inverse kinematic problem within joint limits while avoiding collisions. Three tools were created using MATLAB to solve the industrial robot selection problem: Robot Selector for selecting industrial robots in the custom environment (modeled in OBJ-format) and task requirements, Robot Builder for creating robot model libraries and modeling custom robots and Environment Builder for creating robot environment models in OBJ-format. Website was designed and created for distributing the tools and a source code of the tools. The tools were converted into EXE-format and uploaded to website (robotselection.wordpress.com). The source code was uploaded to GitHub.

A robot selection algorithm was tested empirically with a qualitative method and with a quantitative experiment. The results were good: An inverse kinematic solver succeeded in all 200 cases. The robot violated a collision distance in 1 case out of 200. The cause of the problem got fixed after the experiment. The algorithm was tested with 2 devices. Average processing time with a desktop PC was 3.88 seconds and with a laptop PC 11.5 seconds. Three test subjects tested the tools and created a robot and environment models after getting familiar with the tools. The average modeling time was about 7 minutes with the Environment Builder and about 5 minutes with the Robot Builder. The robot selection took averagely 4 minutes with the Robot Selector.

TIIVISTELMÄ

ANTTI RUOKONEN: Ympäristö- ja tehtävä-perusteinen työkalu teollisuusrobottien valitsemiseksi
Tampereen teknillinen yliopisto
Diplomityö, 84 sivua, 1 liitesivu
Elokuu 2016
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Factory Automation
Tarkastajat: Professori Jose Martinez Lastra ja yliopiston lehtori Andrei Lobov

Avainsanat: robotin valinta, robotin mallintaminen, robotin ympäristön mallintaminen, käänteinen kinematiikka, törmäyksen välttäminen, kinemaattinen tiekartta, MATLAB, robotic toolbox, teollisuusrobotit

Tämän diplomityön tutkimusongelmana on ympäristön ja tehtävän huomioiminen robotin valintaprosessissa, sillä tällä hetkellä ongelman ratkaisemiseen ei ole metodia tai työkalua. Työn tavoitteena on löytää ratkaisu teollisuusrobotin valintaan huomioiden ympäristö ja robotin tehtävä. Tämä yksinkertaistaa ja tehostaa robotinvalintaprosessia.

Tämä työ ratkaisee käänteisen kinematiikan ongelman robotin nivelten rajoissa väistäen samalla ympäristön esteitä. Kolme työkalua kehitettiin käyttäen MATLAB-ohjelmaa tutkimusongelman ratkaisemiseksi: Robot Selector kehitettiin teollisuusrobotin valitsemiseksi omassa ympäristössä (OBJ-formaatissa) ja halutun tehtävän asettamissa rajoissa. Robot Builder kehitettiin robottikirjastojen luomiseksi ja robottien mallintamiseksi. Lisäksi kehitettiin Environment Builder omien robottiympäristöjen mallintamiseen OBJ-formaatissa. Tämän jälkeen työkaluille tehtiin internetsivu (robotselection.wordpress.com) ja työkalut käännettiin EXE-muotoon. Työkalut ladattiin internetsivulle kaikkien käytettäväksi. Lähdekoodi ladattiin GitHub-palvelimelle kaikkien käytettäväksi.

Robotinvalinta-algoritmi testattiin empiirisesti kvalitatiivisilla menetelmillä ja kvantitatiivisella tutkimuksella. Tulokset olivat erittäin hyviä: käänteisen kinematiikan ratkaisualgoritmi ratkaisi kaikki 200 testitapausta. Robotti alitti törmäysrajan yhdessä tapauksessa kahdesta sadasta. Virhe tuli myöhemmin korjatuksi. Algoritmi testattiin kahdessa laitteessa. Prosessoinnin keston keskiarvo pöytäkoneella oli 3,88 sekuntia ja kannettavalla 11,5 sekuntia. Kolme koehenkilöä opettelivat työkalujen käytön, jonka jälkeen he mallinsivat robotin ja ympäristön. Mallintaminen kesti noin 7 minuuttia Environment Builderilla ja noin 5 minuuttia Robot Builderilla. Robotin valinta kesti koehenkilöillä keskimäärin 4 minuuttia.

PREFACE

"The reward of our work is not what we get, but what we become."

-Paulo Coelho

This Master of Science thesis is made at FAST laboratory at Department of Automation Science and Engineering in Tampere University of Technology. This thesis is a continuation of a special assignment in factory automation that I wanted to finish proudly. I wanted to achieve results that really are useful for selecting industrial robots and I hope that my work will be utilized in the future.

Making this thesis was not a straight forward process. Even if I wanted to continue with this subject from the special assignment in factory automation, I had no idea how I can do it or even how long does it take to finish the project. Sometimes I spent a month without progress. However, I really liked to work with this subject and solving the main problem felt like solving a puzzle: When it finally got solved, I got a great rewarding feeling. Fortunately, I got the support from many people along the project. I want to thank them all.

I owe special thanks to Dr. Andrei Lobov for examining this thesis, interesting and challenging special assignment in factory automation and especially for an opportunity to continue the assignment as the Master of Science thesis.

I want also thank Professor Jose Martinez Lastra for examining this thesis.

I want to thank all my friends for support and especially those who have tested my tools and gave ideas. The greatest thanks belongs to my love Tiina for motivation, genuine interest and continuous support. Special thanks to my friends Arttu, Esa, Jyri, Ville and Viktor for great and relaxing weekends.

Last but not least, thanks to my family for supporting me along my studies.

Towards new challenges.

In Tampere on July 8^{th}, 2016

Antti Ruokonen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

a	Link length
d	Link offset
α	Link twist
θ	Joint angle
Σ	Joint type (revolve or prismatic)
CAD	Computer Aided Design
CCD	Cyclic-Coordinate Descent
DH	Denavit-Hartenberg
DOF	Degree of Freedom
FAST-lab	Factory Automation Systems and Technologies Laboratory
GTMA	Graph theory and matrix approach
GUI	Graphical User Interface
MCDM	Multi-criteria decision making
OBJ	Wavefront OBJ –format
RRT	Rapidly-exploring Random Tree
UML	Unified Modeling Language

1. INTRODUCTION

This thesis work starts with introducing the subject, the background and different methods of the main problem, which is inverse kinematics with collision avoidance. Later in the thesis one of the analyzed methods will be selected and modified if necessary. This will be discussed in the approach chapter. Implementation chapter reviews how the approach method was implemented and what was done for solving the problem. After implementation an experiment and its results are discussed. A next chapter will include general discussion of the implementation and the results of the experiment. Then the thesis ends with conclusions.

1.1 Motivation

Selecting an industrial robot is not an easy process, especially when the robot selection is environment- and task-driven. The robot must be suitable for selected task and environment. Nowadays there are no free tools which can be used for selecting the industrial robots regarding the task and the environment. The existing tools are mostly robot sizing tools that checks only the basic task requirements like maximum payload and reach without considering the environment. These tools are in most of the cases designed to select robot models from the tool's manufacturer. For now, engineers must select the industrial robots based of their knowledge and skill to estimate the robot's suitability to environment, when considering the environment in the robot selection.

A working environment of the thesis work is MATLAB with Robotic Toolbox. MATLAB is a fourth generation programming language. MATLAB is a strong tool for handling matrices and numerical computing. MATLAB has also great plotting tools and algorithm implementation. MATLAB was known from my previous works, so this makes it easier to start thesis in this environment. Capabilities and limitations of MATLAB were also already known. Robotic Toolbox is set of algorithms to MATLAB robot modeling. These are the main reasons for selecting MATLAB as the working environment.

1.2 Problem statement

The main problem of the thesis work is to find and implement or create a new algorithm of inverse kinematics that avoids collisions to environment. Inverse kinematics means generating robot joint values that leads a robot tool to required coordinates with a given orientation. This must be done without collisions and within robot joint limits.

To find a suitable industrial robot while considering environment and a task, the inverse kinematic problem must be solved in the joint limits of the robot while avoiding the collisions. The robot cannot be suitable if it collides to the environment. A payload and other task requirements must also be checked. This requires an algorithm that solves the inverse kinematic problem without the collisions or violating the joint limits of the robot and compares an attributes of the robot to requirements of the task. The algorithm also requires models of the environment and the robot. The solutions must be general so it can be utilized with any serial manipulator industrial robot and with any environment.

1.3 Research hypothesis

By solving the inverse kinematic problem within the joint limits while avoiding obstacles of the given environment, it is possible to solve the environment-driven robot selection problem. When this problem is solved the robot is suitable to the environment. By creating adequate models of the robots and the environment, it is possible to avoid collisions to the environment utilizing the created models with a right algorithm. By checking the task requirements of the robot, it is possible to find out the suitability of the robot in the given task. Creating and sharing the solution built on appropriate frameworks like MATLAB, it is possible to make the environment and task-driven industrial robot selection process more efficient and simple.

1.4 Objectives

The objectives of the thesis are to solve the inverse kinematic problem within the robot joint limits and avoid obstacles of the environment and checking the task requirements of the robot. The solutions should be general and work with any environment and any robot. Reaching the goal may result providing a single or multiple tools or algorithms for solving the industrial robot selection problem. Then a source code or the tools and the algorithms are distributed to users.

1.5 Limitations

The limitations of the thesis ensured from the working environment. The thesis is limited to the MATLAB-code. The algorithms require certain toolboxes and libraries like Robotic Toolbox. This environment limits the expendability of the source code, since MATLAB is not as common language as for example C++, even if MATLAB-files can be converted to C++ with converters.

2. THEORY AND BACKGROUND

In this chapter background and theory of industrial robots and the project are discussed in general. The chapter focuses mostly to the robot selection and robot kinematics.

Robots have certain specifications. Depending of the task, some features are more important than others. When selecting the industrial robot, it is important to know the most important qualities. This chapter discusses where to focus in the environment- and task-driven robot selection.

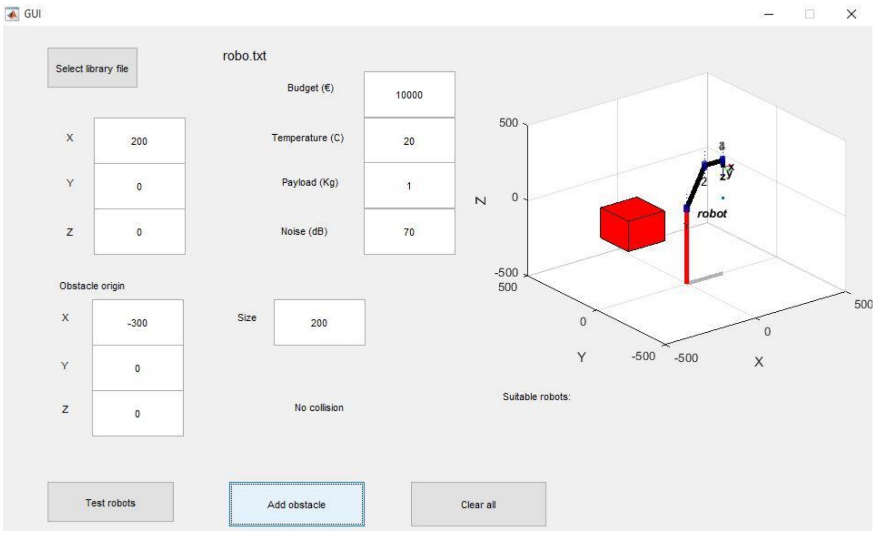
In this chapter different methods are reviewed for solving the problems of this thesis. Inverse kinematic solvers are reviewed. In this thesis we do not consider inverse kinematic problems with velocities because in this thesis the problem is a point-to-point inverse kinematic problem.

After reviewing the inverse kinematic solvers, collision detection methods are discussed. Collision detection methods are useful to understand when implementing the collision avoidance. After discussing the collision detection, different obstacle avoidance methods are reviewed.

2.1 Background of the project

This thesis started as a special assignment in factory automation. The given assignment was to create a tool allowing the modelling robot operational environment and required robot tasks. The tool should be based on the environment details to calculate which robot can be suitable for it. The tool should provide the best structure for the robot, as well as be able to check existing robots if those will fit for the task.

The tool was made using MATLAB and Robotic toolbox version 9.7. This implementation utilized custom robot library file format (txt). The user can import an environment model and a robot library file to the tool using Graphical User Interface (GUI). Using the GUI user can enter environment, robot and target details and run a robot suitability algorithm. The tool gives list of suitable robots to the given task and environment as output. *Figure 2.1* shows the tool as it was in special assignment phase.



robo.txt

X 200
Y 0
Z 0

Budget (€) 10000
Temperature (C) 20
Payload (Kg) 1
Noise (dB) 70

Obstacle origin
X -300
Y 0
Z 0

Size 200

No collision

Suitable robots:

Figure 2.1 *Environment-driven robot selection*

The robot libraries were made manually with text editors and user had to know the right syntax. The environment files had to be made with separate 3D-modelling tool like SolidWorks. The tool did not detect collisions. Environment models were only for choosing robot's and target's location coordinates, but the environment does not actually affect the robot's inverse kinematics in any way. In some situations, the robot can even collide with itself so the robot's ability to reach close target points must be double-checked with another method. The problem of this tool was also the fact that the tool requires the robot library file made with notepad and also model file of the environment in obj-format.

The given assignment was interesting and challenging but it required large amount of work to finish the goals. It was decided ask to continue the assignment as the Master of Science thesis work.

2.2 Selecting an industrial robot

For now, there is no simple and neutral tool for selecting an industrial robot. Especially when considering tasks and a working environment of the robot. Selecting the robot requires knowledge and lot of comparing and consideration [35].

Most common industrial robots are serial manipulators. Industrial robots are usually nowadays capable of different tasks and applications by utilizing different grippers and tools. Still robot manufacturers have models for different purposes. [11, 19]

In a robot selection a task and environment should always be taken into account. The robot must be capable to the selected application. The selected robot should also be able to reach each point of the task and produce sufficient torque. If overload occurs, the robot can even shut down. Budget or schedule can also affect the robot selection. However, a decent and most suitable robot is usually worth of money. [11, 19]

2.2.1 Task-driven robot selection

Application might be the most important factor in the industrial robot selection. Each application makes individual requirements for the robot selection. For example, small part assembly requires completely different type and size robot than welding application. Different application types can be for example welding, painting, packing, assembly, tending or measuring. The application type should be the first thing taken to account when selecting an industrial robot. This determines the type of the required robot. the robot types can be for example SCARA-, delta- or collaborative-robot. However, this thesis and the robot selection tool focuses only to industrial robots. Nowadays the industrial robots are able to execute large amount of different task like welding, tending or painting. Robot manufacturers have an industrial robot model for almost all different applications. [11, 19, 32, 34]

Payload is an important factor when selecting the industrial robot. The payload means the maximum amount of mass the robot can handle. The payload includes also mass of gripper of the robot. There are completely different set of the robot models for handling heavy products comparing for example to fast small part assembly robots. [11, 19, 32, 34]

A workspace of the robot is the area where the robot can reach. This must be considered to make sure that the robot can reach and work in the desired point. Different industrial robots have different workspaces. All the robot manufacturers inform their workspaces of the robots. Before selecting the robot, it should be checked that the robot can reach all the task points with right orientation. [11, 19, 21, 32, 34]

Repeatability and accuracy are important aspects in certain tasks. Repeatability is the robot’s ability to repeat exactly same movements. Accuracy is the robot’s ability to move close to the desired location. For example, the robots that work with circuit boards require often good repeatability and accuracy. The robots must handle small parts repeatedly to exactly same location unlike for example welding tasks. [11, 19, 21, 32, 34, 40]

Speed has different importance with different tasks. For example, packing tasks usually requires the robots with high speed. In some situation even fast delta robots are required just because of the speed. When considering the robots’ speed, it is important to find out actual working speed of the robot: Some manufacturers inform speed of their robots with maximum acceleration rate so the given maximum speed cannot be reached with every task. The speed is given in degrees/second in most of cases. [11, 19, 32]

An inertia of the robots should also be checked when selecting the robot. All tasks require certain amount of torque. All the robot manufacturers inform maximum torques of their robots. If the robot faces overload, it can even shut down. [11, 32, 40]

The robots have different amount of brakes. Some of the industrial robots even have the brakes in every axes. The brakes might be helpful when avoiding collisions. [11, 40]

2.2.2 Environment-driven robot selection

The environment causes requirements to the robots. In some situations, the environment can be designed around the robot but in some situations this is not possible. The robot might be required to reach a task point in the narrow environment. The environment can also cause limitations with a robot mounting or require extra protection against dust, water or flammable materials. [11, 16, 34]

The environment can also cause the robot to reach tricky task points. Even if the points are in the work space of the robot, the environment may constrain the robot to complex position when evading the environment. Less the robot has degrees of freedom, harder it is for the robot to reach the task point in the restrictive environment. More joints allow more different ways to avoid the objects. [11, 16, 19]

Number of axes of the robot is related to the robot's degrees of freedom. The different environments and the different tasks requires different amount of axes. Product movement does not require large amount of axes except if the robot is in a cramped environment. However, few extra axes are not a problem. Actually the extra axes increase flexibility of the robot increasing the robot's degrees of freedom and redundancy of the robot. The redundancy of the robots is discussed in more detailed later in this chapter. [11, 16, 19]

The robots should always have some kind of collision avoidance. Usually the robots are programmed to reach predetermined points with a predetermined orientation. This works until the environment is changed. Machine vision is also used for the collision avoidance. In some solution the environment is modeled and the robot is programmed to avoid the collisions based to the environment model. [11]

Mass of the robot can be important in some situations. If the robot is mounted to for example to rails, a ceiling, a wall, a custom bench or any unusual way.

Some of the environments can require for right IP rating of the robot. The robot may require protection for example against dust, water or flammable environment.

2.2.3 Robot ranking methods in robot selection

Different kinds of industrial robot ranking methods are developed for the robot selection problem. For example, multiple multi-criteria decision-making (MCDM) methods are created for the robot selection problem. In these methods, the robots are ranked based on their attributes such as cost, payload, size, degrees of freedom, programming flexibility and velocity. The attributes can be subjective or objective meaning numerically expressible or not numerically expressible. The different attributes have different importance in each application. The attributes are used to rank the robots and calculating the most suitable robot. [32, 34, 35]

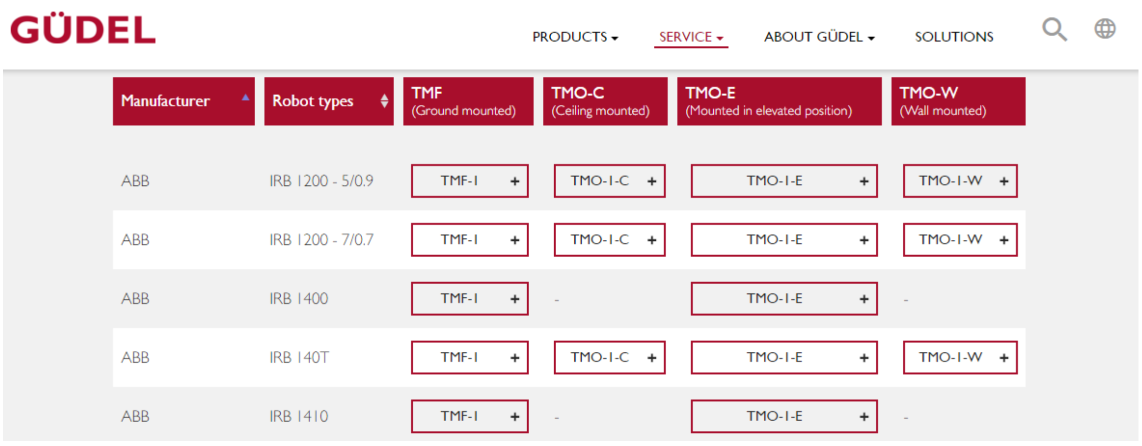
As an example, in graph theory and matrix approach (GTMA) method the robots are first collected in a table. The robot table includes robot names and different robot attributes. In a next phase, data of the table is normalized. This means that for example if the greatest payload value is 60kg, it is turned to value of 1. The payloads of the other robots are indicated as percentage of this maximum value. Then an attribute matrix is generated for the normalized data. The robots can be arranged using a robot index number calculated from the matrix. [35]

The different robot ranking methods use different weight of importance to the attributes. This results different robots as being the best choice with the different selection methods. [35]

However, the robot ranking methods are not suitable for the problem of this thesis. The robot selection problem is not solved using ranking methods since they do not consider environment. However, these methods may be utilized for ranking the robots if multiple robots are suitable for the task and environment.

2.2.4 Existing tools

Nowadays there are few tools for selecting an industrial robot. However, none of them takes environment and task an account. Most of the tools are simply taking into account only robot size or application. In *Figure 2.2* one of the robot sizing tools is shown. The particular tool is mostly based for the robot's mounting.



Manufacturer	Robot types	TMF (Ground mounted)	TMO-C (Ceiling mounted)	TMO-E (Mounted in elevated position)	TMO-W (Wall mounted)
ABB	IRB 1200 - 5/0.9	TMF-I +	TMO-I-C +	TMO-I-E +	TMO-I-W +
ABB	IRB 1200 - 7/0.7	TMF-I +	TMO-I-C +	TMO-I-E +	TMO-I-W +
ABB	IRB 1400	TMF-I +	-	TMO-I-E +	-
ABB	IRB 140T	TMF-I +	TMO-I-C +	TMO-I-E +	TMO-I-W +
ABB	IRB 1410	TMF-I +	-	TMO-I-E +	-

Figure 2.2 *Robot sizing tool [18]*

Most of the manufacturers' robot selectors are especially robot sizing tools. There are also robot programming and modelling tools. These tools are able to import an environment model. After importing the environment, these tools can often detect robot collisions when executing the robot task with the selected robot.

Catia and Delmia are an example of a tool that is used for plan, manage and optimize robots. Catia is the tool for 3D-modeling. With Catia, a user can model robot parts or any other 3D parts. After a modeling phase, the modelled parts can be assembled into the robot model.

In Delmia, the robots can be defined in more detailed. For example, joint rotation limits can be set. The environment of the robot can also be assembled. After finishing the robot and the environment, the robot task can be made. The robot can be commanded to execute the created task. For example, the robot task can be picking an object and placing it to another location. This is done in the robot environment so Delmia is able to detect all collisions. *Figure 2.3* shows ABB robot executing a task in its working environment.

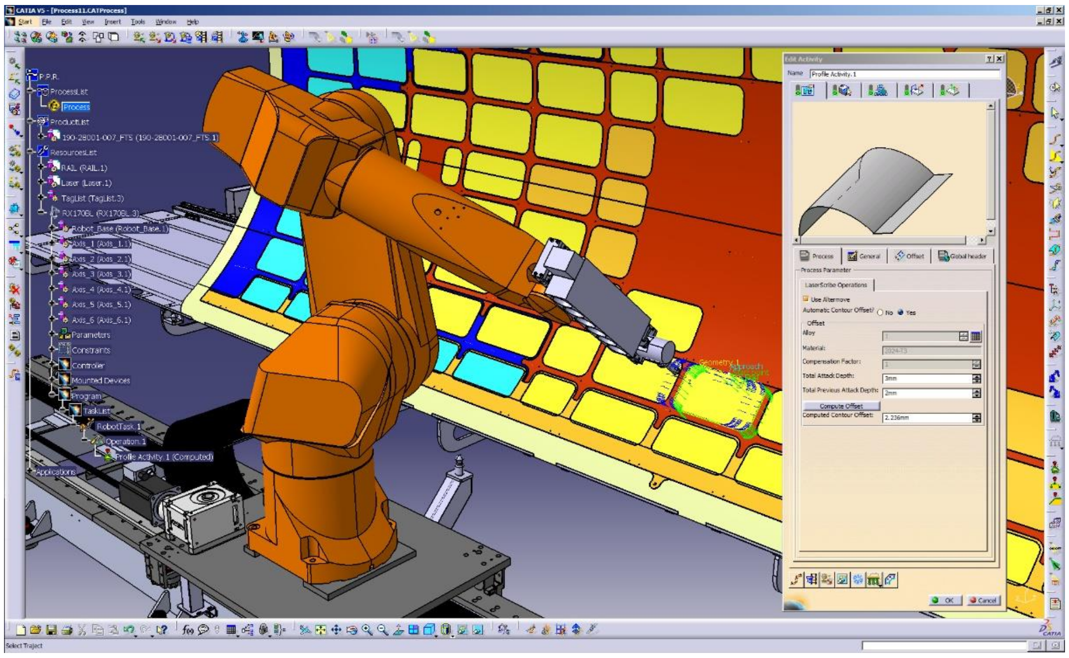
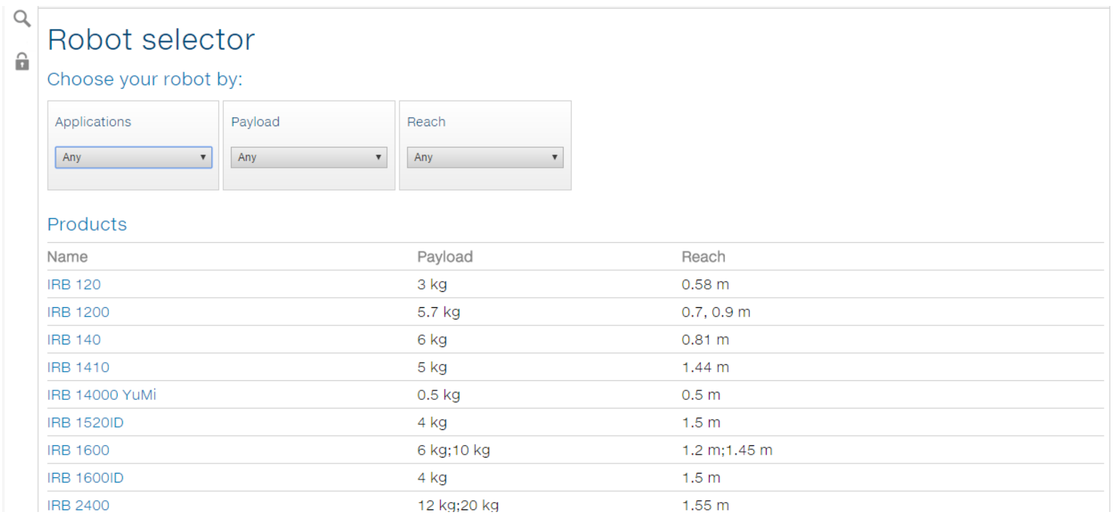


Figure 2.3 *ABB robot in Delmia [12]*

However, Delmia is not suitable tool for the robot selection even it can be used for checking suitability of the robot for the task and the environment. Delmia is an expensive program and it also requires to select the robot first before executing the task in the environment. In the robot selection process user needs to know which robots are suitable for the certain environment and task. Selecting the industrial robot with this kind of tools is time consuming and ineffective way to select the right industrial robot.

Some of the robot manufacturers have a tool for selecting robots. However, these tool are simple. Most of the tools made by robot manufacturers are designed only for robot sizing. For example, ABB has a tool for selecting the industrial robots. This tool is shown in *Figure 2.4*. The robot selecting tool asks application, payload and reach of the robot. After selecting the right values, the tool lists the suitable robots. This kind of tool is a simple way to select right ABB robot. However, this type of tools does not consider the environment. As result of this thesis, the outcome may be similar type of tool that also considers the environment. [1]



Robot selector

Choose your robot by:

Applications: Any
Payload: Any
Reach: Any

Products

Name	Payload	Reach
IRB 120	3 kg	0.58 m
IRB 1200	5.7 kg	0.7, 0.9 m
IRB 140	6 kg	0.81 m
IRB 1410	5 kg	1.44 m
IRB 14000 YuMi	0.5 kg	0.5 m
IRB 1520ID	4 kg	1.5 m
IRB 1600	6 kg;10 kg	1.2 m;1.45 m
IRB 1600ID	4 kg	1.5 m
IRB 2400	12 kg;20 kg	1.55 m

***Figure 2.4** ABB's robot selector [1]*

The manufacturers' robot selecting tools are limited to models of the manufacturer. It is obvious that no manufacturer includes the robots to the tool by other manufacturers. These tools are also limited with the robot selection criteria because the robot selection is based to size and application of the robot.

2.3 Summary of existing robot selection methods

After making the research on the internet and literature, the only way to select the industrial robots considering the environment and the task appears to be by modeling the environment of the robot and the robot itself by using an expensive software and modeling the robot executing the task. Then it can be seen if the robot collides to the environment and then the robot or the environment can be changed and remodeled if necessary. This is time consuming task. In addition to this, other requirements must also be checked manually like payload limits of the robot.

***Table 2.1** Summary of robot selection tools and methods*

Method	**Solves problems**	**Description**
Robot ranking methods	Robot sizing and task eligibility	Robot ranking methods is made for list robots in order of suitability with self-defined importance of robot attributes.
Robot sizing tools	Robot sizing and task eligibility	Simple tool mostly created by robot manufacturers for finding suitable robot with right size for selected application.
Robot programming environments	Robot task and environment eligibility	Tools used for programming robots can be used for checking robot suitability for given task in custom environment.

Table 2.1 summarizes the current methods and tools for selecting industrial robots. Only the robot programming environments can be used for testing the robot task and the environment eligibility. However, this is a slow method for finding the suitable robot and the tools like Delmia are often expensive. As we can see from the table, there are no tool or method for the task- and environment-driven robot selection problem except the robot programming environments.

It appears that collision avoidance has not yet been combined with the robot ranking or robot selection. We can now conclude that the problem is not yet solved at the formulated level. There is currently no decent tool or method for selecting the suitable robots out of a robot library (a set of robot models) for the task taking the environment into account.

2.4 Modeling robot manipulators

Before we can select the suitable industrial robots for a certain task, the robots must be modeled. Utilizing the robot models, we can test the suitability of the robots for the environment by checking collisions.

Robot kinematics focuses robot joint movements and locations without considering causes of the forces. The robot kinematics includes joint positions, velocities and accelerations. In this thesis we focus to the robot position and orientation kinematics. [11, 19]

The industrial robot can in most cases be thought of as chain of links. There are joints between the links. There are six different joint types: revolute, prismatic, cylindrical, planar, screw and spherical joints. This chain type robot structures are called serial manipulators and they are the most common industrial robots. Most of the serial manipulators have an anthropomorphic structure. This structure includes shoulder, elbow and wrist type joints. Some of the serial manipulator industrial robots are shown in *Figure 2.5*. As we can see, these robots are serial manipulators with open loop structures, because there is only one link leading to each joint and therefore no parallel structures. [11, 19, 21]

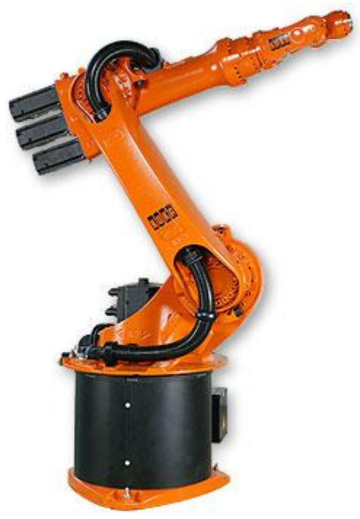


Figure 2.5 *KUKA industrial robot with serial manipulator structure [23]*

Other type of the industrial robots is a closed loop robot. The parallel robots have multiple links from the robot base to the robot tool. This limits workspace and degrees of freedom of the robot, but significantly increase speed and precision of movement of the robot. The parallel robot type is shown in *Figure 2.6*. As we can see from the figure, the particular delta robot has three identic serial links connected to the tool. This enables fast accelerations and accuracy because of three motors are sources of force. [11, 19]

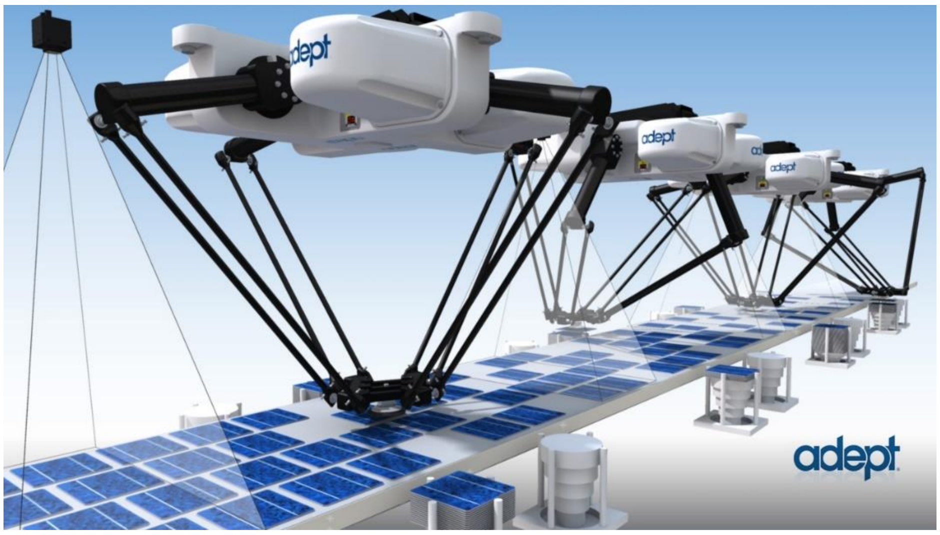


Figure 2.6 *Delta robots with parallel structure [10]*

Because this thesis focuses to the environment- and task-driven robot selection, the parallel robots are not included. The parallel robots have limited workspace and limited amount of degrees of freedom so they are not capable for example avoiding obstacles.

2.4.1 Robot degrees of freedom

Degrees of freedom (DOF) relates the robot’s capability to reach a point in X, Y and Z coordinates and orientations. For example, a six DOF robot can reach any point in its workspace with any orientation along X, Y and Z axes. All the robots with over six DOF are always called redundant robots. [19]

DOFs can be counted by counting the joints of the robot. Usually each actuator increases DOFs of the robot. Some of the joints may have multiple DOFs. For example, human shoulder has three DOFs. [11]

DOF has its limits called a configuration space. Most of the joints have their own movement limits. These limitations can be ensured for example from wires, actuator limits or servo max angles. [11]

2.4.2 Robot DH parameter table

Generally, a base of the robot is called link0. The next link is called link1 and so on until the end-effector. Most of the manipulators have five or six joints. In this thesis, we consider only the serial link manipulators ignoring the parallel robots. [11, 19]

Denavit-Hartenberg notation is a mechanism for defining the serial links. There are four parameters for each link in Denavit-Hartenberg notation. These parameters are called DH parameters. Two parameters are for defining adjacent joint axes and the another two parameters are for defining the adjacent links. [11, 19, 38, 40]

Relationship between the joint axes can be defined with two parameters: a link twist α and a link length a. The link twist α is an angle between the adjacent joint axes. The link twist is measured along a plane whom normal is the adjacent links mutual perpendicular. The link length a is measured along a line that is mutually perpendicular to the both joint axes. The link length is also known as r to avoid confusing to the link twist α. [11, 19, 38, 40]

The adjacent links have one common joint axis. Parameter d is a distance along this common axis from the previous link to the next one. This parameter d is called link offset. Another parameter defining relationship between the neighbor links is a joint angle θ. The joint angle is amount of rotation between the adjacent links from their common joint axis. With the revolute joints joint angle is often a variable that changes with the robot movement and with the prismatic joints, d is a variable. *Figure 2.7* clarifies the robot DH parameters. [11, 19, 38, 40]

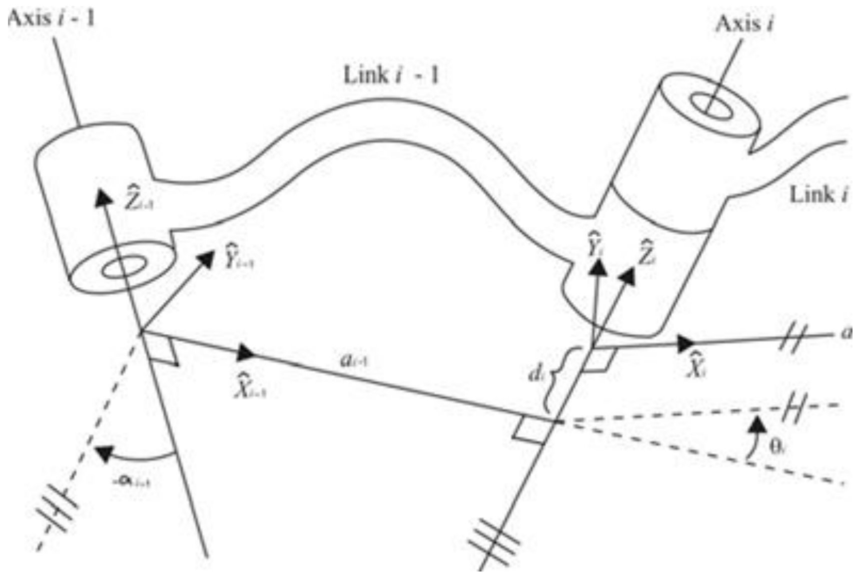


Figure 2.7 *Robot DH-parameter clarification [19]*

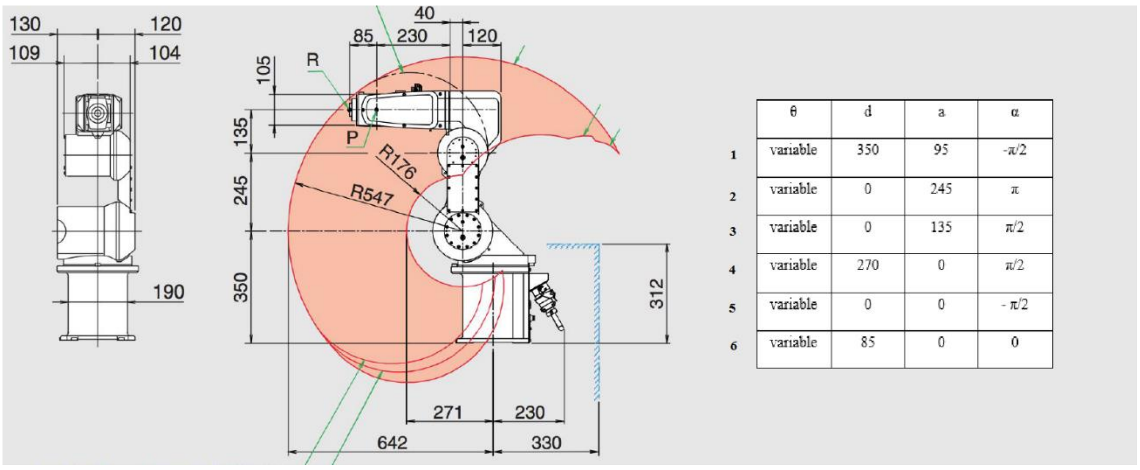
In Robotic Toolbox, there is also a fifth parameter Σ. This parameter defines a type of the joint. Value 0 is for the revolve joint and 1 for the prismatic joint. However, this is not an original DH parameter. Now we can summarize the DH parameters:

> θ = Joint angle (The angle between an adjacent links measured from their common joint axis).
>
> d = Link offset (The distance between an adjacent links measured along their common joint axis).

a = Link length (The distance between an adjacent joints measured along a mutual perpendicular).

α = Link twist (The angle between an adjacent joint axes measured from a plane whom normal is adjacent links mutual perpendicular).

Now we are able to model the whole serial link industrial robot using the DH parameters. The DH parameters for each link of the robot can be combined to a table. This table is called DH table. In *Figure 2.8* there is modeled Mitsubishi RV-3SD robot using the DH parameters. The parameter θ is variable in the every joint, because each joint is revolved. If the joint would be prismatic, θ would be a value and a would be the variable. [11, 19, 31, 38, 40]



	θ	d	a	α
1	variable	350	95	-π/2
2	variable	0	245	π
3	variable	0	135	π/2
4	variable	270	0	π/2
5	variable	0	0	- π/2
6	variable	85	0	0

Figure 2.8 *An example of robot DH-table [29]*

MATLAB can build a model of serial link utilizing Robotic Toolbox according to the DH parameters. We will consider more later about modeling the robots with MATLAB.

2.5 Robot redundancy

Obstacle avoidance can be implemented utilizing redundancy of the robot. If the robot collides to an environment, it is possible that other inverse kinematic solutions do not collide. The robot can be suitable to the task only if the robot does not collide to the environment.

Amount of the robot's joints can be compared to task dimensionality. If the robot has same amount of joints that it is required for the task's dimensionality, the system is called perfectly constrained. In overconstrained system the robot has lower dimensionality that is needed. If the robot dimensionality is greater what is required for the task, the system is called underconstrained. In this situation the robot is called redundant. [11, 16, 19]

Robot redundancy is always depending the task of the robot. The robot is simply redundant in certain task, if the robot's amount of DOF is greater than the requirement of the task. For example, 4-DOF SCARA robot is redundant for a task that requires only positioning tasks in xyz-coordinates, but non-redundant for a task that requires positioning with certain orientation along x and y axes. [11]

Even the robot's redundancy is task depending, 7-DOF robots are widely called redundant robots. This can be explained because maximum task DOF requirement is 6: coordinates x, y and z and orientations along x, y and z (α, β and γ). It can be concluded that if the robot's DOF is equal to or greater than 7, the robot is always redundant. [11]

The robot is redundant if it has more degrees of freedom than is required for executing the task. Redundancy of the robot increase the robot's ability to reach a target point with demand orientation in a restrictive environment. The robot redundancy increases the amount of solutions in the inverse kinematic problem. [11, 16, 19]

Robot redundancy and larger amount of solution to inverse kinematics can be utilized in multiple ways. The robot redundancy can be used to avoid singularities. The singularity is a tricky position of the robot. For example, in the singularity position, the robot may have infinite amount of inverse kinematic solutions that causes the robot spinning or another uncontrolled movement. The robot redundancy can be utilized also for avoiding joint limits. Obstacle avoidance also improves by the robot redundancy because of increased dexterity of the robot. Larger amount of the inverse kinematic solutions can be utilized by removing solutions that would result collision. [11, 16]

2.6 Rotation and transformation matrices

Before we are able to model the robot manipulators, rotation and transformation matrices must be understood. Transformation matrix is a matrix that is used to move and rotate frames. In the transformation matrix a rotation matrix and a translation vector are combined. [11, 19, 38, 40]

The rotation matrix is a 3x3 sized matrix. Let's assume that we have two coordinate systems {A} and {B}. The rotation matrix is

$$ {}^{A}_{B}R = \left[{}^{A}\hat{X}_{B} \quad {}^{A}\hat{Y}_{B} \quad {}^{A}\hat{Z}_{B} \right], \tag{2.1} $$

where ${}^{A}\hat{X}_{B}$, ${}^{A}\hat{Y}_{B}$ and ${}^{A}\hat{Z}_{B}$ are unit vectors defining a relation between coordinate systems of frames {A} and {B}. We get the transformation matrix by adding a translation vector Q and the rotation matrix ${}^{A}_{B}R$ together in a following way:

$$ {}^{A}_{B}T = \begin{bmatrix} {}^{A}_{B}R & Q^{\mathrm{T}} \\ 0\ 0\ 0 & 1 \end{bmatrix}. \tag{2.2} $$

We can now display a clarification of the transformation matrix:

$$
{}^{A}_{B}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & q_x \\ r_{21} & r_{22} & r_{23} & q_y \\ r_{31} & r_{32} & r_{33} & q_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.3}
$$

where q are coordinates and r the rotation components. [11, 19, 21, 38, 40]

We can now make the rotation matrices for rotation along each axis. In these following rotation matrices only the variable angle changes. Changing the angle, we can rotate a frame along any axis x, y or z using the pre-defined matrices. The pre-defined rotation matrices are

$$
{}^{A}_{B}R_z = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.4}
$$

$$
{}^{A}_{B}R_y = \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 0 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}, \tag{2.5}
$$

$$
{}^{A}_{B}R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}. \tag{2.6}
$$

By changing variables α, β and γ, point can be rotated corresponding amount of degrees in coordinate system. Cosine and sine are shortened to c and s. Now we can combine the rotation matrices to following formula:

$$
{}^{A}_{B}R_{zyx} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta s\gamma \end{bmatrix}. \tag{2.7}
$$

Almost every transformation in this thesis is based in this rotation matrix. This form of the rotation matrix applies in almost every situation since it includes rotation along all axes x, y and z. [11, 19, 21, 38, 40]

2.7 Forward kinematics

In order to know if a serial manipulator is suitable to an environment, it is essential to know if the robot collides to the environment. Before we can detect the collisions, a position of the robot must be known. Location and orientation of each link and joint must be known in order to check if it collides to the environment. Forward kinematics solves this problem.

In forward kinematics the robot is in a pose. The problem is to know the coordinates of an end-effector of the robot. The values of each joint is known. For revolved joints these variables are θ and for a prismatic joint a. However, in forward kinematics all the θ and a are known. The solution to forward kinematics problem is the transformation matrices. [11, 19]

When solving the forward kinematics problem, we need a general transformation matrix for any link. Using the DH parameters, we only have the transformation along X and Z axes. This includes the rotation R_X along X axis, the translation D_X along X axis, the rotation R_Z along Z axis and the translation D_Z along Z axis multiplied together. So we get

$$ {}^{i-1}_{i}T = R_X(\alpha_{i-1})D_X(a_{i-1})R_Z(\theta_i)D_Z(d_i). \tag{2.8}$$

Now we can calculate the general link transformation matrix utilizing formulas 2.2, 2.4 and 2.6:

$$ {}^{i-1}_{i}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

After double-checked the result with MATLAB, we get the transformation matrix for any link:

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.10}$$

If the calculation had been made with the Z transformation first and then X, we would have got different matrix. The alternative transformation matrix is called Denavit-Hartenberg matrix:

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_{i-1} & s\theta_i s\alpha_{i-1} & c\theta_i a \\ s\theta_i & c\theta_i c\alpha_{i-1} & -c\theta_i sc\alpha_{i-1} & s\theta_i a \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.11}$$

Utilizing either one of these general solutions of the link transformations, we are able to calculate the transformation of the serial link manipulator multiplying the link transformations:

$$ {}^{0}_{N}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T \ldots {}^{N-1}_{N}T. \tag{2.12}$$

This transformation matrix informs us an orientation and a location of the end-effector of our robot. Forward kinematic problem is now solved. [11, 19, 38, 40]

2.8 Inverse kinematics

With inverse kinematics, the problem is opposite compared to forward kinematics. In the inverse kinematic problem, we know end-effector location and orientation, but joint values of the robot are unknown. Inverse kinematics finds a solution with which joint values the robot's end-effector is located in the predetermined location and orientation. In the inverse kinematic problem, it is possible that no solution exists, one solution exists or multiple solution exist. *Figure 2.9* shows a simple example of the inverse kinematic problem with two link manipulator with two solutions. The complexity of the inverse kinematic problem is exponential in the robot DOF and polynomial in the number of obstacles. In the real world, the robots usually have at least six DOF and at least hundreds of points to avoid. This makes the problem challenging. [11, 19, 28, 40]

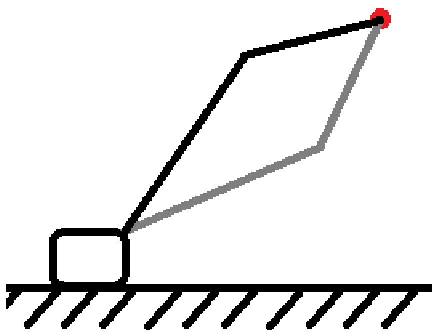


Figure 2.9 *Inverse kinematics example*

There are many different approaches for the inverse kinematics. The inverse kinematic problem can be solved for example with analytic or numerical approach. Different ways to solve the inverse kinematic problem and new methods are invented all the time. The most important and the most established methods are review in the next chapter. [11, 19, 38]

2.9 Solving the inverse kinematic problem

Methods for the inverse kinematic problem finds a solution to nonlinear sets of equations. In the inverse kinematic problem, it is possible that there is large amount of solutions or no solutions exists. [11]

The inverse kinematic problem can be solved with many different ways. The inverse kinematics solving methods can be sort into categories. Most of the methods can be categorized to closed-form solutions or numerical solutions. [11]

2.9.1 Closed-form solutions

The closed-form solutions are faster than numerical solvers and easily able to find all the solutions. On the other hand, the closed-form solutions are depending on the robot's structure and must be generated for each robot separately. [11]

Generally, the closed form solutions can be used only for 6 DOF robots with certain structure. However, large amount of the industrial robot has this specified type of structure. So called ad hoc techniques are the most efficient closed-form methods for finding the solutions to inverse kinematic problem utilizing geometric features of particular mechanisms. The closed-form solutions can be divided to algebraic and geometric methods. [11]

With the algebraic method, equations with the joint variables are created. The equations will be modified into a soluble form to get the joint values. In the algebraic methods few different strategies are used. An equation pair is often a suitable solution instead of multiple transcendental equations. [11]

The geometric methods are based on idea, that includes reducing set of joint variables into functions. This method splits the problem into parts. The separated problems are then solved with the algebraic methods. For example, some of the robots with articulated structure allows solving the inverse kinematics with the geometric methods splitting the problem to inverse position kinematics and inverse orientation kinematics. Both problems are then solved separately. [11]

2.9.2 Numerical methods

Numerical solutions are not depending on the robot's kinematic structure. This means that the numerical methods are not robot depending and they can be used for any robot. On the other hand, the numerical methods are slower than the closed-form solutions. However, this thesis will focus to the numerical inverse kinematic methods since the robot structures can vary significantly. The numerical inverse kinematic solving methods can be divided into three different categories. These categories are called symbolic elimination, continuation and iterative methods. [11]

The symbolic elimination method utilizes analytical manipulation to reduce amount of equations by eliminating variables. With using this method, it is possible to find all the possible solutions. Multiple improvements have been made to this method. Manocha, Canny, Roth and Raghavan are known for improving the symbolic elimination method. [11]

Continuation methods uses a tracking principle. A solution path is tracked from a start system with known solution. A target system's solutions are solved transforming the start system. This method can be used to solve all the possible solutions. [11]

Most of the iterative methods results to a single solution. This solution is based to an input called initial guess. The initial guess is a guess of joint values that are estimated to be near to the solution. A solution time is greatly depending on quality of the initial guess. Most well-known iterative methods are Newton-Raphson, Pieper, modified predictor-corrector, interval analysis, damped least-square and optimization approaches. Especially the interval analysis is promising because it can find all the possible solutions and it uses fast convergence to the solution. [11]

2.9.3 Jacobian matrices

Utilizing a Jacobian matrix is perhaps the most used technique for inverse kinematics. If the forward kinematic problem is stated

$$\boldsymbol{x} = f(\boldsymbol{q}), \tag{2.13}$$

the Jacobian matrix can be defined as

$$J = \frac{\partial \mathrm{f}}{\partial \mathbf{q}}. \tag{2.14}$$

Then the inverse kinematic problem can be solved with formula

$$\dot{\boldsymbol{x}} = J\dot{\boldsymbol{q}}, \tag{2.15}$$

where $\dot{\boldsymbol{x}} = \frac{\boldsymbol{dx}}{\boldsymbol{dt}}$, $\boldsymbol{x}$ is an end-effector vector, $\dot{\boldsymbol{q}} = \frac{dq}{dt}$ and $\boldsymbol{q}$ is the joint variable vector. [11, 19]

Inverse kinematic can be solved with several different ways utilizing the Jacobian matrix: Pseudoinverse of Jacobian that gives least-squares solution with redundant system and matrix transpose for resolving the closed-loop inverse kinematic. [20]

2.9.4 Other methods

Also other methods are developed. These methods cannot be classified to any previous approaches. A few other methods are reviewed in this chapter.

Lagrangian methods can be used to solve the inverse kinematic problem. This method utilizes an objective function. Lagrangian multipliers can be used to extend an underconstrained system to a perfectly constrained system. [20]

Reach hierarchy is a method by James Korein and Norman Balder that tries to solve inverse kinematics with utilizing workspaces of each joint. The method tries to find an intersection between a goal trajectory and a workspace boundary. This method is difficult to use since the workspaces of the robots can be complex and the method requires a lot of precomputation. [20]

Triangulation is a method for solving the inverse kinematic problem by rotating joints one by one. The triangulation is an improved method based to Cyclic-Coordinate Descent (CCD). The joint movement is implemented different way compared to the CCD. The rotation continues until an angle θ between lines p_c to p_t and p_c to p_e is zero. Principle of the triangulation is shown in *Figure 2.10*. [33]

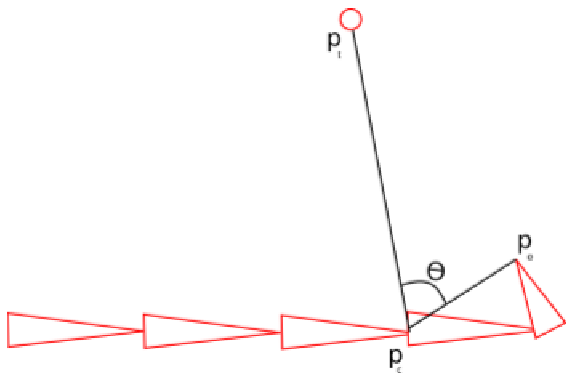


Figure 2.10 *Triangulation principle [33]*

The joints are rotated with the order of importance. The algorithm avoids the joint rotation as much as possible utilizing cost function to find a simple solution. This method guarantees to find the solution if one exists. [33]

2.10 Joint limit avoidance methods

The inverse kinematic problem must be solved within joint limits of a robot. Otherwise the robot is not suitable for the task. All inverse kinematic solvers do not include the joint limits, so this may be implemented with other methods.

Multiple approaches can be used for robot joint limit avoidance. First of the methods is to ignore the joint limits. After solving the inverse kinematic problem, all results that violates the joint limits are discarded. Then the rest of the results are within the joint limits.

While making the obstacle avoidance, the same functions can be used also for the joint limit avoidance. For example, functions like cost-, objective- and task priority -functions can be used to implement the joint limit avoidance.

2.11 Collision detection methods

The robot can be suitable for the task only if the robot does not collide with the environment. Therefore, it is essential to avoid collisions. Before we can avoid the collisions, we should be able to detect collisions. Collision detection methods can be divided into four categories: space-time volume intersection, swept volume interface, multiple interface detection and trajectory parametrization. The most suitable collision detection methods for this thesis are reviewed in this chapter. [30]

Collision detection consumes a lot of CPU resources. The amount of collision tests is large and after each movement of any object, the collisions must be checked. For these reasons it is important to make the collision detection as efficient as possible. [24]

2.11.1 Distance calculation

In distance calculation method, a distance between (in this case) the robot and other objects is measured. If the distance is smaller than a given collision distance, collision occurs.

This method has few problems: It is required to define closest points of the robot and the environment. Then the distance between the robot and the object is the distance between these points. It may require a lot of computing if a distance between all points of the robot model is measured to all points of the object model. All this calculation has to be made after every movement. However, the idea itself if simple and it does not require to generate other collision model shapes.

2.11.2 Bounding volumes

Bounding shapes is a method that makes the collision detection more efficient. This method uses rectangles or spheres to surround objects (*Figure 2.11*). Then an intersection between these shapes are checked. If there are no intersections between the bounding shapes, then there are no collisions. Otherwise, there may be collisions and further tests are required. [5, 30, 41]

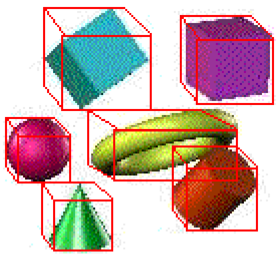


Figure 2.11 *Bounding shapes [15]*

Multiple shapes can also be used for one object. *Figure 2.12* shows a robot manipulator with sphere subdivision. A set of spheres forms the robot collision model. Collisions are not checked between these spheres. Instead, the collisions are checked between these spheres to any spheres constructing an obstacle collision model. [5, 30, 41]

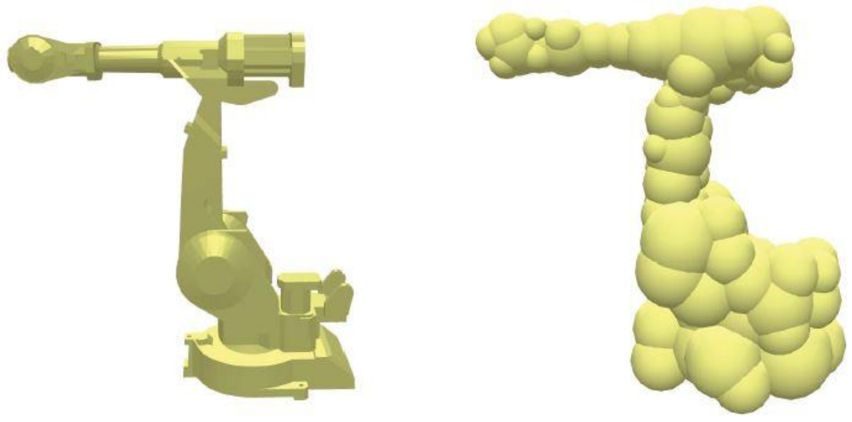


Figure 2.12 Sphere subdivision [5]

The rectangle or sphere intersection test is simple and therefore efficient to use. This makes it possible to check the collisions after every step of movement.

2.11.3 Triangle intersection

Triangle intersection is a method for checking triangle mesh collisions. 3D objects are mainly generated from triangle-shaped polygons. The Collisions can be checked by checking an intersection of these triangles. *Figure 2.13* shows an example of the triangle intersection check.

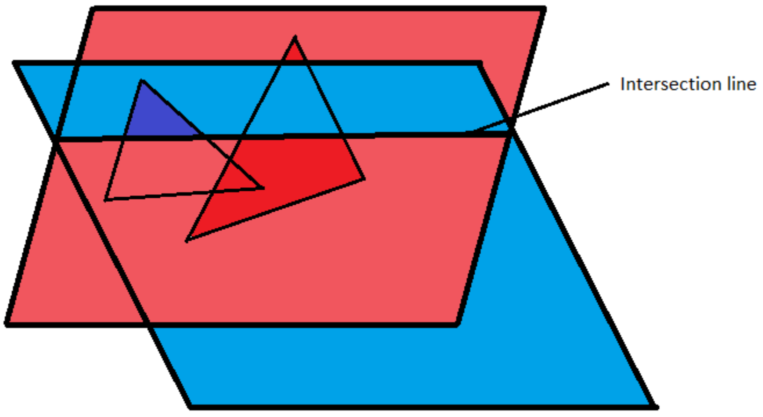


Figure 2.13 Triangle intersection

An intersection of the triangles can be checked by making planes for both triangles. The planes are defined by vertices of the triangles. The collision occurs only if the triangles overlaps each other along an intersection line of the planes. Otherwise, the collision does not occur. As it can be seen from *Figure 2.13* the collision does not occur, since the triangles do not overlap along the intersection line of the planes. [24, 45]

This method may be too slow with large amount of polygons. Kun Qian, Xiaosong Yang and Jianjun Zhang compared this method to the bounding sphere method in 2015. This method was slower than the bounding sphere method. Other drawback is also the fact that this method only works with the polygons. This requires also a 3D model of the robot to test collision. [24]

2.11.4 Spatial partitioning

Spatial portioning is a method for making the collision detection more efficient. This method includes dividing a space into cells. [30]

The method counts a number of objects in each cell after every step of movement. If one cell contains more than one object, more tests are required with other methods. Otherwise, no collision occurs. [30]

2.12 Obstacle avoidance methods

Multiple methods can be used for the robot manipulator's obstacle avoidance. Most of the reviewed method are optimization functions that can be implemented to the collision avoidance. Some of the methods serves more than one tasks simultaneously. The first task is always reaching towards the goal and a secondary task is avoiding the obstacles.

2.12.1 Cost function

Cost function is an optimization function. The cost function increases a variable after system changes. Optimal system status is at local minimum of the cost function. In inverse kinematic -case the cost function can be used for example directly as distance from end-effector to the goal. As the robot joint values change, the lowest cost is the joint value that results the end-effector closest to the goal. [7]

In the obstacle avoidance, the cost function can be utilized with multiple ways. The cost function can be utilized with an inverse relation of distance between the robot and the obstacle. Other method is for example, if the robot violates the collision distance, the cost function can be set to increase. Otherwise the cost function can be set to 0. In this case, the robot avoids to move to the collision distance, since being outside the collision distance results to the lowest cost function. The joint limit avoidance is simple to add to the cost function method: If the robot violates the joint limits, the cost function increases significantly. [7]

2.12.2 Task priority

Task priority is a technique that can be used to the collision avoidance. Aurel Fratu, Jean-Francois Brethe and Mariana Fratu implemented the collision avoidance with this method in 2010. This method serves two tasks. The task priority can be implemented to the collision avoidance for example in the following way: The first step is to determine the closest point of the robot manipulator to the obstacle (critical point). Second step is to calculate the distance from the obstacle to the critical point. [6, 8, 9]

The primary task includes only the end-effector's velocity towards the goal and secondary task is the critical point's velocity away from the obstacle. This method cannot be used in this thesis since this is point-to-point problem and therefore the velocities are not considered. [6, 8, 9]

2.12.3 Objective function

Objective function can be used in the obstacle avoidance with several ways. The function has primary and secondary objectives. The primary objective is obviously reaching the goal. [13]

Secondary objective can vary depending on a situation. For example, the secondary objective can be maximizing the distance between the obstacle and the manipulator or maximizing the area between the manipulator links and the obstacle.

2.12.4 The kinematic roadmap

The kinematic roadmap is a method by Juan Ahuactzin and Kamal Gupta in 1999-2000 for inverse kinematics and collision avoidance. The method is based on Ariadne's Clew algorithm. [2, 3]

The method is an inverse kinematic solver that avoids collisions. The algorithm consists of two sub-algorithms called Explore and Search. Explore is an algorithm that explores the robot's free configuration space and places landmarks in it. Search-algorithm finds the robot joint movement limits and moves each joint to a position where the end-effector is closest to the goal frame. With this method the robot can be moved without checking the collisions, since collision avoidance is taken into account in the limit search phase. This saves a lot of CPU recourses. The cost function is used in the search algorithm. [2, 3]

2.12.5 Potential field

Potential field method is widely used in the robot's obstacle avoidance. Dae-Huyng Park, Heiko Hoffmann, Peter Pastor and Stefan Schaal implemented collision avoidance with the potential fields in 2008. The potential fields are also utilized in the collision avoidance by Cornel Secara and Luigi Vladareanu. With this technique, a predefined potential field is determined around obstacles. While the robot touches to the field, it causes repellent force to the robot link. [13, 14]

This method is also based to the robot velocities. Therefore, it is not suitable for in this thesis.

3. APPROACH

Before starting an implementation, it is essential to define working environment, methods to use and design the implementation. In this case, the working environment was already selected. All the methods for solving the main problem are selected in this chapter.

In this thesis there were several choices to do before starting the implementation. 3D format had to be chosen for modeling the environment. Tools of the thesis had to be defined and distribution channel of tools had to be chosen. In this chapter the approach of the implementation is rationalized.

3.1 Selected method for inverse kinematics

When selecting the method for the inverse kinematic problem, one method was more suitable in this thesis than any other. The selected method solves point-to-point inverse kinematic problem and avoids collisions and robot joint limits. However, many changes must be done to make the solution work with any robot and any environment.

The main problems of this thesis is solved with a similar algorithm with Ariadne's Clew algorithm. The method solves the inverse kinematic problem within joint limits and with collision avoidance. The selected method is fast when compared to the other methods, so it is suitable for executing multiple times for different robots in complex environment to find suitable industrial robots. This method does not require an initial guess since the robot will start from random orientation.

The selected method to solve inverse kinematics is based the Ariadne's Clew algorithm. The algorithm is a path planning algorithm that can be used to solve the inverse kinematic problem. The algorithm was selected as inspirited by Triangulation method by R. Muller-Cajar and R. Mukundan. The Ariadne's Clew algorithm is similar to RRT-algorithms (Rapidly-exploring Random Tree): It rapidly explores the configuration space. [22, 27, 28, 33]

The Ariadne's Clew algorithm consists of two sub-algorithms: Search and explore. Search is an algorithm that checks with an iterative method if the goal is reachable from any known position. Search is similar to the triangulation method. The explore algorithm explores the configuration space with increasing resolution and adds landmarks for search algorithm. [27, 28, 33]

In 1999 Juan Ahuactzin and Kamal Gupta made successful implementation of Ariadne's Clew algorithm to solve the point-to-point inverse kinematic problem. This thesis utilizes that implementation and extends it to general robot- and environment solutions. [2, 3]

3.1.1 Explore

Explore is one of the two sub-algorithms. Explore algorithm explores the configuration space and places landmarks in it. Each time explore is executed, it generates a number of embryos. Originally in the Ariadne’s Clew algorithm the embryos were generated systematically as far from each other as possible. However, to speed up the algorithm, Juan Ahuactzin and Kamal Gupta used quasi-random method to create the embryos. The embryo furthest from the closest landmark is selected as the new landmark. The embryos are created by making random robot joint movements from each previous landmark. *Figure 3.1* clarifies the landmark creation method. The figure shows an example of two-dimension configuration space with 10 previous landmarks. In the figure black X corresponds the previous landmarks, green O are the new embryos and red X is the embryo that is selected to the new landmark since it is the furthest embryo away from the other landmarks. [2, 3]

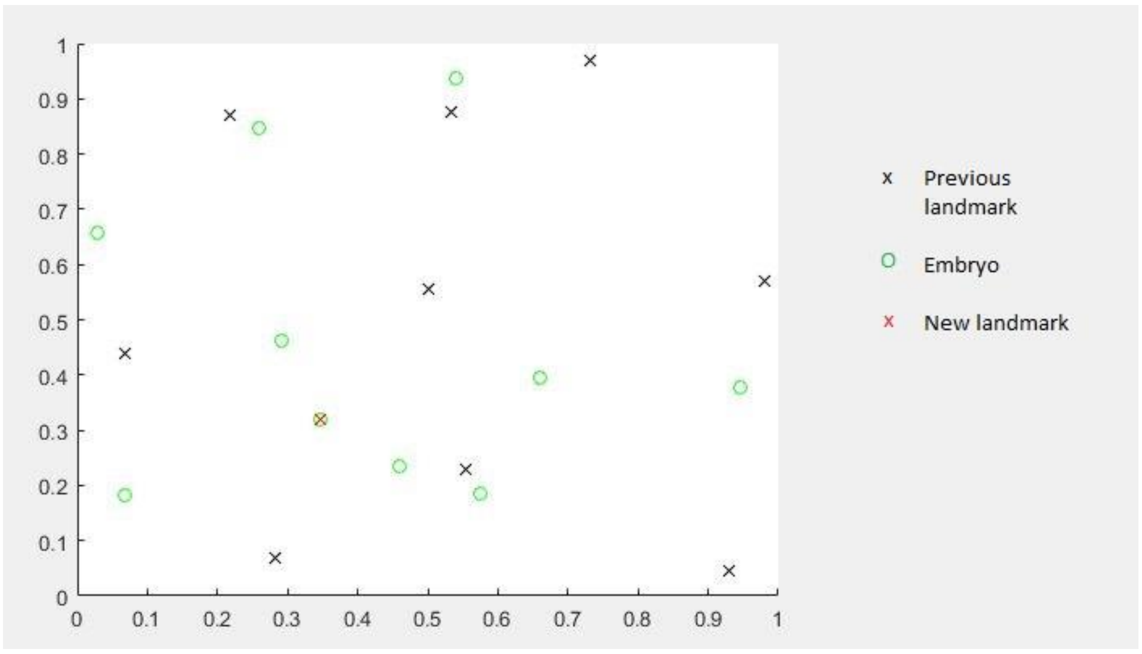


Figure 3.1 *Landmark creation clarification*

Figure 3.2 Shows the same example with 100 landmarks. This method spreads the landmarks evenly even if it is based randomly placed embryos. The landmarks are marks of explored free configuration space. The embryos cannot be placed if it leads the robot to collision. This method rapidly explores the free configuration space of the robot.

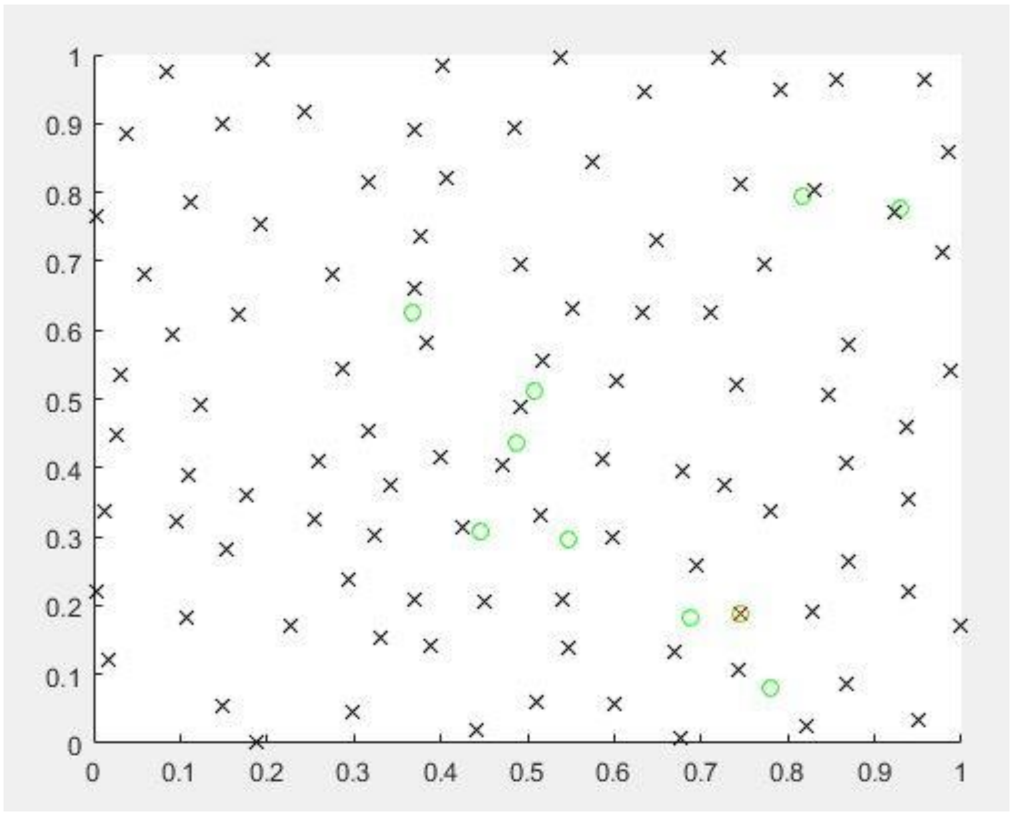


***Figure 3.2** Explore with 100 landmarks*

This method tries to approach the goal from as different directions as possible. As we can see from *Figure 3.2*, as number of the landmarks increases, the resolution of the explored configuration space increases. This is suitable approach for the research problem, since the environment of the robot is always unknown. The method explores the environment during each iteration by placing the landmarks. The structure of the robot does not matter with explore. Even prismatic joints can easily be implemented. However, some modifications have to be made to make the algorithm work with different number of joints. The algorithm must detect the joint type and number of joints before randomizing the joint values. Also the fact that an initial guess is always random instead of constant is suitable for a variable environment.

3.1.2 Search

Search sub-algorithm is executed each time after explore. Search tries to move the robot to the goal from the newly placed landmark. All the landmarks are searched only once. Then the landmark is left to the configuration space to represent an explored place. [2, 3]

Search consists of two phases. The first phase is a joint limit search. The limit search moves the robot joint until collision occurs or the joint limit is exceeded. When this occurs, the direction of the movement is reversed and the joint value is saved. When the other limit is found with same manner, the joint limits Δ_l^{min} and Δ_l^{max} are then found. Other joint values stay fixed while the robot is searching the limits. This phase requires largest amount of computation in the whole algorithm, since forward kinematics and collision detection must be calculated after every step of movement. *Figure 3.3* shows a clarification of the joint limit search phase. [2, 3]

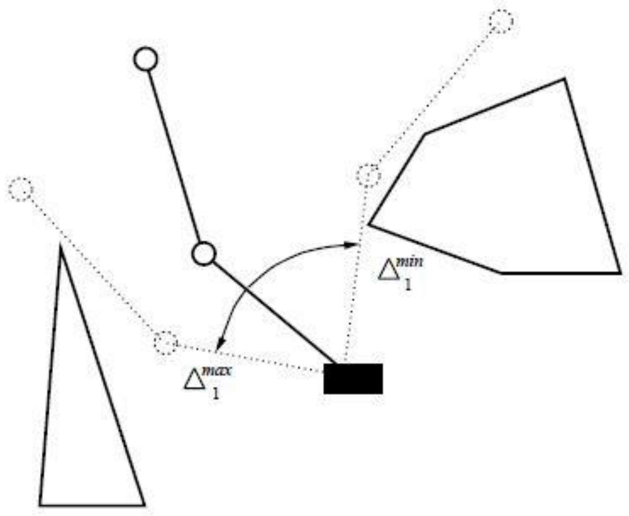


Figure 3.3 *Joint limit search of search sub-algorithm [2]*

Now the algorithm knows the rotation interval for the joint, where the joint is safely able to move without collisions. After the joint limits are found, the second phase of search-algorithm is executed. This phase searches the local minimum of the cost-function from Δ_l^{min} to Δ_l^{max}. The cost-function is the sum of three Euclidean-distances.

$$cost = \sqrt{d_x^2 + d_y^2 + d_z^2}\,, \tag{3.1}$$

where d_x is the distance from endpoint of the robot end-effector's frame F_a unit vector i to an endpoint of the goal frame's F_b unit vector i, d_y is the distance from an endpoint of the robot end-effector's frame F_a unit vector j to an endpoint of the goal frame's F_b unit vector j and d_z is the distance from the endpoint of the robot end-effector's frame F_a unit vector k to the endpoint of the goal frame's F_b unit vector k. This is clarified in *Figure 3.4*. Adding the sum of distances of the unit vectors instead of single point provides the robot to move desired orientation and location instead of location only. [2, 3]

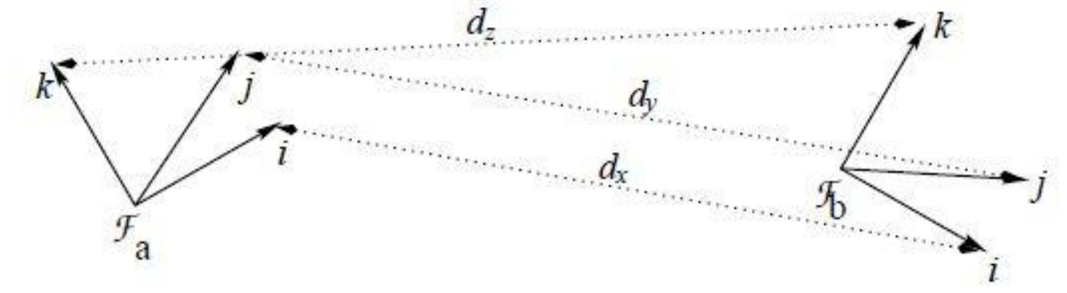


Figure 3.4 *Cost-function clarification [2]*

After the local minimum of the cost function is found between Δ_l^{min} and Δ_l^{max}, the robot sets the joint value to the value where the local minimum is found and proceeds to the next joint executing the same two phases. After all the joints are passed the cycle, search function is ended. It must be noticed that with the general solution, the robots may have any number of joints. The new cycle starts until the goal is reached, an iteration limit is exceeded or the change in the final cost function drops below a given limit (the progress of moving closer to goal is ended). [2, 3]

The inverse kinematic function repeats explore and search functions until a solution is found or the iteration limit is exceeded. When the solution is found, the function returns joint values leading to the goal. [2, 3]

In this thesis, there may exist any number of goal frames. The robot must reach all the goal frames in order to be suitable robot. This must be noted when implementing the solution.

3.2 Selected method for collision detection

The selected method for collision detection is the sphere intersection (bounding spheres 2.11.2). It can also be classified to distance calculation method (2.11.1). The idea of this method is taken from research “Optimization of robot links motion in inverse kinematics solution considering collision avoidance and joint limits” by S. Mitsi, K.-D. Bouzakis, G. Mansour in 1994. This method is simple enough to be repeated with short interval, like after every degree of movement of the robot, and with complex obstacles. Sphere intersection is the method that models the robot and obstacles with spheres. If the sphere from the robot model intersects with the sphere from the obstacle model, collision occurs.

If r_i is a radius of the sphere from the robot model and r_j is a radius of the obstacle model sphere, collision occurs if a distance between the center of those spheres is $r_i + r_j$ or smaller. The spheres can be set to equal size with each other to make the collision detection calculation more simple and therefore faster. Any shape can be modeled with spheres when adjusting amount of the spheres and size of the spheres, since MATLAB is powerful calculating large matrices. It is reasonable to model collision detection with identical spheres instead of set of different sized spheres. In this case $r_i = r_j$. Therefore, the collision occurs when the distance between the center of the spheres is equal or smaller than $2 * r_i$. This distance is called collision distance. An example of the collision models is shown in *Figure 3.5*.

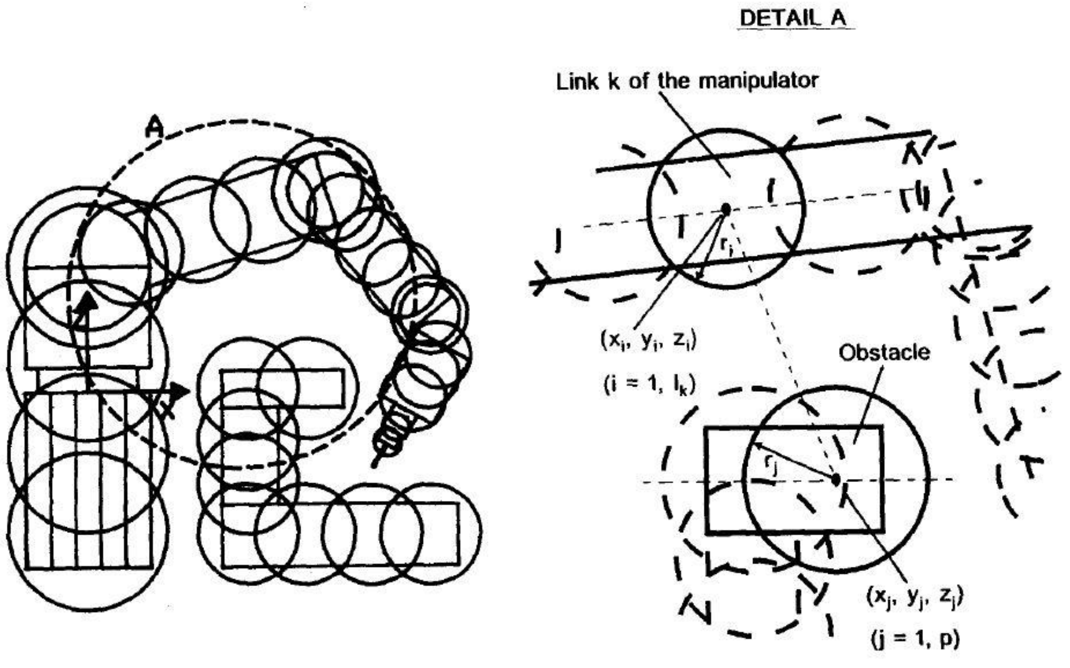


Figure 3.5 *Collision detection principle [36]*

The robot and the obstacles can be modeled with sufficient accuracy using the identical collision spheres. However, density of the spheres must be adjustable. The collision distance must also be adjustable since it directly corresponds size of the spheres.

The challenges with this method are with the placement of the spheres. The models must be filled with the spheres regularly and the solution must be general and work with any model. Other challenge is to optimize the amount of the spheres to avoid unnecessary calculation.

3.3 Selected method for obstacle avoidance

The approach of obstacle avoidance is included in the search sub-algorithm. At the first phase of the search function includes a limit searching. This phase includes collision avoidance component. After each degree of movement, the collision will be checked between the robot and the obstacle. If the collision occurs, the joint movement limit is set. Otherwise the movement is continued until the joint limit exceeds or the collision occurs. [2, 3]

Since the robot limits are searched before every movement, the collisions do not have to be taken into account later in any other phase. This method is fast enough to be calculated with every movement in the complex environment with large amount of collision model spheres.

3.4 Challenges of the main algorithm

Main challenges of the selected approach might be making the solution general. The solution must work with any serial manipulator robot model and with any environment model. The solution should work with the different sized robots and with prismatic and revolved joints.

Other challenges are with processing time of the algorithm. The goal is to make as efficient and as fast algorithms as possible and avoid unnecessary calculation. With narrow environments it may require large amount of landmarks so it is essential to have fast algorithms. It is interesting to find out how the robots will find the way through hole-shaped obstacles with this approach.

Challenges may also occur with placing the bounding spheres regularly. Bugs and mistakes in source code will also cause minor challenges in the beginning of implementation.

3.5 Selecting working environment

When starting this project as a special assignment, it was allowed to select any programming environment comfortable for a student. My clear choice was MATLAB, which was known from previous assignments.

MATLAB is a strong fourth generation matrix-based programming language especially handling matrices. Matrix handling is useful feature in robotics. MATLAB is also good at numerical computing, that is needed especially when solving the inverse kinematic problem. MATLAB is mostly used for math, graphics, programming and simulating. [26]

With MATLAB graphical user interfaces (GUI) can be done easily. GUI is a window that can include for example push buttons, sliders, checkboxes, plots or editable text fields. GUI is the interface that the user manipulates and where the results are displayed.

MATLAB also has code converters to many languages. Finished MATLAB code can be turned to java, C++ or even an application. After finishing a programming phase, a built application can be distributed to the users and no MATLAB are then needed for executing the tool except MATLAB runtime. [26]

3.6 Robotic Toolbox

Robotic Toolbox is a robot related open source function pack for MATLAB. Robotic toolbox provides many functions for robotics. For example, kinematics-, dynamics- and trajectory –related functions are provided. Robotic toolbox is especially for serial link manipulators. [31]

With Robotic Toolbox, it can be easily defining a serial manipulator using the DH parameters. After definition of the robot, it can be plotted and moved.

3.7 Modeling environment in MATLAB

3D objects can be modeled with several different ways in MATLAB. All the different methods are made for different situations. In this chase, the main reason for selecting one above others is the fact that it works best with 3D file format chosen.

First way to model the environment is a mesh-command. Mesh draws a wireframe mesh based from the given data. This command is not used in this thesis. [25]

Command for drawing surfaces is called surf. Surface is a three-dimensional filled mesh. Different variants of surf commands exist. Trisurf-command creates surface from triangles. This command is useful when creating a single pre-defined object that consists of large amount polygons. [25]

For drawing single polygon, there is a fill3-command. The command creates a three-dimensional polygon and fills it with defined color. Fill3-command is used in this thesis to create single polygons. This can be utilized for example when removing the single polygons without clearing whole graph. Especially the environment modelling tool utilizes fill3 drawing the polygons one by one. [25]

In MATLAB, there are also pre-defined shape commands. For example, a sphere-command creates a sphere with defined size. The created sphere can be plotted for example with surf-command. Different shapes have their own commands. [25]

MATLAB is a versatile programming language, so there are many other ways to create 3D objects. However, these reviewed methods are probably the most common ones.

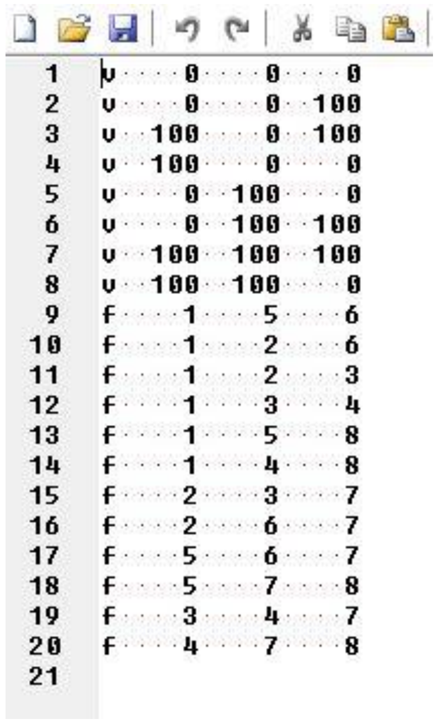
3.7.1 Chosen 3D format

There are many types of 3D-files. In this project, it was required to import models to MATLAB, so it is essential to be able to easily draw the shapes of the file using MATLAB functions. Because there are many ways to draw polygons in MATLAB, it was reasonable to select polygon mesh -based format.

The polygons are triangles that are defined by three corner points. There are many quite similar polygon mesh formats, because the idea behind the polygon definition is simple. Most simple polygon mesh format was found is called Raw mesh (.raw). In the raw-format each line consists of three vertices defining the triangle (X1 Y1 Z1 X2 Y2 Z2 X3 Y3 Z3). However, raw-format is not so widely used. In this project a user should be able to model the environment with any 3D-modeling tool, so more common format is required.

Wavefront OBJ –format is the polygon mesh format developed by Wavefront Technologies. OBJ-files are in common use and almost every 3D-modeling software can save polygon meshes in obj-files. OBJ consists of two parts. The first part of the file is the vertices part. Lines defining the vertices starts with character v. After v, there are X, Y and Z coordinates of the point separated with whitespaces. Each vertex lines defines a corner point of the polygon. [43]

After the vertex part, there is a polygon face element. The face element defines the polygon by connecting three corner points to each other. A line in the face element starts with character f. Then the line includes three numbers separated by whitespaces. Numbers in the face element lines are corresponding to the line numbers of the vertices being connected. In *Figure 3.6* an example of OBJ-file is shown in Notepad2. The represented object is a cube with side length of 100 units. A unit type is not defined in the obj-format. [43]



```
 1  v    0    0    0
 2  v    0    0  100
 3  v  100    0  100
 4  v  100    0    0
 5  v    0  100    0
 6  v    0  100  100
 7  v  100  100  100
 8  v  100  100    0
 9  f    1    5    6
10  f    1    2    6
11  f    1    2    3
12  f    1    3    4
13  f    1    5    8
14  f    1    4    8
15  f    2    3    7
16  f    2    6    7
17  f    5    6    7
18  f    5    7    8
19  f    3    4    7
20  f    4    7    8
21
```

Figure 3.6 *Cube in obj-format viewed in Notepad2*

Obj-format includes many optional features. Lines starting with # are comment lines and therefore are skipped. The format also supports textures, vertex normal and parameter space vertices. [43]

3.7.2 Modeling obj-files in MATLAB

Modeling obj-files in MATLAB can be done with several steps. First the file must be opened and read. Then vertices and faces must be saved to separate matrices. Then a shape can be drawn.

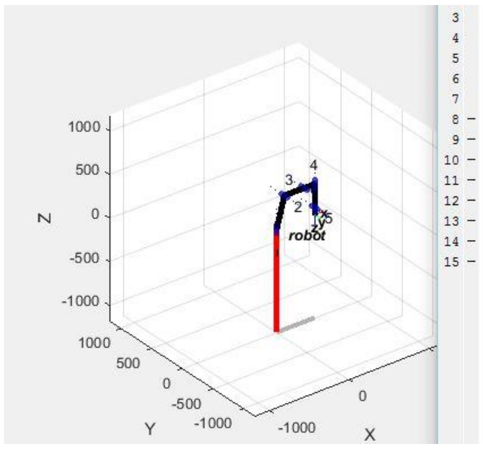
The first phase is to get data when drawing the obj-file in MATLAB. The data can be imported from GUI of the obj-file. For example, if a user wants to draw a cube in MATLAB from the obj-file. The data must be read from the file.

The vertices are read first. As long as lines starts with character v, numbers from the line can be saved to a vertices matrix. The vertices matrix can be defined to be a $3 * n$ size matrix, where X, Y and Z coordinates have own columns. Each line defines one vertices point. After reading the vertices, face lines are read as long as the lines starts with character f. The faces can be saved to own matrix similar to the vertices.

When drawing the shape, fill3 or trisurf –commands can be used. For example, when using fill3, the function requires matrices for X, Y and Z and color of the polygon. X, Y and Z are matrices with 3 rows and n columns. The rows are coordinates to the corresponding axis of the different polygons. The columns define different corners of that polygon in the corresponding axis. For example, the first column of matrix X consists all X coordinates of the first polygon. The second column includes X coordinates of the second column. The matrices X, Y and Z can be generated from the face matrix taking the corresponding coordinates from the vertices matrix and saving them to the X, Y and Z matrices. Then the shape can be drawn with the command *fill3(X, Y, Z, C),* where C is the color of the polygon.

3.8 Modeling robots in MATLAB

Robots can be modeled in MATLAB utilizing the Robotic Toolbox. Modeling the robot is based to the robot DH parameters. In Robotic toolbox, each link is defined separately. In the link definition, a link name and the DH parameters of the link are given. Additional parameter is Σ indicating a joint type. Value 0 equals a revolved joint and value 1 a prismatic joint. The links are then connected to each other with SerialLink-function. Optionally a robot base can be defined. *Figure 3.7* shows Mitsubishi RV-3SD robot with code of the model. [31]



```
Link1 = Link([0 350 95 -pi/2 0]);
Link2 = Link([0 0 245 pi 0]);
Link3 = Link([0 0 135 pi/2 0]);
Link4 = Link([0 270 0 pi/2 0]);
Link5 = Link([0 0 0 -pi/2 0]);
Link6 = Link([0 85 0 0 0]);
robot = SerialLink([Link1 Link2 Link3 Link4 Link5 Link6]);
robot.plot([0 0 0 0 0 0])
```

Figure 3.7 *Robot model with model code*

Serial link manipulators can be modeled in this way. Parallel robot structures can be made using multiple serial links and connecting them together.

3.9 Designing the tools

When designing the tool of the thesis, it can be noted that the tool can be divided into three parts. There are three main uses for the tool. The first part of the tool is for modeling an environment of the robots. Second part is for modeling the robots. Third use of the tool is to test the modeled robots in the modeled custom environment. *Figure 3.8* shows the UML (Unified Modeling Language) use case diagram of the tools. Tool this versatile may easily become too confusing and complicated for the user. Therefore, it is reasonable to make three separate tools for keeping the tools user friendly.

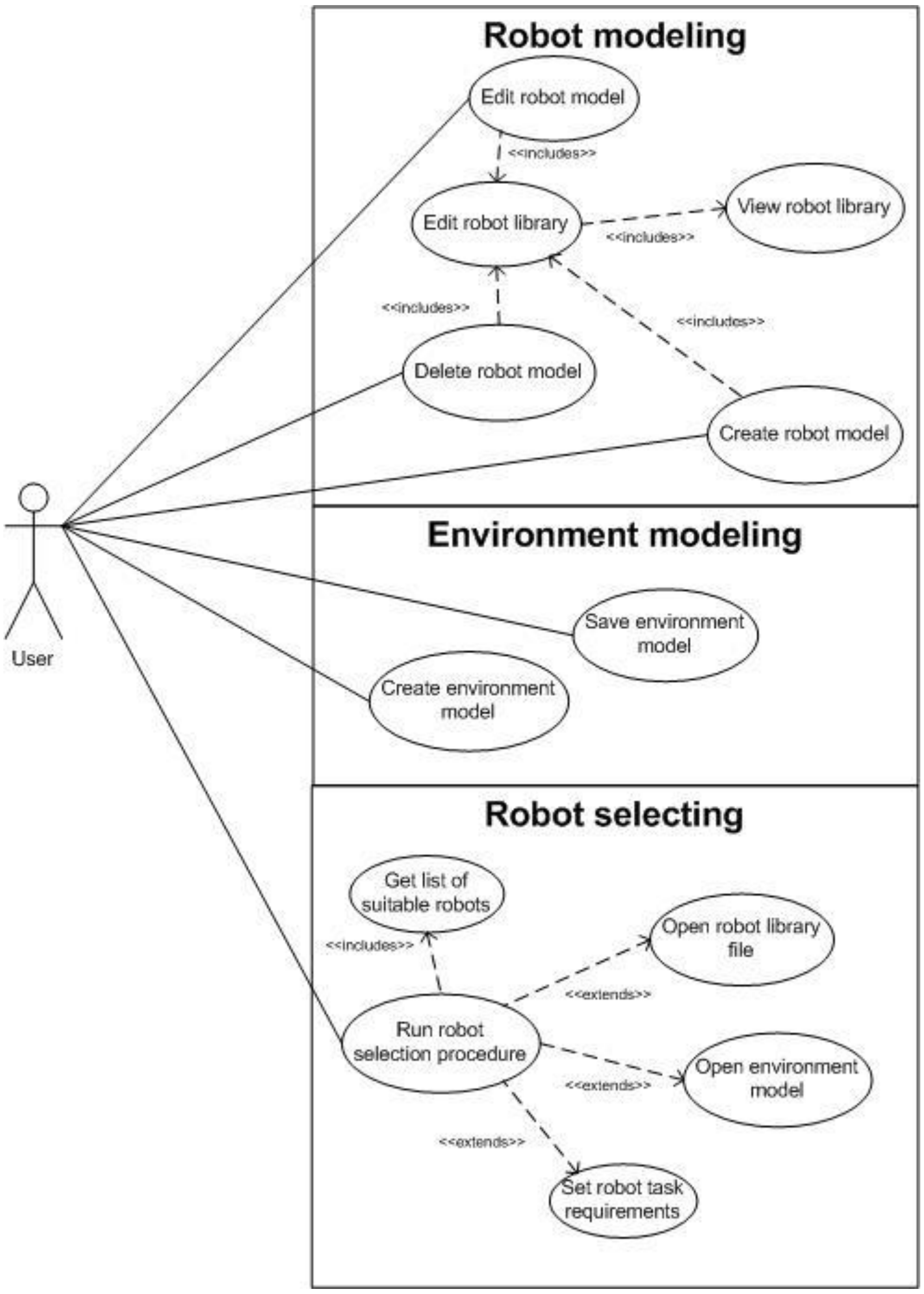


Figure 3.8 *UML use case diagram for the tools*

The robot modeling and the environment modeling tools are actually assistance tools for the robot selecting tool. It should be possible to use the robot selecting tool without other tools. The environment and the robot modeling tools are only for making files for the robot selector. However, the robot selecting tool should be capable to open the environment files made with any 3D modeling tool instead of the environment modeling tool.

Common file formats must be selected between the tools. The environment model file format has already selected. OBJ format is selected as the environment model format.

Therefore, the environment modeling tool should be simple 3D modeling tool that saves the models as obj-files.

For the robot modeling tool, new format must be created. This format should include a list of robots, their DH parameters and other specifications. This custom robot library data format can be implemented as a txt-format.

The robot selecting tool should be capable to open the obj-files and the robot library files. Then the tool should build the robot models and the environment models from the selected files. After this, the user will be able to select task requirements and run the algorithm. After calculation, the tool will list the suitable custom robots for the selected task in the modeled custom environment. Phases of the robot selection process for each tool is shown in UML architecture diagram in *Figure 3.9*.

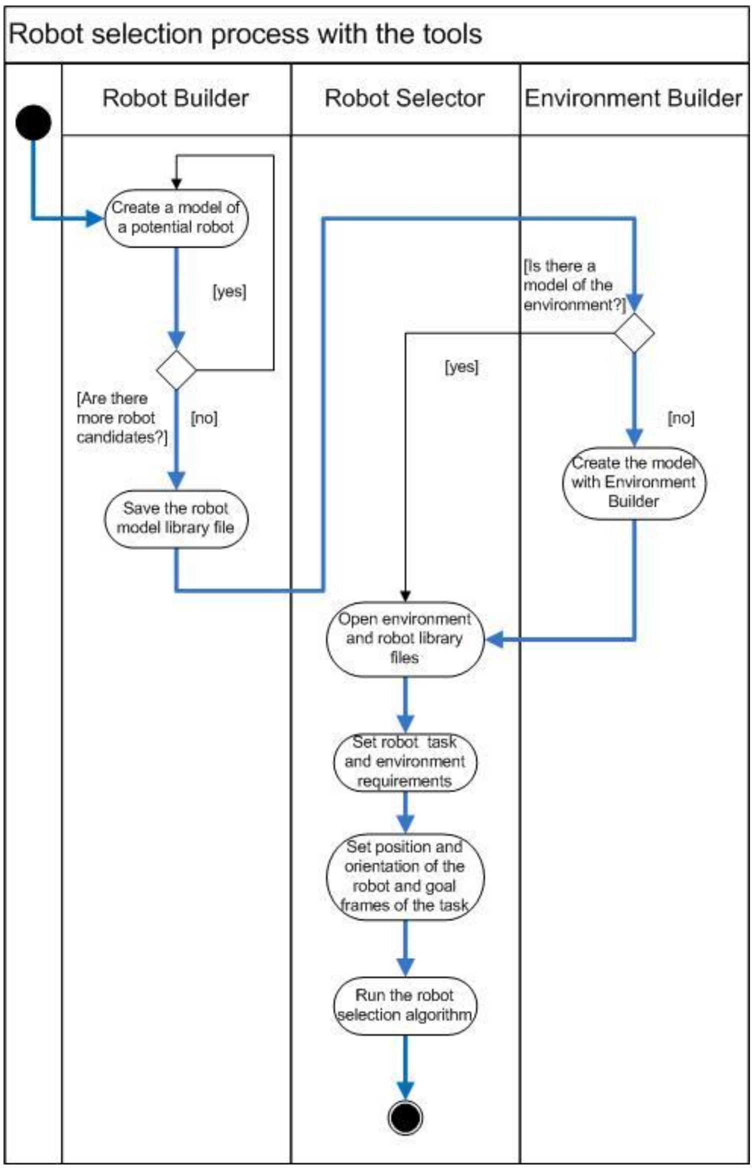

Robot selection process with the tools
Robot Builder
Robot Selector
Environment Builder
Create a model of a potential robot
[yes]
[Are there more robot candidates?]
[no]
Save the robot model library file
[Is there a model of the environment?]
[yes]
[no]
Create the model with Environment Builder
Open environment and robot library files
Set robot task and environment requirements
Set position and orientation of the robot and goal frames of the task
Run the robot selection algorithm

Figure 3.9 *UML activity diagram about robot selection process with the tools*

As we can see from the *Figure 3.9* above, the robot and the environment models are built first using the Environment Builder and the Robot Builder. The robot model and the environment model procedures are described in this chapter. After the models have been made, the Robot Selector is used to find the suitable industrial robots for the task and the environment.

3.9.1 Tool for modeling an environment

For fulfilling requirements of the thesis, the tool for modeling the environment is required. The environment modeling tool will be able to model the environment with a user friendly interface. The user will be able to build the robot working environment model, for example, a robot cell. *Figure 3.10* shows the environment modeling procedure. The model must be measured before it can be modeled. After the measurements, the model can be created by covering all the obstacles with rectangles. Then the model can be saved. The actual modeling procedure will be explained in more detail in the next chapter.

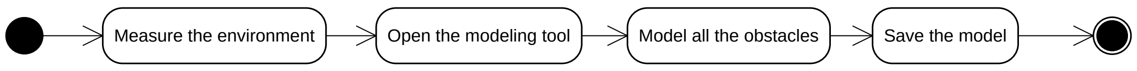


Figure 3.10 *UML activity diagram for modeling environment*

Models can be done using rectangles with sufficient precision. The rectangles will be easily placed, scaled and rotated. The models are only for collision avoidance. Therefore, it is not necessary to model the environment with exact precision. However, almost any shapes can be done with rectangles small enough or inserting multiple rotated rectangles in same location.

When modeling the environment with the rectangles, it is necessary to have a decent interface. The model must be displayed while modeling and the rectangles will be preview before placing them. The user friendly interface also requires features like an undo-button for keeping the tool usable.

After finishing modeling, the user can save the model as obj-file. The obj-file can then be opened in the robot selecting tool.

3.9.2 Tool for modeling industrial robots

Robot modeling is also required in this thesis. With the robot modeling tool, the user can model and design own robots with the user friendly interface. Basic knowledge of robotics is required for this tool (especially DH parameters). *Figure 3.11* shows the robot modeling procedure. Before the modeling, the user needs to get the robot attributes (such as DH table) which can be found for example from robot manuals and drawings. The robot attributes can then be filled into the robot modeling tool. The actual modeling procedure will be explained in more detail in the next chapter.

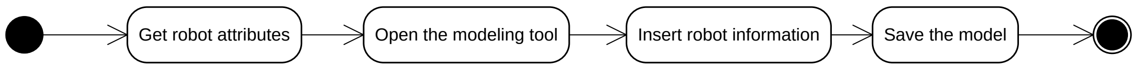


Figure 3.11*UML activity diagram for modeling robots*

The robot modeling tool is based on the DH parameters. The user can model the robots filling the robot DH parameter table while monitoring the current robot model. The user can define robot basic information, like name, maximum payload, the DH parameters and joint limits. Since this thesis is environment- and task-driven robot selection, environment and task details are also defined including task types, temperature range and noise level. With the robot modeling tool, the user should also be able to test the modeled robots. This ensures that the robot moves as it is designed.

The robot models will be stored in the custom file format. Txt-files are useful for storing this type of data. These files can be called robot library files. These files include all the data for each robot model. The user will be able to create new robot library files. In the robot library, the user is able to view the robots, create new robots, delete robots and edit robots.

The robot libraries can be opened in the robot selecting tool. The robot selecting tool then runs an algorithm that finds the suitable robots from the selected robot library in the custom environment.

3.9.3 Tool for selecting industrial robots

Tool for selecting industrial robots is the main tool. This tool will run the main algorithm utilizing inverse kinematics and collision avoidance. Then the tool will list all the suitable robots for given task in the custom environment.

The tool must be able to open the environment and the robot library files. The environment files are in OBJ-format and made with the environment modelling tool. However, the tool will be able to open OBJ-files made with another tools and make a collision model regardless the tool that is used to model the environment. Environment details such as room average temperature or robot maximum noise can also be filled in a GUI of the tool. These details affect the robot selection. The robot can be placed to any location in the environment. The robot can also be rotated along any axis to any orientation. Therefore, it is possible to mount the robot for example to a ceiling or a wall.

Task can also be adjusted. Task type can be selected from a task list. The task types can be for example welding, painting, assembly, tending or any. Task payload can also be selected. This means for example weight of an object being moved. A location of the task will be implemented with adding goal frames. These frames can be for example start- and endpoints of a welding task or picking and placing points of an assembly task.

The user can also adjust other options to avoid long and unnecessary processing. These options include roadmap options (landmark- and iteration limits) and collision options such as a collision distance.

After selecting a robot search, the tool will test the robots from the selected robot library to the required task. The inverse kinematic algorithm is used with the collision- and joint limit avoidance. Then all the succeeded robots are listed and showed to the user.

3.10 Distribution of tools

Contact was made to several companies. The thesis was well explained and co-operation was offered. Some of the companies were interested about the thesis but still no deal was made. This led to free distribution of the tools.

In the present days it is obvious to select internet as the distribution channel of the tools. Internet provides great availability of the tools for everyone.

When selecting a web content management environment, WordPress was the best choice. Over 26% of internet pages is made with WordPress and 50 000 new web pages are created every day. WordPress is free and open-source system and it supports mobile devices and it was easy to learn. WordPress was obviously the best selection as website platform. [42, 44]

The source code is also distributed. The selected channel to distribute the code is GitHub. GitHub is made for distributing and developing especially open source code so it is a good choice for this project allowing anyone to edit it after finishing the thesis.

4. IMPLEMENTATION

In this chapter an implementation of the tools and the webpage is discussed. Three tools were created: Environment Builder for modeling the robot environments, Robot Builder for the robot modeling and Robot Selector for solving the main problem. Webpage was created for distributing the tools.

4.1 Environment Builder

Tool for modeling the robot environments is named Environment Builder. Environment Builder is a tool for building 3D-models for another tool called Robot Selector. This tool saves the models to obj-files. The environments can be modeled with any other 3D-modelling tool as long as the format of the environment is obj.

Environment Builder is simple and good enough for modeling the robot environments like robot cells. Environment Builder is based on rectangles. The user can choose start and end points for each axis. These six values define the rectangle. The rectangles can also be rotated with any amount of degrees along any axis. The user sees a preview of the current rectangle all the time while filling the values in GUI. A shape adding algorithm is described in Algorithm 4.1.

```
1. get values from GUI
2. calculate and save midpoint
3. get rotation values from GUI
4. select corners of the rectangle and save to new matrix
5. create variable matrix for rotated coordinates
6. while loop for all corner points of rectangle:
    a. reduce object midpoint from coordinates to rotate
       object along itself, not with origin
    b. rotate coordinates using transformation matrix
       and save rotated coordinates
    c. add coordinates to midpoint to move point back
       to original location but rotated
    d. select next point
7. create rectangle matrices X, Y and Z
8. create and update vertices and face matrices
9. remove preview shape
10.    draw new shape to figure
```

Algorithm 4.1 *Add shape*

Each time after adding the rectangle, vertex and face matrices are updated. The vertex matrix simply includes coordinates of corners of the rectangle. The matrix is $3 * 8$ sized since the rectangle has 8 corners. The face matrix is used to connect the vertices to each other to construct the polygons. Size of the face matrix is $3 * n$ depending on a number of the rectangles. For each rectangle, 12 rows are required since the rectangle can be created with minimum of 12 triangle-shaped polygons. The face matrix has a constant pattern and it always uses the pattern numbers and sums the multiplication of 8 and number of shapes to the pattern.

The user can also undo the shapes. An undo-button removes the latest rectangle. The button can be used as many times in a row as user wants. Algorithm 4.2 shows the principle of undo-function.

```
1. continue if number of shapes > 0
2. delete shape i (newest shape)
3. while loop for all faces and vertices of rectangle
    a. delete face of shape i from face matrix
    b. delete vertices of shape I from vertices matrix
    c. increase counter
4. decrease shape counter i
5. update figure
```

Algorithm 4.2 *Undo*

The user is also able to clear all and start the modeling all over again. This removes all the rectangles. The finished model can be saved with a save-button. The save-function simply prints the vertex and face matrices to txt file and names it as obj-file.

Graphical user interface (GUI) of the Environment Builder is simple. The GUI of Environment Builder is shown in *Figure 4.1*. The GUI includes only the preview image of the current environment, location fields, rotation fields and four buttons.

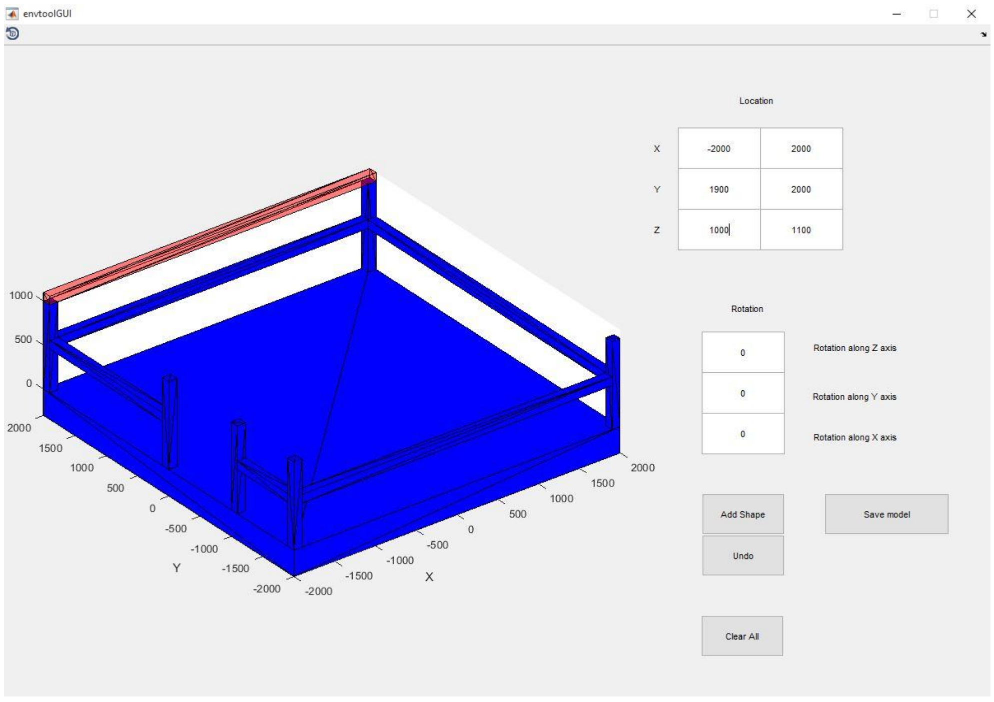


Figure 4.1 *Environment Builder GUI*

Six location fields are used to determine the rectangle. Fields are for start- and endpoints of the rectangle in X, Y and Z coordinates. Three rotation fields are used for rotating the rectangle along the X, Y and Z axes.

The four buttons are included. The add shape –button creates a currently previewed (red) rectangle. The undo-button removes the latest rectangle and converts it to the preview-shape. The clear all –button removes all the rectangles. The save model –button creates the obj-file and saves the rectangles to the file.

When modeling the environment with Environment Builder. The first thing is to measure the environment, which can be done for example by creating drawings of the environment. The user can simply get the dimensions of the environment utilizing already existing CAD models. Alternative method is simply to go in front of the environment and select any point as the origin and then measure the distances and dimensions of the obstacles with a ruler. After the measurement, the user should be able to locate the obstacles in the coordinate system of the modeling tool. The most important part is to measure location of the obstacles. All the obstacles must be located, which may cause collisions.

After the measurement phase the modeling can be started. When using the Environment Builder, all the obstacles are covered by rectangles. The rectangles are defined with coordinates of two opposite corners. When all the obstacles are covered, the model is ready and it can be saved. The environment modeling procedure is shown in *Figure 4.2*.

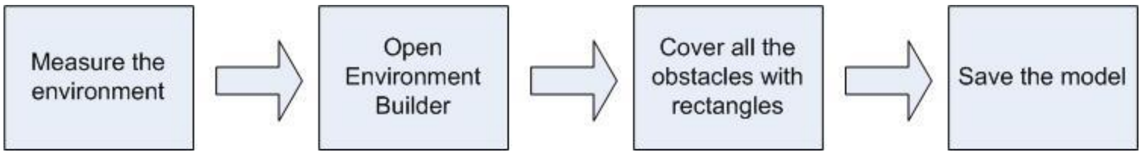


Figure 4.2 *The environment modeling procedure*

When creating the environment model. The most important thing is to locate the shapes at right place. The number of the shapes or looks of the model does not matter, since the model is only used for the collision avoidance. For example, if the model includes a table, the table can be covered with single rectangle if the robot does not have to reach under the table. Multiple obstacles can also be covered with single rectangle, if the robot is not required to reach between the obstacles. Actually small amount of the shapes decreases processing time of the robot selection process. *Figure 4.3* shows an example of robot model and a sufficient model.

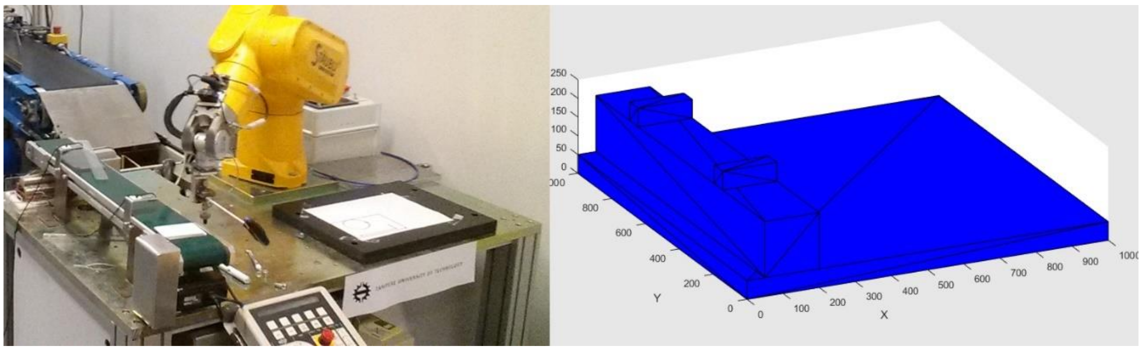


Figure 4.3 *Robot environment (left) and a sufficient collision model (right)*

For example, if the task of the robot is to pick up an item from a table and place it to a conveyor, the required model does require only a few rectangles to cover all the obstacles. *Figure 4.3* show the robot environment and the sufficient model of the corresponding task for Robot Selector.

When creating the models, the most time consuming phase is to measure the location of the obstacles. The modeling phase usually takes only dozen minutes.

4.2 Robot Builder

Robot Builder is the tool for modeling the robots and creating the robot model libraries. Robot Builder includes a preview image, so the user can all the time observe the robot being modeled. The tool has fields for basic robot information like robot name, maximum payload and suitable applications. The GUI also has fields for the robot DH-parameter table. The user models the robots filling the DH-table. Joint limits must also be filled.

While modeling, the user can move robot joints using sliders in the GUI. The tool will update the preview image with new joint values. Therefore, the user can double-check that the robot model works the way the user wanted. Robot Builder also allows the user to browse and edit the robots in the robot library. The robots in the robot library are listed at listbox in the GUI.

The user can save the robots with a save robot –button. An algorithm behind the button is presented below as Algorithm 4.3.

```
1. get some values from GUI
2. read strings from robot task list
     a. add identifier character for each identified task
3. get robot name from GUI and remove whitespaces
4. save line: robotname jointnumber taskchars payload
   mintemp maxtemp noise
5. for each joint
     a. if sigma = 0
          i. save line: theta d a alpha sigma min-
             limit(rad) maxlimit(rad)
     b. else
          i. save line: theta d a alpha sigma min-
             limit(mm) maxlimit(mm)
6. delete robot
7. update robot
```

Algorithm 4.3 *Save robot*

A delete robot –button executes an algorithm that deletes the current robot. The algorithm is described below as Algorithm 4.4.

```
1. get variables from GUI
2. scan file with 8 strings
3. loop while i = 0
     a. try (continue if no errors occur)
          i.  find next robot name and joint number
         ii.  if robot names match selected robot
                1. read lines amount equal of robot joints
                2. read rest of the lines and save them
                   to temp file
        iii.  else
                1. for all lines of current robot
                     a. read and save line to temp file
     b. catch (error occurred)
          i.  set i to 1
4. copy temp file to original file
```

Algorithm 4.4 *Delete robot*

The tool also has buttons for creating new robots in the robot library, creating new robot libraries and saving the robot library as.

Adding a new robot to the robot library follows a robot modeling procedure. The user must execute six steps in order to add the robot to the library. The modeling procedure is shown in *Figure 4.4.*

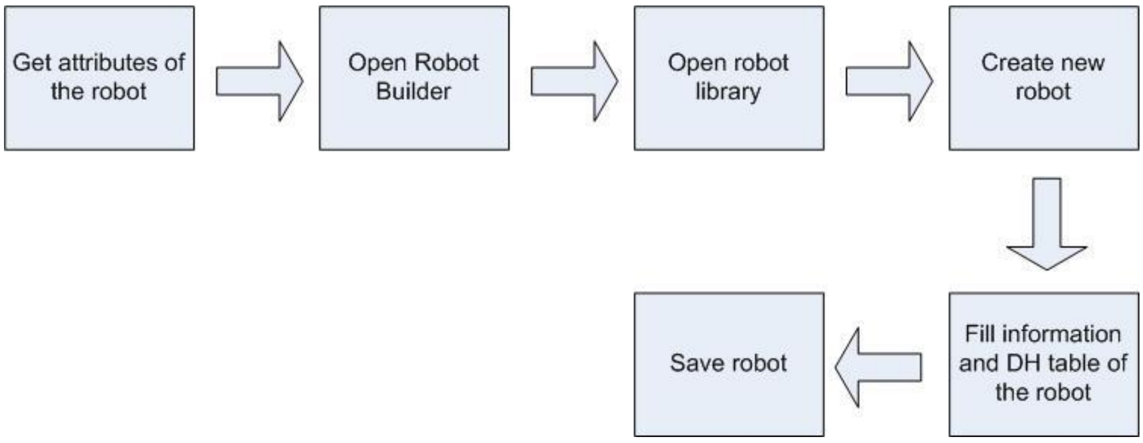

Get attributes of the robot
Open Robot Builder
Open robot library
Create new robot
Fill information and DH table of the robot
Save robot

Figure 4.4 *Robot modeling procedure*

When modeling a new robot, the modeling procedure starts by getting a required information of the robot. The information consists of DH table and other details of the robot. Robot sheets, manual and drawings can be used as source of the information. Robot DH parameters can be found on robot manuals. If the manuals do not include the DH parameter table, the user can create one by utilizing robot drawings. In this phase, the skill to generate the parameters from the drawings are required. Alternative method to model the robot is to use trial and error technique when filling robot DH parameters into Robot Builder. When the user gets accessed to the information, Robot Builder can be opened. The user then selects a robot library which the new robot will be added by clicking “select robot library file”. Then the new robot can be added by clicking “add new robot”. The user is now able to fill the robot information fields. When the fields are filled, the library can be saved by clicking “save robot”, which updates the robot data to the selected library file.

4.2.1 Robot library format

Robot library format includes data of the robots in the library. Robot names, suitable applications, payloads, DH-table and joint limits are stored in the library. The library file can include any number of robots.

The robot library format was implemented in txt-format. The data is stored in a matrix manner: the data is stored in rows and columns. Cells are separated from each other with whitespaces. Table 4.1 shows the structure of the library files.

Table 4.1 *Robot library format*

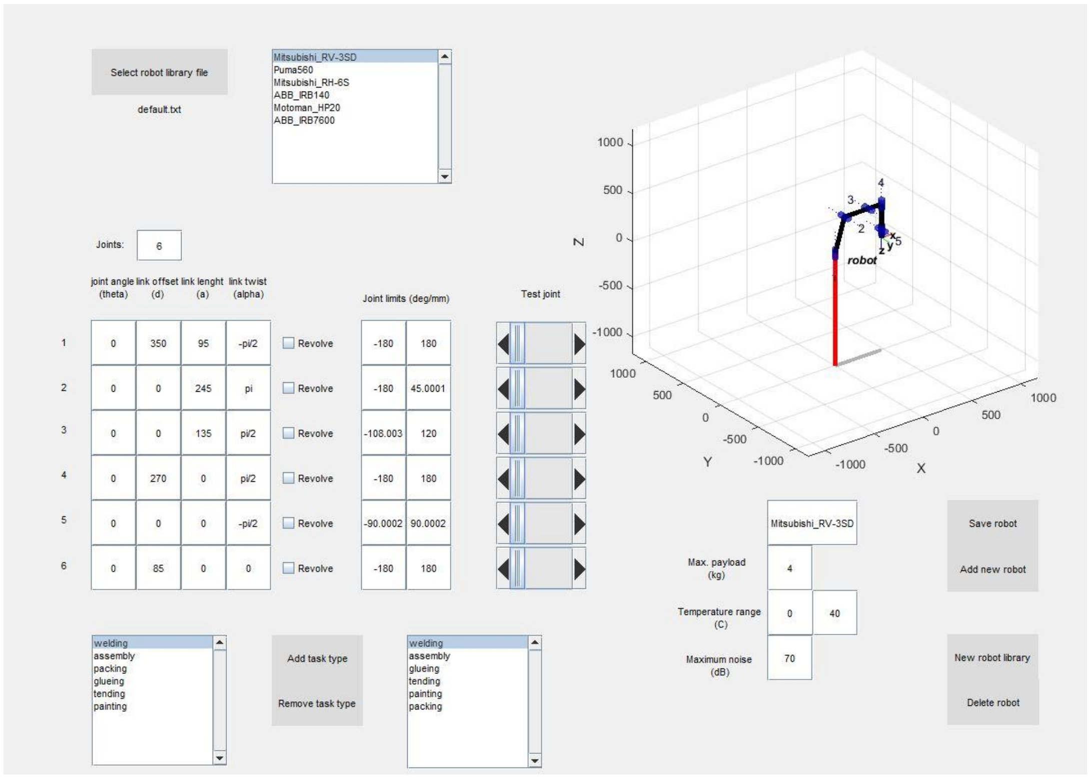
robot name	number of joints	suitable tasks	maximum payload	minimum tempera-ture	maximum tempera-ture	maximum noise
theta 1	d 1	a 1	alpha 1	sigma 1	joint 1 limit (min)	joint 1 limit (max)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
theta n	d n	a n	alpha n	sigma n	joint n limit (min)	joint n limit (max)

The first row of the library contains a main data of the first robot: robot name, number of joints, suitable tasks, payload limit, temperature range and noise limit is defined. Robot name is the name of the robot. Number of joints is an amount of joints in the robot model. This value is also used to keep track while reading the file since the number indicates how many rows will be read until the next robot model. Suitable tasks -cell is a string containing a set of task id characters. For example, if the cell contains the task id characters 1wa, the robot would be suitable for tasks any (1), welding (w) and assembly (a). Each task type has its own id character. Maximum payload is the robot’s maximum payload in kg. Two next cells are for a temperature range in °C. Maximum noise is a maximum noise limit in dB that the robot is able to cause. All the remaining rows of the current robot model contains a data of the robot joints. The rows repeat the following structure as many times as number of joints -cell indicates: θ, d, a, α, Σ, minimum limit of the current joint and maximum limit of the current joint. θ, d, a, α and Σ are the values of the robot's DH-table.

After the whole DH-table is read of the current robot, the robot model is finished and the library continues with the next robot. The structure of the Table 4.1 is repeated as long as there are robots in the library.

4.2.2 Robot Builder GUI

The GUI of the Robot Builder (*Figure 4.5*) displays the robot library data and allows the user to edit and add robots to the library. Understanding of the robot DH table is required to model the robots and use the interface.



default.txt

Mitsubishi_RV-3SD
Puma560
Mitsubishi_RH-6S
ABB_IRB140
Motoman_HP20
ABB_IRB7600

Joints: 6

	joint angle (theta)	link offset (d)	link lenght (a)	link twist (alpha)		Joint limits (deg/mm)	
1	0	350	95	-pi/2	☐ Revolve	-180	180
2	0	0	245	pi	☐ Revolve	-180	45.0001
3	0	0	135	pi/2	☐ Revolve	-108.003	120
4	0	270	0	pi/2	☐ Revolve	-180	180
5	0	0	0	-pi/2	☐ Revolve	-90.0002	90.0002
6	0	85	0	0	☐ Revolve	-180	180

welding
assembly
packing
glueing
tending
painting

welding
assembly
glueing
tending
painting
packing

Mitsubishi_RV-3SD

Max. payload (kg)	4	
Temperature range (C)	0	40
Maximum noise (dB)	70	

Figure 4.5 *Robot Builder GUI*

List of the robots in the library is located at top of the GUI. By clicking the robot in the list, data in the GUI automatically updates with the selected robot's data and a preview figure. The selected robot is editable. The DH-parameters, suitable applications, payload limits, temperature range, joint limits and the name of the robot can be edited. The joints can also be moved within the joint limits.

At the right side of the GUI, four buttons are located. With the buttons a new robot or a new library can be created, the selected robot can be saved or deleted or the robot library can be saved as another library file.

4.3 Robot Selector

The tool for selecting the suitable robots from the robot library in the custom environment is called Robot Selector. This tool includes the algorithms for the main problem (an inverse kinematic solver with obstacle avoidance).

With the Robot Selector, the user selects an environment model for the robot and a library of robots being tested. An algorithm for opening the OBJ-file into MATLAB was taken from Alec's Web Log website [4]. After selecting the files, task requirements are filled (payload of the task, application etc.). Goal frames are then created which includes positions and orientations of the task. When all the requirements are filled in, the user starts the robot selection process. Principle of the robot selection procedure is shown in *Figure 4.6* as UML activity diagram.

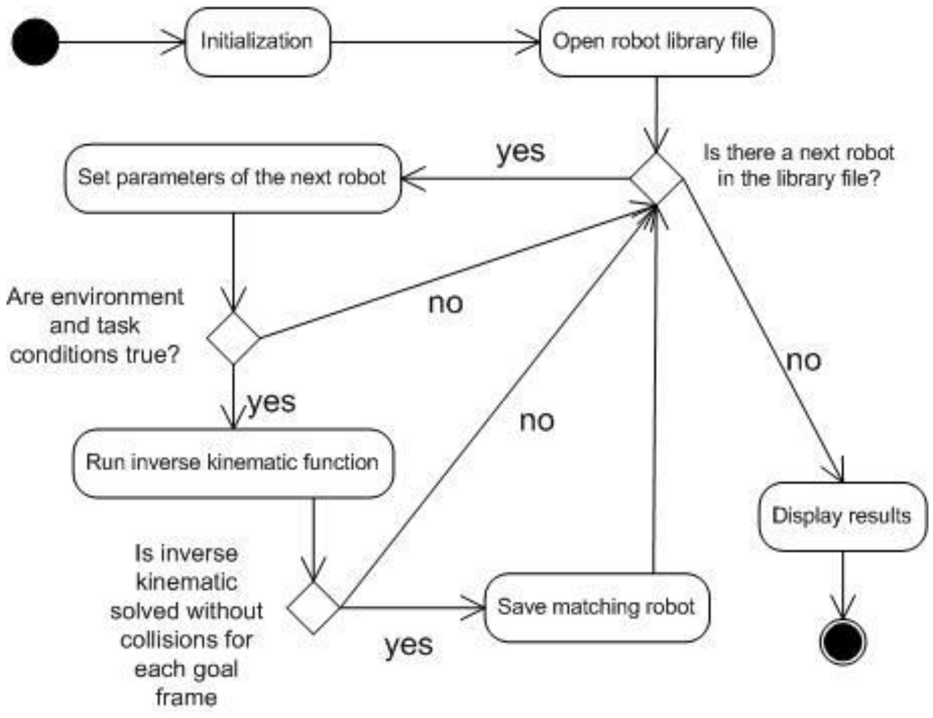

Initialization
Open robot library file
yes
Is there a next robot in the library file?
Set parameters of the next robot
Are environment and task conditions true?
no
no
yes
no
Run inverse kinematic function
Display results
Is inverse kinematic solved without collisions for each goal frame
Save matching robot
yes

***Figure 4.6** UML activity diagram of the robot selection procedure*

The procedure simply tests one robot at time from the robot library. At first, the environment and task conditions are checked. Then an inverse kinematics problem is solved within joint limits and while avoiding collisions. If the robot fulfills the environment and task condition or any goal frame is not reachable, the robot is not suitable. Otherwise, the robot is suitable. This process is repeated until all the robots are tested. Then the results are shown to the user.

After the inverse kinematic solver was finished, the GUI of Robot Selector and all the code of the tool was completely remade. The tool was started almost year ago as a special assignment so the code was tangled after many changes and tests of different inverse kinematic methods. The GUI and the code was then designed and programmed all over again to get an understandable, efficient and reliable code.

4.3.1 Inverse kinematics

In this implementation, the inverse kinematic problem is solved with Ariadne's Clew algorithm –based solution. Juan Ahuactzin and Kamal Gupta have made successful implementation of solving a point-to-point inverse kinematic problem utilizing the Ariadne's Clew algorithm in 1998-2000. This method was carefully studied and modified for getting a general solution with any serial manipulator in any environment. [2, 3]

In this method, the inverse kinematic function simply repeats explore and search -phases. The repeating continues until cost function reaches selected accuracy or an iteration limit exceeds. Algorithm 4.5 shows the principle of the inverse kinematic solver.

```
1. Initialization
2. While cost-function > cost resolution and counter <
   landmark limit
     a. EXPLORE
     b. SEARCH
     c. Update closest solution
     d. Increase counter
3. Return results
```

Algorithm 4.5 *Inverse kinematics solver*

Explore sub-algorithm have been modified to speed up the calculation time and extended to work in any environment and with any robot. The first modification in the algorithm creates the robot with correct amount of joints from input data. The robot DH-table is required in order to calculating forward kinematics. The second modification is that each embryo is created by setting the joint values to random between the joint limits. The original method of explore creates random movements from each previous landmark. The difference between the methods is that the random movements are designed to "bounce" at collisions. In my method the robot is set to the random orientation and if collision occurs, the embryo is discarded. My method is designed in this thesis to speed up the algorithm since it does not need to save the joint movements. Also the original method requires a constant robot starting position, which causes multiple disadvantages since the structure of robot and the environment is unknown. It is also empirically noted that this method is faster than the original one. Third modification to the original algorithm is that always 10 embryos are created instead of one for each landmark. As it is noted in *Figure 3.2* at chapter 3.1, the landmarks are set regularly even with this method. 10 embryos are empirically selected number that is large enough to create the landmarks far away from each other and still prevent the algorithm to slow down with large amount of the landmarks. The Explore algorithm is shown below as Algorithm 4.6. [2, 3]

```
1. Create robot
2. Set initial values
3. While ii <= 10
    a. For i = 1 to number of joints
         i. Select link i
        ii. rand1 = set random number between joint lim-
            its
       iii. q0(i) = rand1
        iv. Calculate forward kinematics
    b. Initialization
    c. Check collision
    d. If collision does not occur
         i. For i = 1 to amount of landmarks
              1. Calculate distance to landmark i
              2. If distance < markdistance, set this
                 distance as markdistance
        ii. If markdistance > markmaxdist
              1. set markmaxdist as markdistance and
                 set this embryo as landmark
              2. qi = q0
       iii. Increase ii by 1
4. Update landmark matrix
5. Return landmark matrix and qi vector
```

Algorithm 4.6 *Explore*

The Search algorithm has also been under large modification. Many changes were made to make this iterative algorithm as fast and as accurate as possible. Three different versions were made, and one of them was faster and more accurate than others beyond any doubt. The first search algorithm was accurate, but also slow, since for each joint, the robot moved one joint for one degree in a limitsearch phase and a cost function phase and then made calculations (collision detection or cost function depending on the phase). Therefore, there was a lot of unnecessary calculation.

The second implementation was a mix of the first and the third method. But the third one was most efficient. The principle of this method is presented in *Figure 4.7*. From the another joint limit, cost function is calculated for every 10 degrees of robot joint movement (red sample). Then the result of the function is compared to the previous result. If the cost function keeps decreasing, the sampling frequency stays constant and next sample is taken after next 10 degrees of movement. However, if the cost function is increased from the previous sample, the pointer moves 2 samples backwards (to make sure that the local minimum is not passed) and increases the sampling frequency to 1 degree (blue samples). When the cost function of blue samples starts to increase, the previous sample is set, as the local minimum and the frequency is then decreased back to the red sample's frequency. Then the process is continued until the other joint limit is reached.

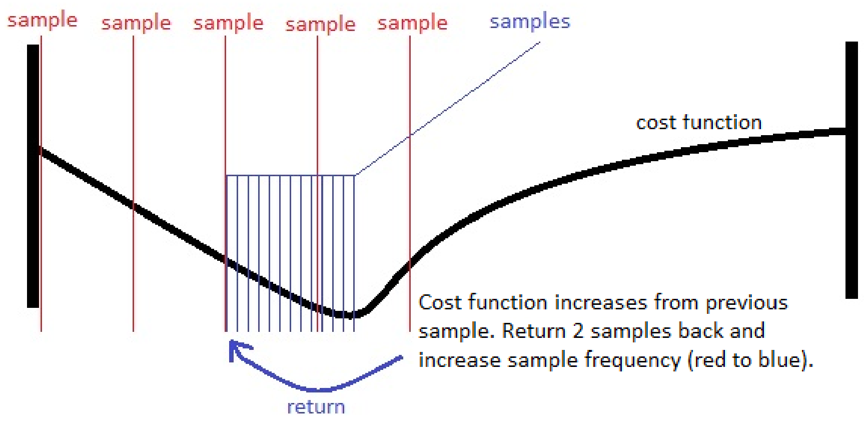

sample
sample
sample
sample
sample
samples
cost function
Cost function increases from previous sample. Return 2 samples back and increase sample frequency (red to blue).
return

Figure 4.7 *Search iteration principle*

The Search algorithm is complex. The principle of the algorithm is explained below in Algorithm 4.7.

```
1. Create robot
2. Set initial values
3. While counter < iteration limit
     a. Set joint iteration direction
     b. For each joint:
          i. Find joint limits:
               1. Move forward
               2. Test collision
               3. If collision occurs or limit is found,
                  set previous value as limit and change
                  direction until both limits are found.
                  Otherwise continue movement
         ii. Find local minimum:
               1. Move joint
               2. Calculate cost function
               3. If cost function starts to increase,
                  return back and increase frequency.
               4. If cost function is lower than before,
                  set this position as local minimum
               5. If cost function is increasing, de-
                  crease sample frequency
     c. End search if position found, cost function is
        not decreased enough compared to previous iter-
        ation or iteration limit is exceeded.
4. Return joint values and cost function value
```

Algorithm 4.7 *Search*

4.3.2 Collision detection

Collision detection is implemented with the distance measurement method. The algorithm measures the distance between the robot and the environment after each time the robot is moved. If the distance reaches a given limit, collision occurs. This limit is called collision resolution.

When implementing the collision detection, we need to know the distance between robot and the environment. In this implementation, it is a distance between closest point of the robot to a polygon of the environment.

The environments are built from the polygons. The first problem is to fill all the polygons with points at regular intervals. After this phase, the closest point can be found and the distance from the robot to the environment can be calculated. When solving the polygon filling problem, the most optimal solution found is to utilize vectors. Any of the polygon’s vertices can be selected as point1. From the point1, one of the polygon's edges can be selected as a vector v_1 and another edge as a v_2. These vectors can be calculated reducing point1's coordinates from the destination point's coordinates. When adding a point inside the polygon, it is first needed to add the coordinates of the point1. Then when adding the vectors v_1 and v_2 multiplied with variables m and n. The new point will end up inside the polygon, if

$$m + n \leq 1. \tag{4.1}$$

The vectors are clarified in *Figure 4.8.*

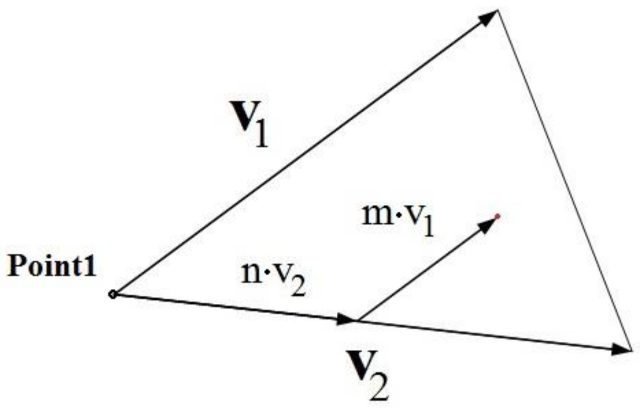


Figure 4.8 *Polygon vector clarification*

Next problem is to calculate the variables m and n to get the points inside the polygon with regular intervals. Each polygon in the model can be different sized and shaped compared to the other polygons of the model. The length of the v_1 and v_2 can also be different. This causes problems when having constant values as point interval. Two methods were developed for the problem.

The first method modifies the values of m and n for each polygon to modify the point intervals. The optimal values for m and n can be calculated to get decent interval between the points for each polygon. A variable for controlling the point interval is called accur (environment model accuracy). The point interval for each polygon can be calculated utilizing lengths of the vectors. If accur1 is a point interval factor for a certain polygon, we can generate a formula

$$accur1 = \frac{lenght\ v_1}{lenght\ v_2} * accur \tag{4.2}$$

or as

$$accur1 = \frac{\sqrt{(x_{v1})^2 + (y_{v1})^2 + (z_{v1})^2}}{\sqrt{(x_{v2})^2 + (y_{v2})^2 + (z_{v2})^2}} * accur \quad (4.3)$$

to calculate the point interval for the polygon. The maximum value for the accur1 can be set to $1{,}3 * accur$ and the minimum value to $0{,}3 * accur$.

Now the variables m and n can be calculated using formulas

$$n = \frac{lenght\ v_1}{accur1} \quad (4.4)$$

and

$$m = \frac{lenght\ v_2}{accur1} \quad (4.5)$$

to get the point intervals corresponding to accuracy of the selected environment model.

Differences between lengths of the v_1 and v_2 causes some of the polygons to have points with decent intervals and some the polygons only with few points even if area of the polygon is large. *Figure 4.9* shows an environment before and after the point interval optimization. Some of the polygons have few points while some of the polygons are filled decently.

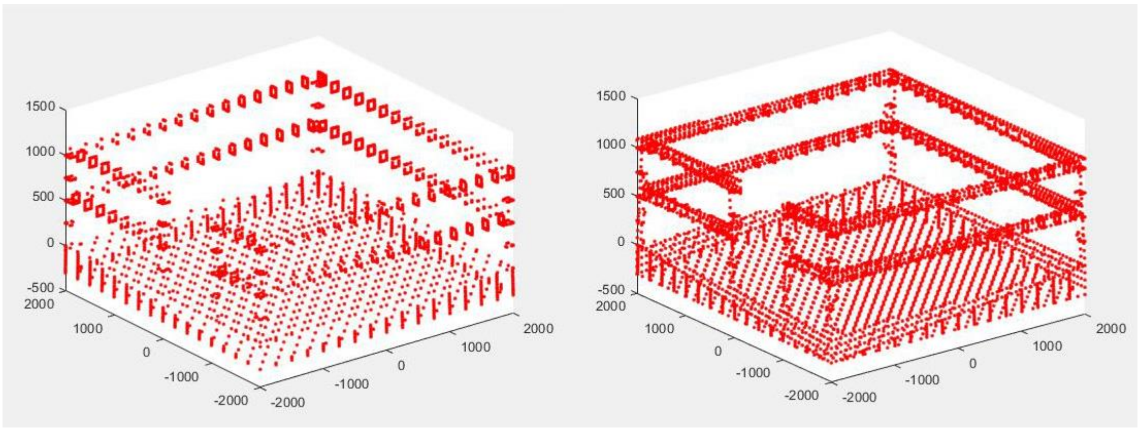


Figure 4.9 *Before and after a point interval optimization*

The point interval optimization is required to prevent collisions. If the environment point model is not accurate enough or it includes caps, the tool could interpret sparse points as free space.

Adding the points in the polygon is implemented with while-loop. The loop keeps the point interval regular utilizing values m and n. The counters $c1$ and $c2$ are used for filling the polygon. The loop lasts while

$$c1 * accur1 < lenght\ v_1. \tag{4.6}$$

The counter $c1$ starts from 0. At the beginning of the loop the counter $c2$ is set to 0. Then starts another loop that lasts while

$$\frac{c1}{n} + \frac{c2}{m} < 1 \tag{4.7}$$

to keep the points inside the polygon. In the loop coordinates x, y and z are saved to matrices X, Y and Z. The counter $c2$ is then increased by 1 and the second loop ends. After the second loop, one row of points is filled in the polygon. Then the counter $c1$ is increased by 1 and filling of another row is started. After these two loops are executed for each polygon. The environment model is filled with the points with the given accuracy. *Figure 4.10* shows an environment built with Environment Builder and converted to points with this algorithm utilizing the point interval optimization.

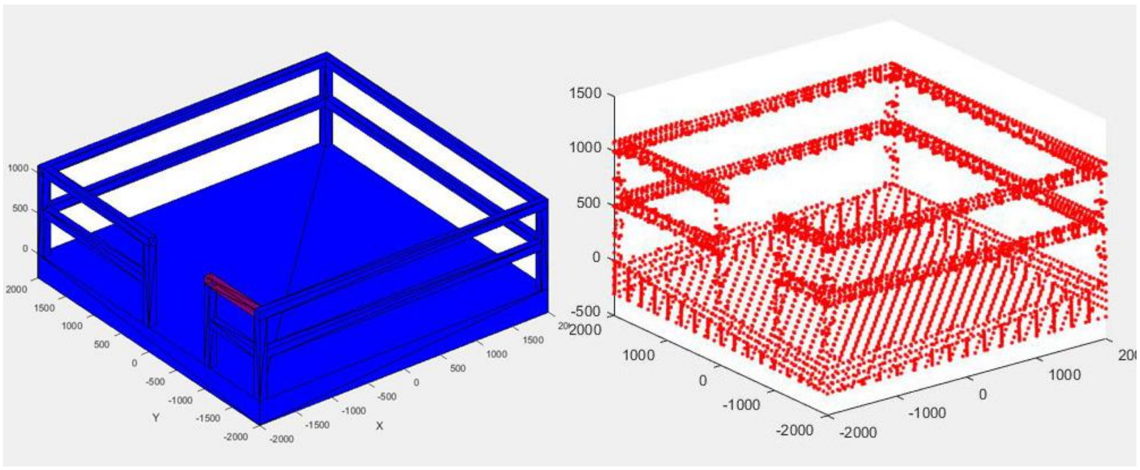


Figure 4.10 *Environment conversion to set of points*

Several obj-files were tested. Results were good with different sizes of polygons. Amount of the polygons was also varied. The problem was gaps in the polygons caused by different sized edges of the polygons. Depending which vertex is chosen as the point1, gaps may occur even with the point interval optimization. For narrowing the gaps, the point interval must be set low. This leads to large amount of the collision model points, which causes lot of calculation in the collision detection algorithm and slows the robot selection tool.

The second method is developed later to prevent the unnecessary calculation and make the code faster and more efficient. This method discards the idea of setting point intervals separately for each polygon. The method is based for selecting optimal vertex as the point1. The optimal vertex is always opposite of the longest polygon edge. This can be implemented with simple if-conditions. *Figure 4.11* shows an environment filled with the second method.

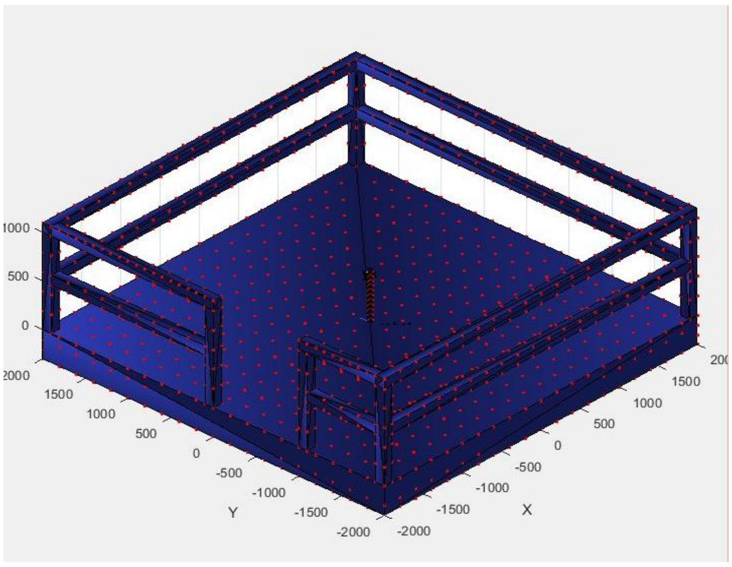


Figure 4.11 *Results of point1 selection method*

The environment collision model point generating algorithm is shown below (Algorithm 4.8).

```
1. Set initial values
2. Import an environment model and generate vertices and
   faces matrices
3. For each line in faces matrix:
     a. Get corresponding vertices from the vertices ma-
        trix
     b. Calculate lengths of each edge
     c. Set vertex opposite to longest edge as point1
     d. Set vectors and point intervals
     e. Set counter c1 to 0
     f. while c1*accur< length
          i.  Set counter c2 to 0
         ii.  while c1/n+c2/m<1
                1. Save coordinates of the point to ma-
                   trices X, Y and Z
                2. Increase c2 by 1
        iii.  Increase c1 by 1
4. Return X, Y and Z
```

Algorithm 4.8 *Environment model point generator*

After the points have been saved to matrices X, Y and Z, it can be easily calculated the distance to the closest point. For all values from X, Y and Z, the Euclidean distance will be calculated from desired point using the Pythagorean formula. The distance is first set to infinite. If the calculated distance is shorter than previously, the current distance is saved to the variable. Then the distance to the next point is measured. When the distance to all the points have been calculated, the point with shortest distance is the closest one and the distance to that point is the distance to the environment. The distance calculation algorithm is shown in Algorithm 4.9. Note that many coordinates can be set to input of the distance calculator algorithm.

```
1. Set initial values, k = 1
2. While k <=size(X)
     a. i = 1
     b. While I <= size(P)
          i. Get coordinates from matrix P
         ii. Calculate distance from current point of
             matrix P and coordinate from matrices X, Y
             and Z at row k.
        iii. If distance < mindist distance
               1. mindist = distance
               2. Save current coordinates of P and XYZ
                  to matrix S
         iv. Increase i by 1
     c. Increase k by 1
3. Return matrix S and minimum distance
```

Algorithm 4.9 *Distance calculator*

The points from the robot model are generated regularly for each link. Algorithm 4.10 shows the robot model point generation. For all the points of the robot, the distances are measured to the environment points.

```
1. Set initial values, i to 1 and counter count to 0
2. While i = 1
     a. Calculate location of next joint of the robot
     b. Measure distance between previous and current
        joint
     c. Set point interval to distance/accur and counter
        c to 1
     d. While counter c*accur<distance
          i. Add point to matrix P and increase counter
     e. Set current point to previous point
     f. If count = amount of joints
          i. Set i to 2
     g. Increase counter count by 1
3. Return matrix P
```

Algorithm 4.10 *Robot model point generator*

A point with shortest distance to the environment is the closest point to the environment. Now the distance between the robot and the environment can be measured. *Figure 4.12* shows the result of the distance measurement algorithm from the robot to the environment model.

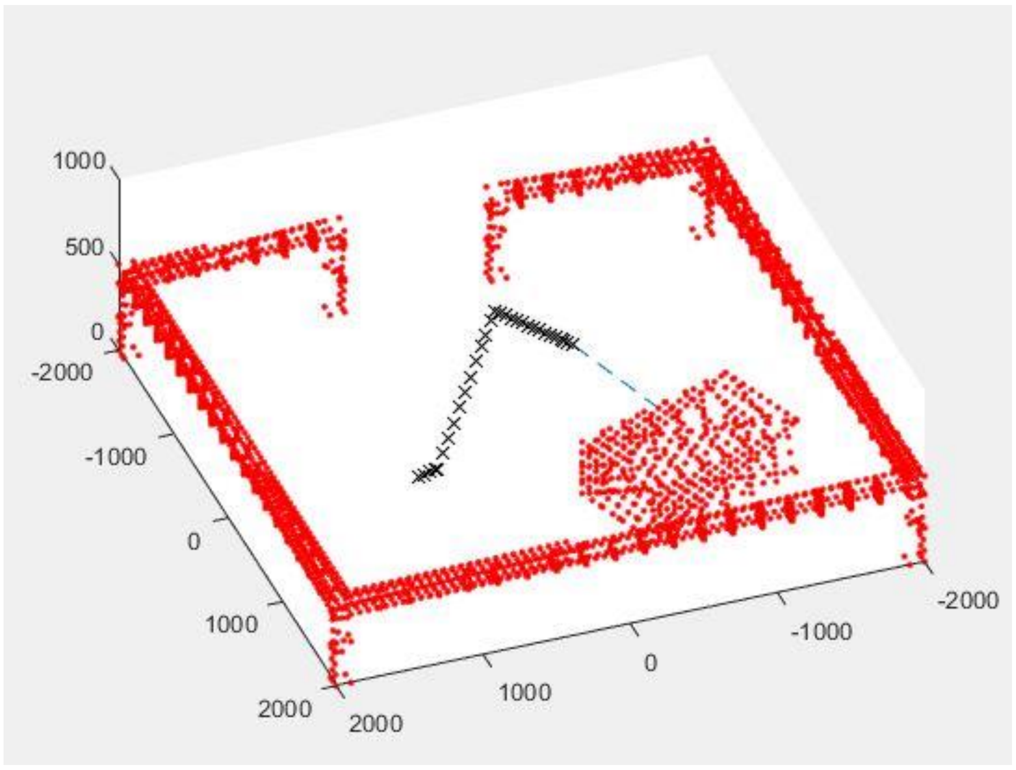


Figure 4.12 *Measuring distance between robot and environment*

In the figure, the closest distance to the environment is between an end-effector of the robot and a table in the environment. A floor should be left from the model if the robot is mounted to it. Otherwise shortest distance will be from the robot mounting point to the floor. That might disturb collision avoidance since the robot distance to the environment is always constant and may be within the collision distance.

4.3.3 Obstacle avoidance

Collision avoidance components are included in the Search and Explore algorithms so no separate collision avoidance algorithm is needed. In the limitsearch part of the Search function, the collisions are detected and the robot joint limits are modified to avoid collisions. The explore function discards an embryo if it causes collision.

These methods ensure that no collision are able to occur in any case when robot joint values are modified. The collision is defined as a distance between the robot collision model point to the environment collision model point. If this distance is smaller than the collision distance, then collision occurs. This situation cannot happen in Explore or Search algorithms.

4.3.4 GUI of Robot Selector

Robot Selector’s GUI was developed gradually from the special assignment. As more features was made, the required fields were added to the GUI. Therefore, the GUI was never really designed. After finishing the inverse kinematic solver, the final requirements for the GUI was understood. The code of the Robot Selector and the GUI was then designed and recreated. *Figure 4.13* shows the previous GUI in a left side and the current GUI in a right side.

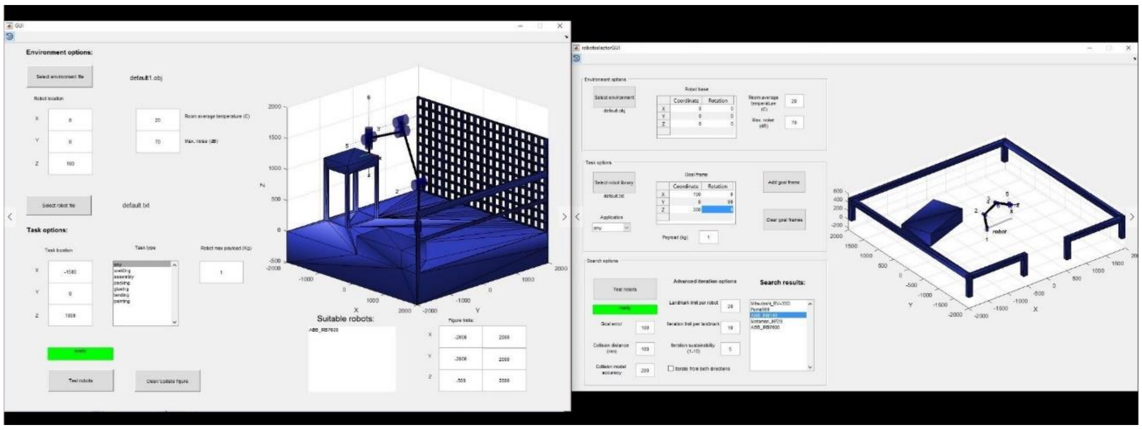
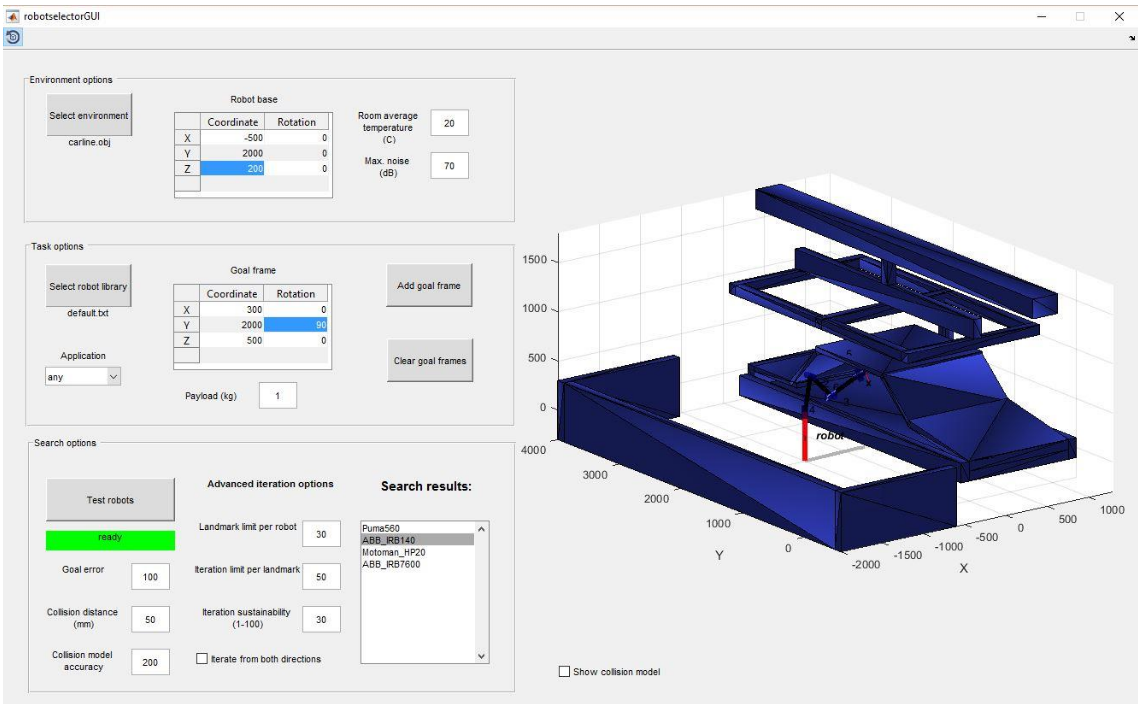


Figure 4.13 *Previous (left) and current (right) GUIs of Robot Selector*

The current GUI (*Figure 4.14*) has three panels in it. The first panel is an environment panel. It includes a button for selecting an environment model file, a robot location and an orientation table, average temperature of an environment and a maximum noise the robot is able to cause. Second panel is a task panel. It includes a button for selecting a robot library file, a task application selector and a payload field. In the second panel, goal frames are also created. Coordinates and orientation values of the frames for each axis is filled. The frame is shown in a preview image in real time. Add goal frame –button creates the selected goal frame. Clear goal frames –button removes all the created goal frames. The final panel is search options –panel. It includes a button for starting the selection process. Under the button, a status bar is displayed. A goal error –field is for selecting an accuracy for a distance between the robot’s end-effector and the goal frame. A collision distance –field is for setting a minimum distance between the environment model and the robot. A collision model accuracy is a frequency for the collision model points. The model can be displayed by selecting a checkbox “show collision model”. Advanced iteration options include a landmark limit of the Explore-algorithm, an iteration limit for the Search algorithm, a factor for a Search function’s sustainability for a single landmark (iteration sustainability). A large number continues the iteration even if the Search does not progress significantly. A checkbox “iterate from both directions” iterates with a normal direction and a reversed direction (moves a last joint first) for each landmark.



robotselectorGUI

Environment options

carline.obj

Robot base

	Coordinate	Rotation
X	-500	0
Y	2000	0
Z	200	0

Room average temperature (C): 20

Max. noise (dB): 70

Task options

default.txt

Application: any

Goal frame

	Coordinate	Rotation
X	300	0
Y	2000	90
Z	500	0

Payload (kg): 1

Search options

ready

Goal error: 100

Collision distance (mm): 50

Collision model accuracy: 200

Advanced iteration options

Landmark limit per robot: 30

Iteration limit per landmark: 50

Iteration sustainability (1-100): 30

☐ Iterate from both directions

Search results:

Puma560
ABB_IRB140
Motoman_HP20
ABB_IRB7600

Figure 4.14 *Robot Selector GUI with carline-environment*

The current GUI appears to be understandable and easy to use even for new users. Several test-persons have tried without guidance.

4.4 Robotselection website

Website for the tools was created. The main requirements of the website were ability to share the tools, professional look and ease of use.

When defining the “professional look”, many websites of large brands was visited. Some notes were made that was common with look of the websites. The websites used large images and backgrounds. The amount of text was low and the main sites were simple, but it was easy to learn more. A navigation was also easy in the websites. Based on the findings, the professional look consists of the large background or front images, the simple pages with small amount of text and the easy navigation.

WordPress was used for creating the website. Free website plan was used with WordPress that caused a few limitations (limited customization and address must end with “wordpress.com”. WordPress websites automatically supports mobile devices. This feature saves a lot of work. The professional look was easy to create with editing pre-made themes. Theme “Radcliffe” was selected as a base of the website. The finished website can be seen in *Figure 4.15*. A video presentation was also created for these tools.

The finished website can be found at: http://robotselection.wordpress.com

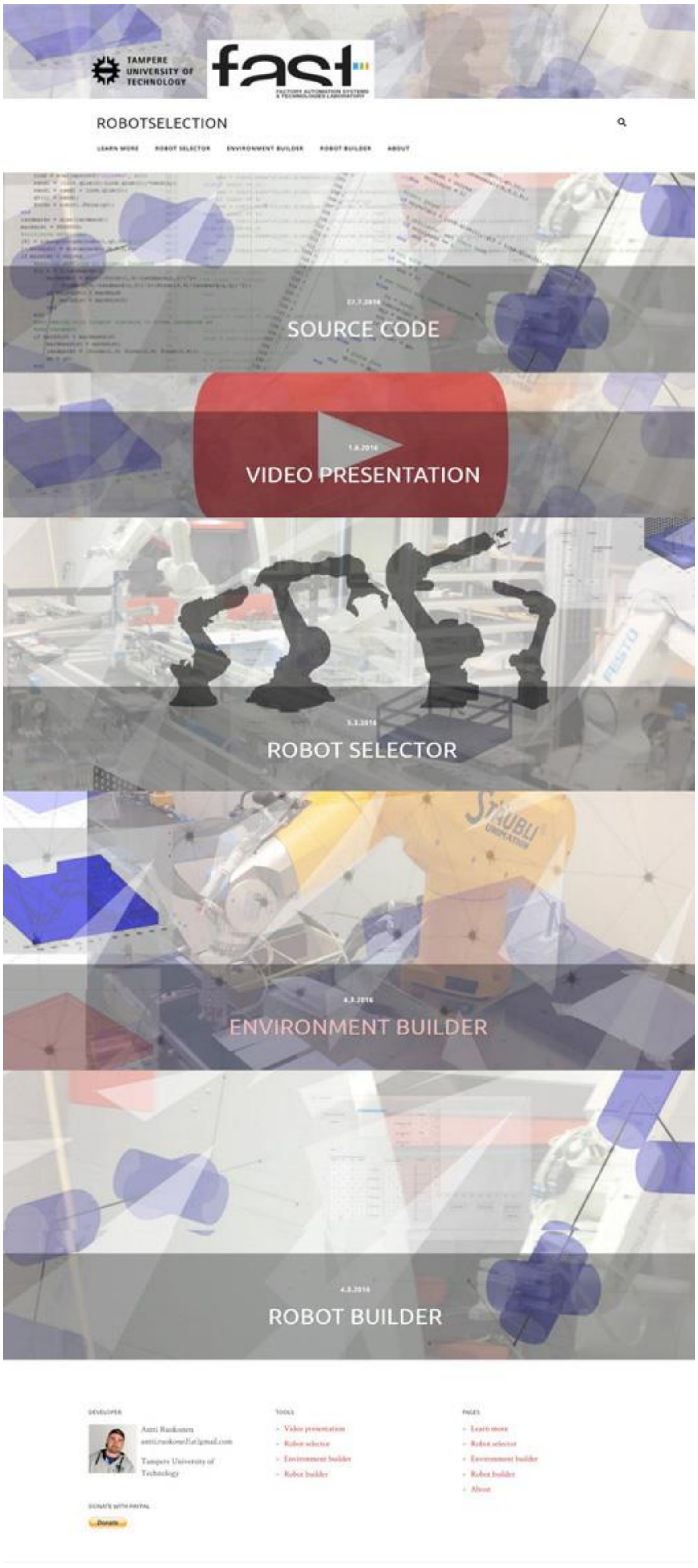


Figure 4.15 *A front page of the robotselection website*

Problems with a file sharing occurred since no exe-files was allowed to upload at the site. The problem was solved using Dropbox. A new dropbox-folder was created and shared to public. The tools were added to the dropbox-folder. By editing a link to the shared dropbox-file, an automatic download can be done. For an example, by editing “www.dropbox.com/...” to a following form “dl.dropboxusercontent.com/…” the direct download link is created. [39]

The website was then finished. Other problems did not occur. Creating the website was a straightforward process that was implemented in less than a week.

4.5 Distribution of the source code

Distribution of the source code was implemented via GitHub. GitHub is the largest source code host of the world.

Distributing the source code via GitHub allows many useful features for extending the code later. These features are for example bug tracking, feature request, task management and wikis of the code.

All the generated algorithms and the files are uploaded to GitHub https://github.com/rogasus/robotselector. Therefore, no code is attached to this document.

5. RESULTS

The main algorithms were tested with custom environments and robots. Complex and narrow environments were tested and goal frames was placed to tricky locations. Also different types of robots were tested: 6-DOF serial manipulators, SCARA-robots and even a PPP-robot. An experiment was also made with 100 samples.

The tools were also tested by modeling a real robot and robot environment at FAST-laboratory. User experiences were tested with three subjects and by observing while they use the tools.

5.1 Experiment

An experiment was made for testing the collision avoidance and the inverse kinematic solver. In the experiment, a random goal frame was generated in a robot’s free configuration space. Then the inverse kinematic -algorithm was executed and the results was saved in a MS Excel file. A testing environment is shown in *Figure 5.1*. The experiment included 100 random goal frames.

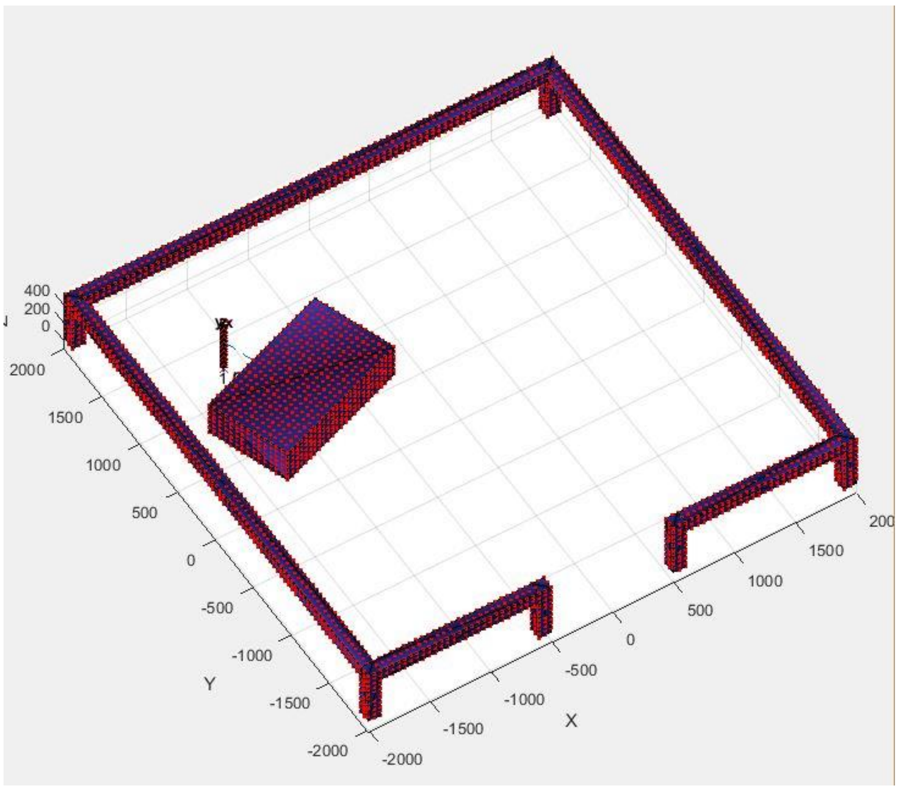


Figure 5.1 *Experiment environment*

The experiment was executed with two different devices (a powerful desktop computer and an average laptop). Statistics of the devices is shown in Table 5.1.

Table 5.1 *Experiment devices*

	Desktop PC	**Laptop PC**
Processor	Intel® i5-4670K 3.40 GHz	AMD A10-8700O Radeon R6, 10 Compute cores 4C+6G 1.80GHz
RAM	16 Gt	12 Gt
System type	64-bit operating system, x64-based processor	64-bit operating system, x64-based processor

After programming the experiment, the code was executed. The collision distance was set to 100mm. A landmark limit was set to 50 and a search iteration limit to 100. Table 5.2 shows the results with desktop PC.

Table 5.2 *Results of the experiment with desktop PC*

	Elapsed time (sec)	**Collision distance (mm)**	**Number of landmarks**	**Number of search iterations**
Average	3.88	221	1.16	2.48
Minimum value	1.43	82	1	1
Maximum value	21.47	270	4	13

The same experiment was made with the second device (laptop PC). The results with the laptop is shown below in Table 5.3.

Table 5.3 *Results of the experiment with laptop PC*

	Elapsed time (sec)	Collision distance (mm)	Number of landmarks	Number of search iterations
Average	11.5	221	1.19	2.53
Minimum value	4.46	111	1	1
Maximum value	58.3	270	4	13

The inverse kinematic solver found a solution in all 100 cases with both devices. Elapsed time was short especially compared to the original research result (34.5sec and 11.5sec). However, the original research was made in 1999 when CPU processing power was way lower. The algorithm found the solution mostly from the first landmark with average 2.5 search iterations. However, the collision distance was set to 100. With the desktop PC in 1 case out of 100, the robot violated the collision distance. A reason for this exception was found after checking the algorithm code and the code was fixed between the experiments. The experiment was not repeated with desktop PC because the difference of the results between the devices is only the elapsed time, which is not affected by the changes in the code. Otherwise the robot always avoids collisions and keeps the distance to the environment greater than the collision distance.

5.2 Results of collision avoidance

The collision avoidance was tested with several different ways. The experiment tested 100 random goal frames in the robot's free configurations space. The collisions were also empirically tested with different environment models and robots (5.3 Results of inverse kinematics). In order to test completely the collision avoidance, a collision model creation was also tested and analyzed with different environment files.

5.2.1 Results of collision model creation

The results of this point1 selection method were good. The points are set with given interval regularly. Several obj-files are tested again. The method works well also with long polygons. *Figure 5.2* shows two test files. Environment in left side was generated with MeshLab from a SolidWorks model. The long and narrow polygons are filled regularly. A shuttle (right) came with MS 3D Builder. It consists of large amount of small polygons. This method works well even in this case. However, the algorithm always places points to each corners of the polygon. This leads to dense point sets with small polygons (notice how pieces at the right side of the shuttle and tip of the shuttle are almost completely red for the points). However, in this case such details are unnecessary and therefore models with large amount of small polygons are not recommended for the Robot Selector.

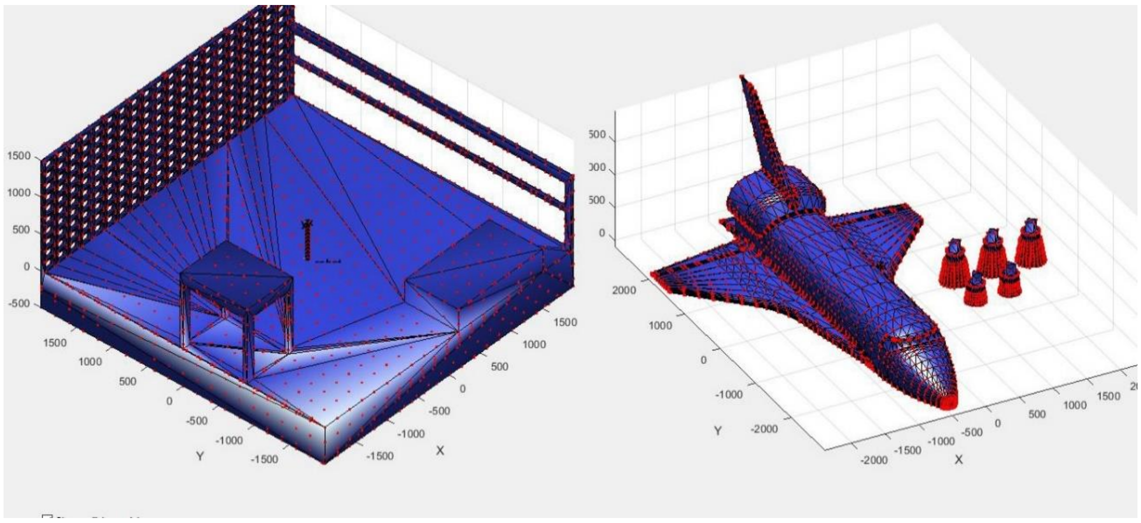


Figure 5.2 *Testing point1 selection method with wide polygons (left) and small polygons (right)*

With the previous method, the collision model points were created in some cases too frequently, which slow the program or too seldom to make the robot not to detect an obstacle. However, the current method works perfectly. Overall the collision model creation works with this method well.

5.2.2 Results of obstacle avoidance

The obstacle avoidance is tested with empirical way and with the test algorithm. The obstacle avoidance seems to be effective. The robots are not allowed to approach points in the collision model within the collision distance. Complicated objects were tested. Later in this chapter (5.3 Results of inverse kinematics) a wall with a hole -test is discussed. The robot avoided the obstacles while solving the inverse kinematic problem.

In rare cases, the algorithm violated the collision distance little bit (one search step). After checking the code, the error was found in a while-loop in the code. After adjusting the loop, the collision violation did not occur again (after 200 runs). We can assume that the problem is now fixed. The experiment was not repeated since with laptop PC (tested after fixing the error) there was no collision distance violation and only the processing time changes with the different devices.

Otherwise, the collision avoidance works well. In more than 99% of cases the robots avoided obstacles and after fixing the code, it never violated the collision distance limit again. Overall obstacle avoidance appears to work well.

5.3 Results of inverse kinematics

The inverse kinematic solver was tested with the experiment and with several custom environments. 6-DOF robots were tested as a default setting. Also prismatic joints were tested. The prismatic joint tests included a SCARA-robot and a PPP robot in a warehouse environment. The inverse kinematic solver was tested also with tricky tasks and environments including a wall with a hole -test and reaching under a table (*Figure 5.3*).

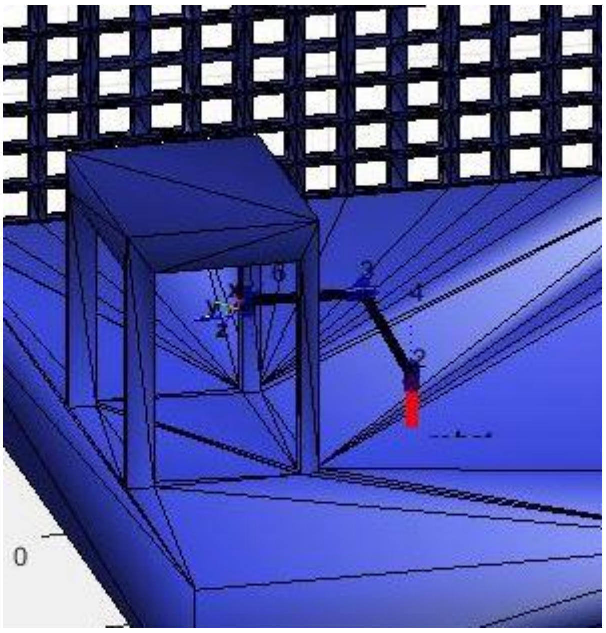


Figure 5.3 *Robot reaching under a table*

In most of the cases, the robot appears to find an inverse kinematic solution if one exists. In some cases, the robot was not able to find the solution because of tricky task location and a landmark maximum limit exceeded. With 100 as the landmark limit, the algorithm succeeded to find the solution almost every time. The landmark and search iteration limits appears to greatly correlate with robot's ability to solve the inverse kinematics in the complex environment.

The search function appears to be effective. With the warehouse-environment the Search in most cases found the solution from the first landmark for the PPP robot (3 prismatic joints). The Search function seems to succeed even with a very bad landmark as Explore's result. *Figure 5.4* shows one of the situations: 2 landmarks was used to find the solution. Green circles present an embryos of the latest landmark.

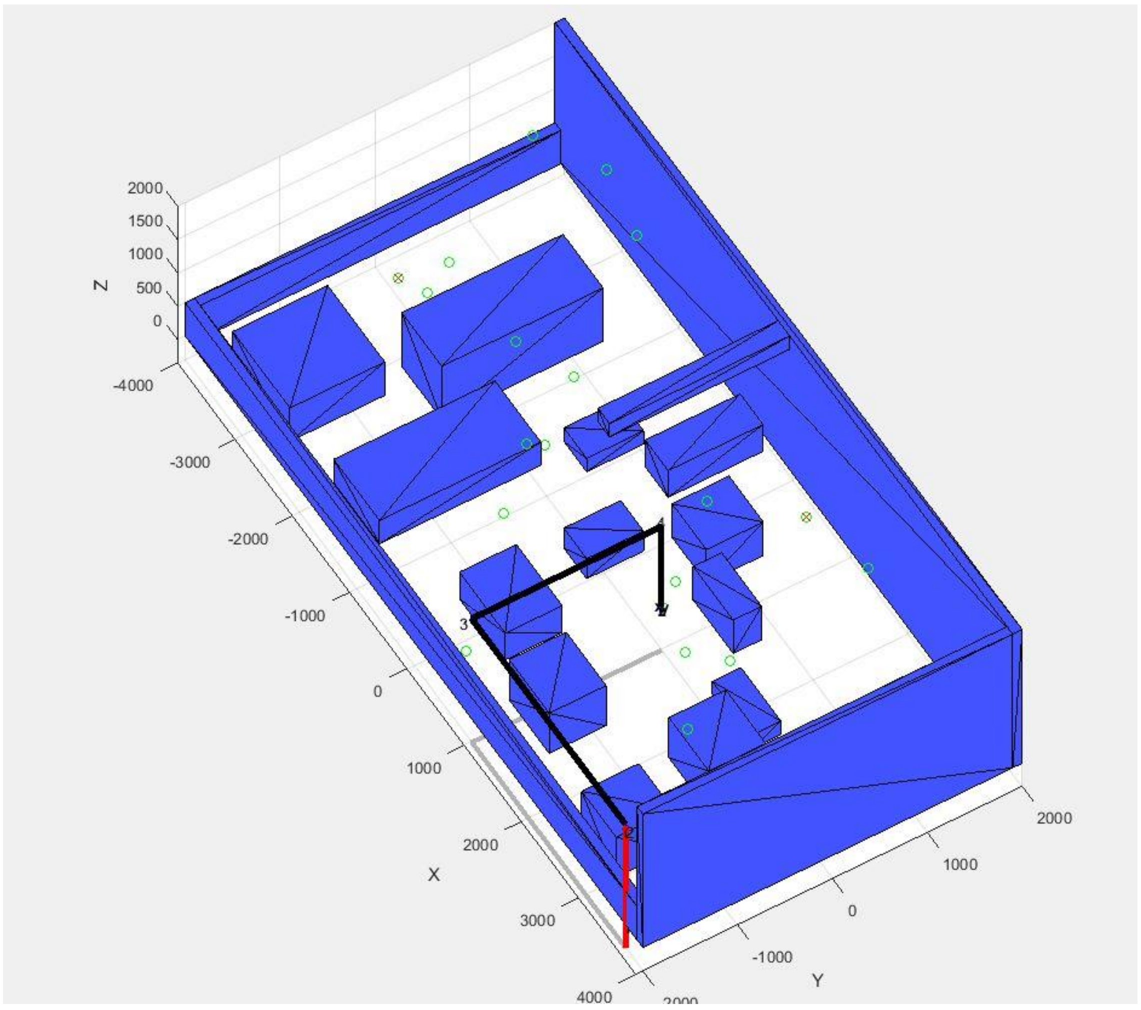


***Figure 5.4** Explore and search with PPP robot*

The Explore function can be tested with the tricky environments. The Search function mostly requires a line of sight to the goal. Explore function’s ability to place the landmarks in the narrow environment was tested especially with the wall with a hole -experiment. The Explore function seems to succeed even with these tricky and narrow environments. The wall with a hole -environment was created and tested (*Figure 5.5*).

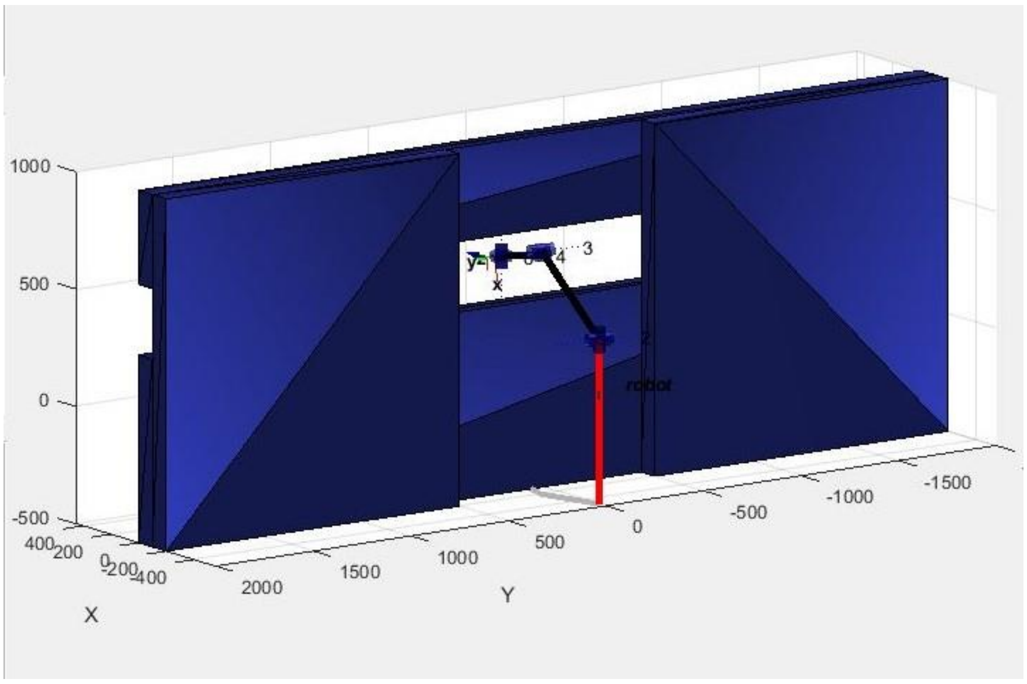


Figure 5.5 *Wall with a hole -environment*

Only few of the landmarks ended up another side of the wall (*Figure 5.6*). However, it was enough for the solution since the Search algorithm is effective and finds the solution from wide range in most cases.

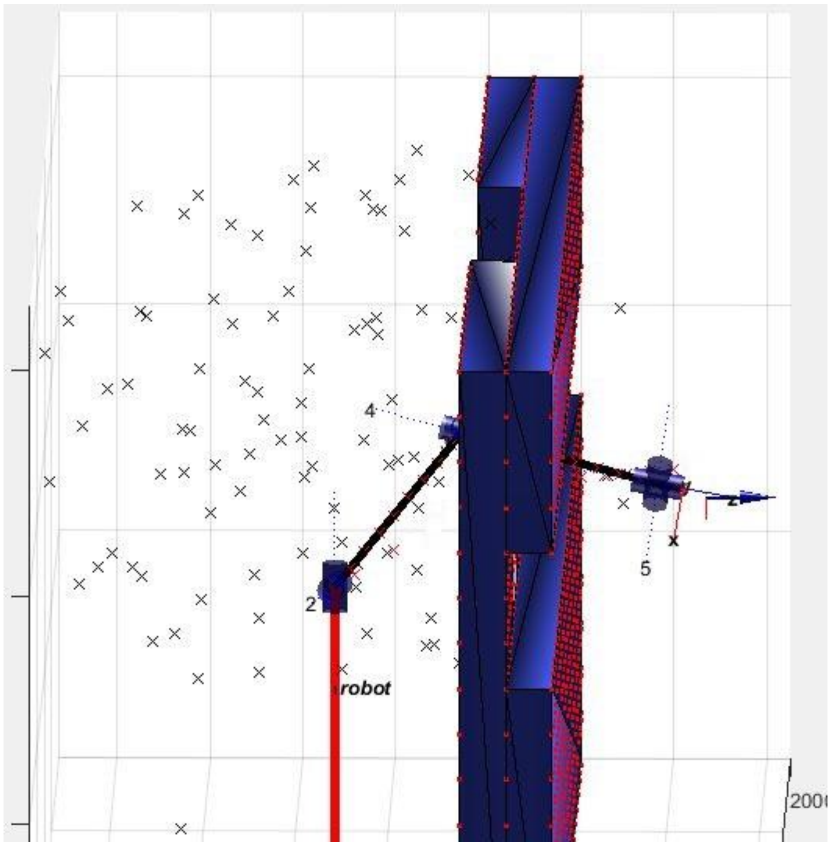


Figure 5.6 *Wall with a hole -obstacle in test*

Overall, the inverse kinematic solver appears to be effective and find the solution even in the narrow environment. Search is the most effective part of the tool. Even with a bad initial guess (landmark), the robot finds a way to the goal frame in common robot cell -type environments. With the narrow environments, it takes longer to find the solution since a larger part of embryos will be discarded because of collisions. Additionally, it is often required to increase the landmark limit with the tricky environments. In an ordinary environment, inverse kinematic problem is solved quickly with default settings.

5.4 Testing the tools with FAST-laboratory's robot

The tools were tested by modeling a robot environment and testing the robot selection algorithm in the real environment. The actual robot and the environment are located in FAST-laboratory (Factory Automation Systems and Technologies Laboratory), which is a laboratory by Department of Automation Science and Engineering in Tampere University of Technology.

Multiple robot environments were viewed. Mitsubishi RV-2AJ and its environment was selected for the modeling and testing. At the front of the environment, a corner of the table was selected as the origin of the environment. For next, the location and dimensions of the obstacles were measured with a ruler. The robot was modeled with Robot Builder utilizing drawings of the robot from Mitsubishi website. The environment was modeled with Environment Builder utilizing the self-made drawings. The modeling procedures followed the steps shown in *Figure 4.2* and *Figure 4.4*. *Figure 5.7* shows the selected robot environment and the environment model.

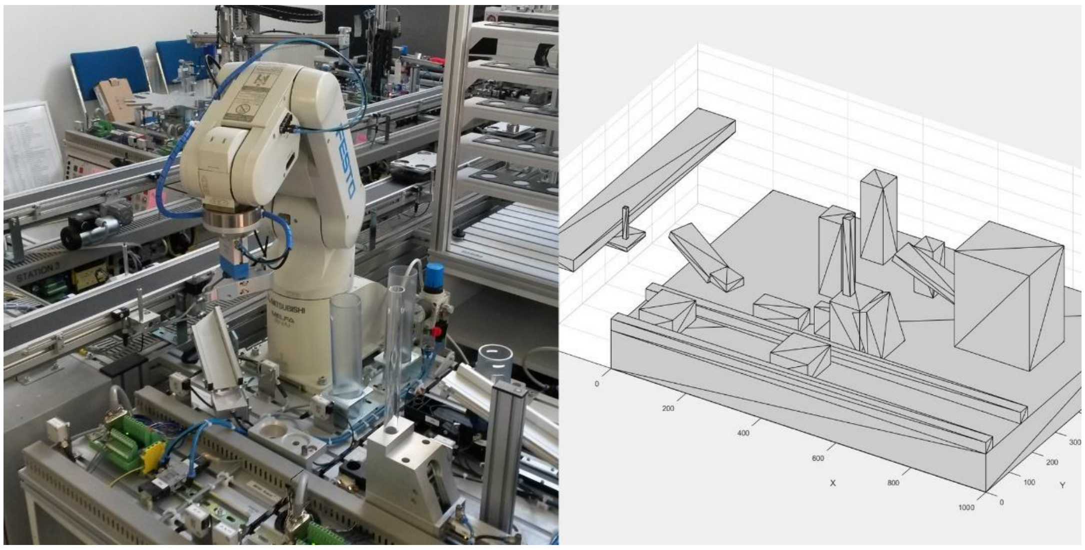


Figure 5.7 *Mitsubishi robot in FAST-lab (left) and the corresponding environment model (right)*

The Mitsubishi RV-2AJ model was tested with the environment model. Three goal frames were used in the test. The robot reached all the frames and was suitable for the task. *Figure 5.8* shows the Mitsubishi RV-2AJ robot in the environment.

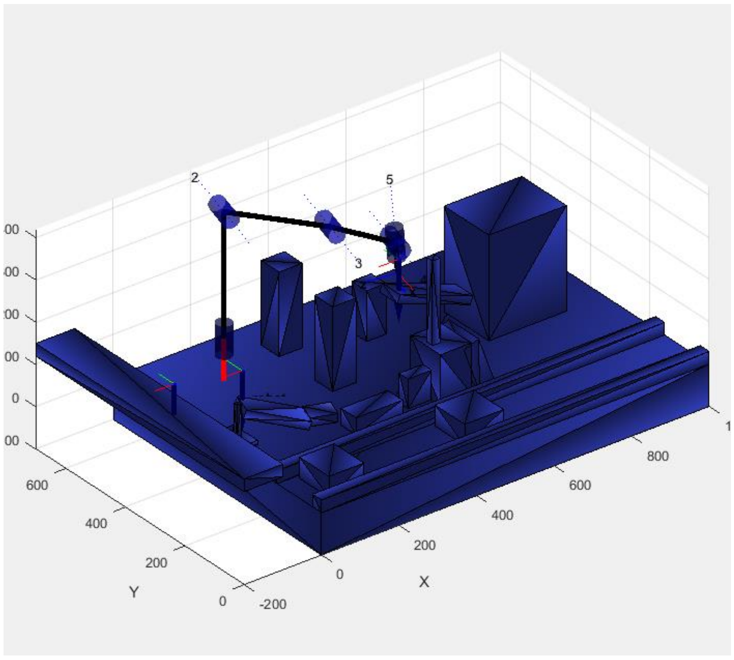
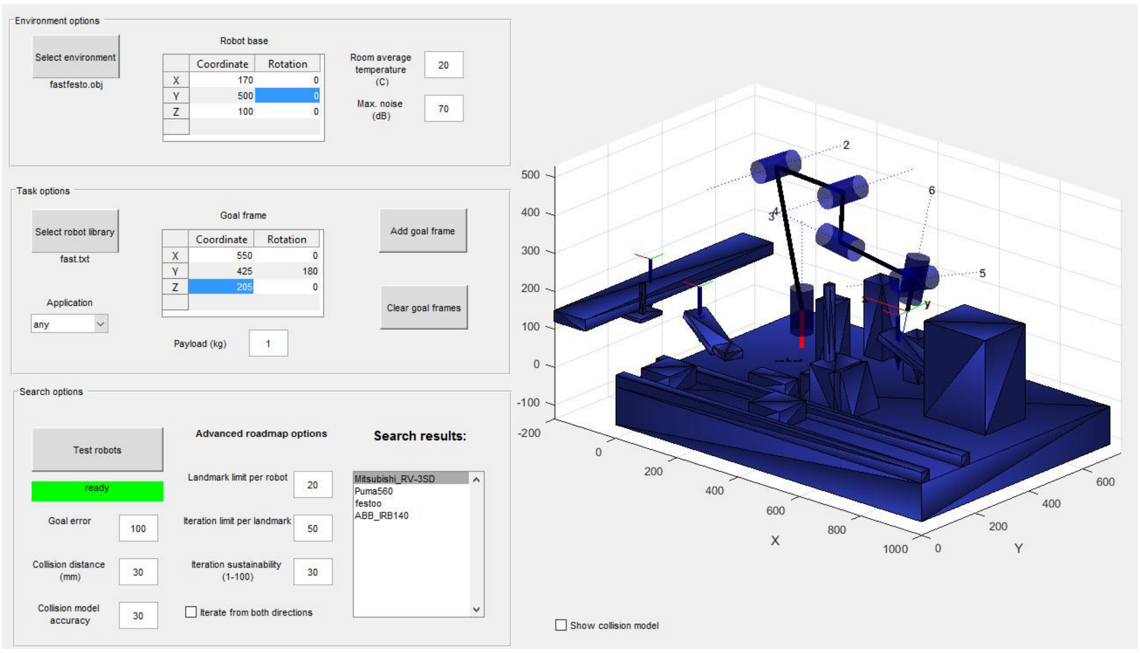


Figure 5.8 *Testing Mitsubishi RV-2AJ with the environment model*

The Mitsubishi RV-2AJ robot model was added to the robot library. Then the robot selection algorithm was started. All three goal frames were used in the robot selection. *Figure 5.9* shows the results of the robot selection process.



Environment options

fastfesto.obj

Robot base

	Coordinate	Rotation
X	170	0
Y	500	0
Z	100	0

Room average temperature (C): 20

Max. noise (dB): 70

Task options

fast.txt

Application: any

Goal frame

	Coordinate	Rotation
X	550	0
Y	425	180
Z	205	0

Payload (kg): 1

Search options

Goal error: 100

Collision distance (mm): 30

Collision model accuracy: 30

Advanced roadmap options

Landmark limit per robot: 20

Iteration limit per landmark: 50

Iteration sustainability (1-100): 30

☐ Iterate from both directions

Search results:

Mitsubishi_RV-3SD
Puma560
festoo
ABB_IRB140

Figure 5.9 *The results of the robot selection algorithm in the Mitsubishi RV-2AJ's environment*

Making the drawings of the environment took around half an hour. It took about one hour to find manuals of the robot and generate the DH table, model the robot and model the environment utilizing the drawings. The processing time of the Robot Selector took about 220 seconds with the desktop PC of Table 5.1. Four robots were found suitable for the task and the environment. Mitsubishi RV-2AJ was one of the suitable robots. The other robots of the results were Mitsubishi RV-3SD, Puma 560 and ABB IRB 140. In this case the robot was already selected. Otherwise, these four robots could be compared and one of them could be selected. The whole process took under two hours.

5.5 User experiences

The point of the user experience experiment was to find out a directive amount of time, that it takes to model robots, environment and find out the suitable robots in the custom environment. The usability of the user interface was reviewed and all other issues were searched.

The user experiences were tested by observing three subjects. All the subjects were 6th year students at Tampere University of Technology. One of the subjects has factory automation as the major. The subjects were given access to the tools for getting familiar with the interface without guidance. Notes were made while the subjects were using the tools. After the subjects were familiar with the tools, an assignment was given. The assignment was to model a simple environment shown in *Figure 4.3* with environment builder, Mitsubishi RV-2AJ with robot builder and add the model into default robot library. Then the subjects tested the robot in the modeled environment with Robot Selector. The task of the robot was to reach on the center of a conveyor. Dimensions of the environment and DH-table of the robot was given to the subjects.

Few issues were found for each tool that caused problems. The Environment Builder draws a preview shape before adding it. One of the subjects saved the model assuming that the preview shape is included in the model. With all the subjects, it took dozen minutes to understand how to model multiple shapes with the Environment Builder.

With Robot Builder the main problems occurred with robot joint limits and while saving the robot. One of the subjects did not save the robot after modeling. The subject assumed that the tool was saving the progress automatically after making changes. All the subjects tried to move joints of the robot before setting the joint limits.

With Robot Selector, the main problems were with placing the robot or the goal frame into illegal position. Two of the subjects wondered, why the tool does not let to add the goal frames. The reason was that the frames was within collision distance. For adding the goal frame, the frame must be moved further away from the obstacle or decreasing the collision distance. Similar problems occurred with placing the robot. All the subjects faced the problem with placing the robot too close to the environment. Reason for this was that the robot was mounted into a table causing a collision with the robot base and the table. This issue was fixed after the experiment. The tool now informs if the robot is in the collision distance.

After the subjects were familiar with the tools, the actual assignment was fluent. The time of the modeling and time of robot selection was measured with a stopwatch for all three tools. The average time of the environment modeling with Environment Builder was around 7 minutes. The average time of the robot modeling assignment with Robot Builder was around 5 minutes and the average time of the robot selection assignment with the Robot selector was around 4 minutes. The results of the experiment are shown on Table 5.4.

Table 5.4 *Results of the user experience experiment*

	Elapsed time with Environment Builder (min)	**Elapsed time with Robot Builder (min)**	**Elapsed time with Robot Selector (min)**
Subject 1	7:03	6:56	4:34
Subject 2	1:44	2:43	5:06
Subject 3	11:07	4:50	2:57
Average	6:38	4:49	4:12

After the user experience experiment, it can be deduced that the tools are usable after getting familiar with the tools. Getting familiar to all the tools takes around dozens of minutes. After users know how to use the tools, it takes several minutes to model environment or a robot. The subject with the robotics background (subject 2) performed considerably faster than the others. However, the Robot Selector have few issues with the robot and goal frame placing that caused the problems and slows the learning process, but it was made more clear after the experiment. The tools are not as user friendly as was expected, but the tools are usable after all.

6. DISCUSSION

The produced tools work for the given purpose. The main goal of the thesis is achieved and the tools work great for solving the robot selection problem. The inverse kinematic problem is solved within joint limits while avoiding obstacles. The results are good: The inverse kinematic solver succeeded to find a solution in every iteration and a collision distance is avoided in more than 99% of the situations in the experiment and it can be assumed that the one exception with the collision distance violation is now fixed.

The state of art got extended after this thesis. As it was said in the background chapter 2.3, engineers had only robot sizing tools, ranking methods and modeling tools to aid the robot selection problem. After the thesis, the new method was created for selecting industrial robots while considering the environment and the task of the robot.

Environment Builder is simple and accurate enough to model most of robot environments like robot cells. Other tools can be used to create the models. However, to keep the model simple enough, it is recommended to use small amount of large polygons instead of large amount of small polygons. As noted earlier with the shuttle-test, large amount (>10 000) of small polygons (polygon edge less than collision model accuracy) causes unnecessary computation and may slow the program. These details are not needed and therefore it is recommended to use Environment Builder or rough models. It takes only a dozen minutes to model the robot environment with Environment Builder.

Robot Builder is the only tool for creating robot libraries. Robot Builder utilizes the DH-parameter table, so basic knowledge of robot modeling is recommended (instead of using the tool with trial and error technique). When modeling the robot, drawings or at least dimensions of the robot is required. Unfortunately, all the robots must be modeled before testing. The default robot library contains only a few example robots. This limits the utilization of the tools. It is recommended at least in this phase that user models only already selected few robots and uses these tools to double check the robot’s suitability for the environment and the task. The ideal situation is that the user has multiple pre-modeled robot libraries for each robot types. It is not efficient to mix for example SCARA-robots and large 6-DOF manipulators in the same library to avoid unnecessary computation. The ideal situation contains the robot library for the different robot sizes.

Robot Selector is simply the tool for using the robot library and the environment model. Environment and task options are simple to use and understand. With these options, only “issue” left was with a robot mounting: If an environment floor is modeled (or any other plane robot is mounted to), the robot and the plane where robot is mounted to causes a collision. To avoid this, the tool simply does not allow to set the robot too close to the environment. The mounting plane can be left out of the model or the collision distance can be adjusted. Task options appears to be understandable according to my friends that tested the tool. On the other hand, the search options may require a knowledge of the algorithm. However, the settings were marked as “advanced iteration options”. These settings contain for example iteration limits and a factor that maintains the iteration even if it does not progress significantly. The default values were set for as versatile situations as possible. The values were selected with empirical manner. To get correct results, the user should be able to adjust the basic settings, like goal error, collision distance and collision model accuracy. To make the collision model accuracy more understandable to the user, the checkbox is added to display the collision model. Increasing the accuracy processing time also increases. The ideal settings have sufficient accuracy and decent progression duration. The tool is also designed to be as user friendly as possible. The status bar will always display the reason why the selection is not allowed to start. The tools were not as user friendly as was designed. However, the tools are usable after getting familiar with the tools.

The website became better than expected. The website works with mobile devices and is easy to navigate. Additionally, the website has professional look that was defined in Approach chapter. Problem with the website is the fact that users do not end up in the site. Even with good keywords does not result decent listing at Google search.

These tools can be used as tool for selecting the suitable industrial robots and especially for double-checking the selected robot’s suitability for the environment and the task. The main problem may be the fact that the user must model the robots by self, and it requires a knowledge of defining DH parameters. Pre-modeled robot libraries would help the robot selection process and make it faster and easier. The other disadvantage of these tools may be the MATLAB. Downloading unknown applications may be limitation for many users. The MATLAB Runtime is a quite large file. It would be way better solution for utilization if the program was made with C++. In my mind the ideal solution is that the tools can be used with only browser without downloading anything. However, this requires MATLAB server for computation. If this work is continued in any form, it would be suggested to make these tools as browser application.

Overall the research and the produced tools are successful. The tools are helpful solution even now for the robot selection problem. The reason for collision distance violation is now fixed. The work also has potential opportunities for further work.

7. CONCLUSIONS

The environment- and task-driven industrial robot selection methods were examined. It was found out, that there is currently no tool or method for selecting industrial robot taking into account simultaneously task and environment. All the tools or methods are made for robot ranking or sizing. Only way to take the environment into account is to simulate the task into robot programming environment or estimate robot's suitability.

To solve the environment- and task-driven industrial robot selection problem, a tool for selecting robots was created. In order to create the tool, the main research problems became inverse kinematics, collision avoidance and joint limit avoidance. Methods for solving these main problems were reviewed utilizing literature.

The inverse kinematic problem was solved within joint limits while avoiding obstacles. The solution algorithm was based to Juan Ahuactzin's and Kamal Gupta's successful research "The Kinematic Roadmap: A Motion Planning Based Global Approach for Inverse Kinematics of Redundant Robots". The solution utilizes Ariadne's Clew algorithm that consists of two sub-algorithms: Explore, which explores free configuration space of a robot, and Search, which tries to reach a goal frame utilizing cost-function.

The problem was solved using MATLAB and Robotic Toolbox by Peter Corke. As a result, three tools was created. The first tool is called Robot Selector, which is designed for testing the robots in custom environment with custom task requirements. Second tool is called Environment Builder. It allows user to create 3D-models of the robot environments and save it as Wavefront's OBJ-files for the Robot Selector. Third tool is Robot Builder, which is designed for creating custom robot libraries including models of custom robots. EXE-files was created for all these three tools to let anyone to use created tools without MATLAB. Then a website (robotselection.wordpress.com) was designed and created for distributing the tools. The source code was uploaded to GitHub.

Tools were tested with a quantitative experiment and empirically qualitative way. The experiment included 100 executions of the robot selection algorithm with 2 devices. With all of the cases, inverse kinematic solver found a solution. Collision avoidance worked with more than 99% of the cases. In 1 out of 200, a robot violated collision distance. However, the error that caused the collision got fixed after the experiment. Processing time of the algorithm was short (average 3.88 sec) with desktop PC, with laptop the average processing time was 11.5 sec. The tools were tested with empirical way by making challenging environments. The algorithm succeeded to find a solution when iteration limit was increased. Prismatic joints were also tested with a PPP-robot, that found goal frame mostly from the first iteration. A graphical user interfaces were tested by few of my friends without any instructions. The interfaces appeared to be usable after getting familiar with the tools.

The tools have few problems. It is hard to end up into the website even if user tries to search solutions for industrial robot selection problem. Other problem is small robot model library. This leads the user to model robots by self, which requires downloading my another tool (Robot Builder) and time to model the robots for Robot Selector. Third problem is that the tools are in EXE-format. All users do not want to download EXE-files from an unknown developer. The tools also require MATLAB runtime to run.

The project of this thesis leaves promising future work. The tools could be converted for example to C# or Java to reach better expandability. It would also be better solution to make these tools as a web browser application so the user does not have to download any files. The tools can be distributed also to other websites. The user interface can be improved, especially with Robot Builder, for example by creating a new interface, which allows the user to model robots without a knowledge of DH parameters.

This thesis solves the main problems with the kinematic roadmap -method. As a result, three tools were created for solving the environment- and task-driven robot selection problem also leaving promising opportunities for future work.

REFERENCES

[1] ABB Robot Selector, http://new.abb.com/products/robotics/robot-selector, [Accessed: August 2016]

[2] Ahuactzin Juan, Gupta Kamal. 1998. The Kinematic Roadmap: A Motion Planning Based Global Approach for Inverse Kinematics of Redundant Robots. IEEE. Transaction on Robotics & Automation

[3] Ahuactzin Juan, Gupta Kamal. 1999. Completeness Results for a Point-to-Point Inverse Kinematics Algorithm. IEEE. International Conference on Robotics & Automation

[4] Alec's Web Log. Website. http://www.alecjacobson.com/weblog/?p=917. [Accessed: August 2016]

[5] Alex Visser, Zengxi Pan, Stephen van Duin. 2015. Bounding Sphere CAD Model Simplification for Efficient Collision Detection in Offline Programming. IEEE. International Conference on Cyber Technology in Automation, Control, and Intelligent Systems

[6] Anthony A. Maciejewski, Charles A. Klein. 1985. Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments. International Journal of Robotics Research

[7] Ashkan M. Jasour, Mohammad Farrokhi. 2009. Path Tracking and Obstacle Avoidance for Redundant Robotic Arms Using Fuzzy NMPC. American Control Conference

[8] Auerl Fratu, Laurent Vermeiren, Antonie Dequidt. 2010. Using the redundant inverse kinematics system for collision avoidance. ISEEE. International Symposium on Electrical and Electronics Engineering

[9] Aurel Fratu, Jean-Francois Brethe, Mariana Fratu. 2010. Redundant inverse kinematics system for obstacles avoidance.

[10] Automation.com website. http://www.automation.com/library/articles-white-papers/robotics/industrial-robots-with-image-processing-in-the-photovoltaic-industry. [Accessed: August 2016]

[11] Bruno Siciliano, Oussama Khatib. 2008. Handbook of Robotics. Springer

[12] Cenit website. http://www.cenit.com/en_EN/plm/digital-factory/services/developments-for-catiadelmia-v5-and-v6.html. [Accessed: August 2016]

[13] Cornel Secara, Luigi Vladareanu. Iterative strategies for obstacle avoidance of a redundant manipulator. Wseas Transactions on Mathematics

[14] Dae-Huyng Park, Heiko Hoffmann, Peter Pastor, Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. Humanoids 2008 - 8th IEEE-RAS. International Conference on Humanoid Robots

[15] Euclidean space website. http://www.euclideanspace.com/threed/animation/collisiondetect/. [Accessed: August 2016]

[16] Farbod Fahimi. 2009. Autonomous Robots Modeling, Path Planning and Control. Springer

[17] GitHub website. http://www.github.com. [Accessed: August 2016]

[18] GUDEL robot sizing tool. http://www.gudel.com/service/productservice/robot-sizing-tool. [Accessed: August 2016]

[19] John J Craig. 2005. Introduction to Robotics Mechanics and Control. Third edition. Pearson Prentice Hall

[20] Kang, Teresa. 2000. Solving inverse kinematics constraint problems for highly articulated models. University of Waterloo

[21] Karl Mathia. 2010. Robotics for Electronics manufacturing. Cambridge

[22] Kuffner James, LaValle Steven. 2000. IEEE. RRT-Connect: An Efficient Approach to Single-Query Path Planning. International Conference on Robotics & Automation

[23] KUKA robotics website. http://www.kuka-robotics.com/usa/en/products/industrial_robots/low/kr6_2/start.htm. [Accessed: August 2016]

[24] Kun Qian, Xiaosong Yang, Jianjun Zhang. 2015. An Adaptive Spherical Collision Detection and Resolution Method for Deformable Object Simulation. 14th International Conference on Computer-Aided Design and Computer Graphics

[25] MathWorks. 2016. Website. http://se.mathworks.com/help/. [Accessed: August 2016]

[26] MathWorks. 2016. Website. http://se.mathworks.com/products/matlab/. [Accessed: August 2016]

[27] Mazer Emmanuel, Ahuactzin Juan, Bessière Pierre, Talbi El-Ghazali. 1993. The "Ariadne's Clew" Algorithm: Global Planning with Local Methods. IEEE/RSJ. International Conference on Intelligent Robots and Systems

[28] Mazer Emmanuel, Ahuactzin Juan, Bessière Pierre. 1998. The Ariadne's Clew Algorithm. Journal of Artificial Intelligence Research 9

[29] Mitsubishi robotics. Robot catalog. Website. http://mitsubishirobotics.com/pdf/MEAU_product_catalog.pdf. [Accessed: August 2016]

[30] P. Jimènez, F. Thomas, C. Torras. 2001. Computers & graphics 25. 3D collision detection: A Survey. Elsevier

[31] Peter Corke. 2016. Website. http://petercorke.com/Robotics_Toolbox.html. [Accessed: August 2016]

[32] Prasenjit Chatterjee, Viajay Athawale, Shankar Chakraborty. 2010. Selection of industrial robots using compromise ranking and outranking methods. Elsevier

[33] R. Muller-Cajar and R. Mukundan. 2007. Triangulation: A new algorithm for inverse kinematics. Proceedings of Image and Vision Computing

[34] R. V. Rao, B. K. Patel, M. Parnichkun. 2011. Industrial robot selection using a novel decision making method considering objective and subjective preferences. Elsevier

[35] R. Venkata Rao. 2007. Decision Making in the Manufacturing Environment. Springer

[36] S. Mitsi, K.-D. Bouzakis, G. Mansour. 1994. Optimization of robot links motion in inverse kinematics solution considering collision avoidance and joint limits. Mechanism and Machine Theory. Elsevier

[37] Shiqi Ou, Dehong Qui. 2009. A Novel Bounding Sphere Scheme to Compute Intersection of Subdivision Surfaces. CiSE. International Conference on Computational Intelligence and Software Engineering

[38] Tadej Bajd, Matjaz Mihelj, Marko Munih. 2013. Introduction to Robotics. Springer.

[39] TechApple website. Website. http://techapple.net/2014/04/trick-obtain-direct-download-links-dropbox-files-dropbox-direct-link-maker-tool-cloudlinker/. [Accessed: August 2016]

[40] Thomas R. Kurfess. 2005. Robotics and Automation Handbook. CRC Press

[41] Wei Zhao, Ru Wen. 2012. The Algorithm of Fast Collision Detection Based on Hybrid Bounding Box. ICCSEE. International Conference on Computer Science and Electronics Engineering

[42] Wikipedia article: WordPress. https://en.wikipedia.org/wiki/WordPress. [Accessed: August 2016]

[43] Wikipedia. 2016. Website. https://en.wikipedia.org/wiki/Wavefront_.obj_file. [Accessed: August 2016]

[44] WordPress homepage. www.wordpress.com. [Accessed: August 2016]

[45] Xiufen Ye, Le Huang, Lin Wang, Huiming Xing. 2015. An improved Algorithm for Triangle to Triangle Intersection Test. IEEE. International Conference on Information and Automation

APPENDIX 1: LINKS

Robotselection website:

https://robotselection.wordpress.com/

Source code:

https://github.com/rogasus/robotselector

Video presentation:

https://www.youtube.com/watch?v=8M0BSGviIRM