NTNU

Norwegian University of
Science and Technology

Optimal 3D Path Planning for a 9 DOF Robot Manipulator with Collision Avoidance

Kristoffer Aasland

Master of Science in Engineering Cybernetics
Submission date: June 2008
Supervisor: Alexey Pavlov, ITK
Co-supervisor: Neil Stembridge, ABB

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

Oil and gas (O&G) companies are continuously aiming to reduce both capital and operating costs of O&G installations, whilst maintaining an absolute focus on Health, Safety and Environment (HSE). As a means of achieving these goals, it is proposed to develop a fully automated offshore O&G installation, which can be operated from an onshore site. If a fully automated offshore installation concept is to prove feasible, the robot manipulator must be able to access all areas of the process equipment, such that relevant inspection and maintenance tasks can be conducted.

Previous project on this subject provided algorithms for path planning with collision avoidance for a simplified model of the robot manipulator in a simplified environment. In this project the emphasis should be on improving the efficiency of the proposed algorithms, achieving optimality of the planned paths and implementing the algorithms for a realistic model of the ABB robot manipulator for simulations as well performing experiments on a test rig either at ABB in Oslo. The main tasks of this project are outlined as follows:

- Create a realistic model of the 9 DOF ABB robot manipulator.
- Try out different methods for making a sample-based global path planner, in the specific case of the robot and environment used by ABB, to increase the chance of finding a feasible path.
- The planner should be optimized at the following areas: 1. High chance of finding a feasible path. 2. Short query time. 3. Short path
- Investigate the path planning problem for the case of given initial and final configurations and optionally for the case when only the position and orientation of the end effector are specified.
- Investigate the obtained results with simulations
- Implement the path planner and perform experiments on the test rig at ABB in Oslo.
- Document the obtained results.

Assignment given: 07.01.2008
Supervisor: Assistant Professor Alexey Pavlov, ITK NTNU
Co-supervisor: Dr. Neil Stembridge, ABB

Assignment given: 07. January 2008
Supervisor: Alexey Pavlov, ITK

Preface

This paper is the result of a 30 credits master thesis in the 10th term of the Master of Engineering Cybernetics education at the Norwegian University of Science and Technology (NTNU). In this paper an optimal 3D path planning scheme with collision avoidance for a robot manipulator is described. The optmail 3D path planning scheme is based on the well know Probabilistic RoadMap (PRM) method. It is optimal in the sense of greatest chance finding a feasible path, if it exsists, query time and path length. The effect of using a powerful local path planner based on the potential field method is also investigated.

I would like to thank my supervisor at NTNU, Alexey Pavlov, for guiding me through this project and help keeping me on the right track. Also thank to ABB for providing the task and connecting it to a practical application. And last, a great thank to my co-supervisor Neil Stembridge at ABB that has made a great effort working on the implementation of the path planner on the real system.

Kristoffer Aasland
28 May 2008

Abstract

This paper describes development of an optimal 3D path planner with collision avoidance for a 9 DOF robot manipulator. The application of the robot manipulator will be on an unmanned oil platform where it will be used for inspection. Most of the time the robot manipulator will follow a pre-programmed collision-free path specified by an operator. Situations where it is desirable to move the end effector from the current position to a new position without specifying the path in advance might occur. To make this possible a 3D path planner with collision avoidance is needed.

The path planner presented in this paper is based on the well known Probabilistic Roadmap method (PRM). One of the main challenges using the PRM is to make a roadmap covering the entire collision free Configuration space, $Cfree$, and connect it into one connected component. It is shown by empirical testing that using a combination of the Bridge Sampling technique and a simple Random sampling technique gives best $Coverage$ of the $Cfree$ space and highest $Connectivity$ in the roadmap for the given environment. An algorithm that increase the $Connectivity$ and sometimes provide $MaximalConnection$ is also described. A bacup procedure that can be executed on-line if a query fails is also presented. The backup procedure is slow, but it increases the chances of succeeding a query if the goal is in a difficult area. It is also investigated if the coverage and connectivity can be further improved by using the potential field planner when connecting the waypoints. Empirical testing showed that the improvements of $Coverage$ and $Connectivity$ were limited, and the sampling and query time increased.

The query time for a roadmap containing 400 nodes and one containing 1000 nodes was compared. It turned out that a large roadmap did not necessarily affect the query time negative because it made it easier to connect the start and goal nodes. Three existing path smoothing algorithms and a new algorithm, called Deterministic Shortcut, were implemented and tested. Empirical testing showed that the Deterministic Shortcut algorithm outperformed the others when it came to path smoothing versus time.

Contents

*

Chapter 1

Introduction

1.1 Motivation

In petroleum industry, as in all industry, there is always a demand of increasing efficiency, lower production costs and at the same time more focus on EHS, environment, health and safety. In addition to this the access to hydrocarbon reservoirs is getting harder and often lead in to very harsh environment, such as arctic areas and distant deserts. One way of facing these challenges is to make the oil platforms unmanned and control them from a remote control station. This would result in better EHS and decreasing costs due to less off shore manpower and considerably smaller and lighter platforms. Realization of unmanned oil platforms would make it economically reasonable to extract oil and gas from small reservoirs that is not economically reasonable with existing technology.

One of the main challenges on an unmanned platform is to be able to control and measure everything that's happening to be sure it works satisfactorily. One way of solving this challenge is to equip the platform with a lot of different sensors. But to be able to get a satisfactorily monitoring this would demand a enormous amount of sensors, which would be very expensive. A cheaper and more flexible solution is letting robots do the job. Using robots brings new challenges in to the light. One of them is to move the robot from one position on the platform to a completely different position on the platform without colliding it into any of the process equipment.

1.2 Optimal 3D Path Planning with collision avoidance

An extensive research has been done regarding path planning with collision avoidance for different kinds of robots, both in 2D and 3D environments. There exists many different methods that work well for robots with a low degree of freedom and in 2D environments. Problems for path planners with collision avoidance arise when dealing with robots with many degrees of freedom in 3D environments. Then a lot of the path planning methods fails or gets to complex to implement. Another factor that add the demand to the path planner is the complexity of environment and surrounding obstacles. For the robot manipulator and the environment used in this project, all the mentioned factors that complicates the task above is presented.

For some of the path planners developed earlier years the computationally cost and storage has been a problem, especially working with robot manipulators with many degrees of freedom. As known has the computer performance increased many times the last 10 years and storage has become incredible cheap. This has made it possible to develop very accurate path planners with collision avoidance that finds a collision free path for a many degree of freedom robot manipulators in a complex environment in a satisfactorily amount of time.

1.3 Project Scope

In this project is an optimal 3D path planner with collision avoidance for a 9 DOF robot manipulator developed. The path planner is optimized for a 9 DOF ABB Robot manipulator moving within a limited and static environment with processing equipment as complex obstacles. The requirement that the robot manipulator should be able to move between and close to the obstacles is emphasized. The path planner is optimized given the following criteria. Highest chance as possible of finding a feasible path when this exists. The feasible path should be as short as possible and be found as fast as possible.

1.4 Paper layout

This paper starts with explaining common terms used in robotics in Chapter 2. Chapter 3 summaries the knowledge gained from the literature study done in the project. Information about different existing path planners is

described, some of them in more detail than others. What adds the demand of the task in this project is the complexity of the robot system. A detailed description of the 9 DOF robot system and how it is modelled is therefore given in Chapter 4. The chapter also provide a sketch of the robot and its corresponding DH table. It also describes how the robot manipulator is modelled in Matlab. In Chapter 5 the main work done in this project is presented. It presents how an optimal path planner is developed for the given robot system and environment. The implementation and real experiments performed at ABBs test rig in Oslo is presented in Chapter 6. Implementations issues are described and how they are dealt with. Chapter 6 is then followed by a final discussion and conclusion in Chapter 7.

Chapter 2

Preliminaries

In this section a selection of basic terms are explained. They are necessary to be familiar with to fully understand the content of this paper. The terms described below is common terms in robotics and are based on the introduction chapter in [SHV06]

2.1 The Robot Manipulator

A robot manipulator consist of many links connected together. Between to links there is a joint which can either be prismatic or revolute. A revolute joint is like a hinge and allows rotation around one axis between two links. A prismatic joint allows linear motion between two links. The combination of links and joints forms a kinematic chain.

2.2 The Configuration Space

A robot manipulators pose is referred to as a configuration and gives a complete specification of the location of every point on the manipulator. The set of all configurations is known as the configuration space. The joint variables is used to describe a robot in configuration space. The joint variable for joint number i is q_i and is an angle, $q_i = \theta_i$, for a revolute joint and distance, $q_i = d_i$, for prismatic joint. A vector q then represent a configuration.

The number of "degrees of freedom", DOF, of a robot manipulator is given by its number of joints. To be able to reach a point in workspace in an arbitrary orientation a robot manipulator needs 6 DOF, 3 for positioning

and 3 for orientation. The number of DOFs is equal to the dimension of the configuration space. The robot used by ABB, an ABB IRB 2400/16, has 6 revolute joints and is connected to a gantry crane which has 3 prismatic joints. Hence the configuration space of the entire system has 9 dimensions. When a manipulator has more than 6 DOF it is called a redundant manipulator.

When dealing with collision avoidance the configuration space is often broken up in two parts. The configuration space of interest is then the C_free space which is the set of configurations where the manipulator do not intersects with any obstacles.

2.3 The Workspace

The workspace of a manipulator is the total volume swept out by the end effector as it executes all possible motions. The workspace is constrained by the manipulators joint limits and its geometry. The robot used by ABB is operating in a 3 dimensional works space.

The Denavit Hartenberg link frame convention (DH) and the Manipulator Jacobian are also two common terms in robotics. If the reader is not familiar with these terms please see Chapter 4 for details.

Chapter 3

Literature study

Path planning with collision avoidance treats the problem of finding a path for a robot, with one or many DOFs, in a static or dynamic environment. It is important to distinguish between path and trajectory. A path is a geometric description of robot motions and is made independent of time and has no dynamic aspects such s velocity or accelerations taken into account. One usually distinguishes between static and dynamic environments. In a static environment the location of obstacles is know in advance and can be modelled in some 3D modelling format. A dynamic environment changes through time and has mobile obstacles. In this case a priori knowledge of obstacles is often limited and requires use of sensors to achieve information about the surroundings. In this project path planning of a robot manipulator with collision detection in a static environment is considered.

During the last three decades extensive research has been done in the field of path planning with collision avoidance for robots in a static environment, from now on just referred to as path planning. This is due to growth in number of robot applications and greater challenges in the tasks accomplished. In addition, planning in many degrees of freedom is becoming more and more important in emerging applications, e.g. computer animations [GD98]. Another reason that has increased the research in path planning and made new methods functional in practice is increase of computer power and cheap storage.

The solutions to the path planning problem can be divided into to main methods, graph methods and potential field methods. The graph method can be further divided into two distinct areas, roadmap and cell decomposition approaches [MC94].

3.1 Potential Field Planner

In path planning, knowledge of the C_free is of great interest. Unfortunately, making an explicit representation of C_free is hard, very time consuming and often impossible. One path planning approach that does not need information about C_free in advance is the "Potential Field" method. The potential field method incrementally explores the configuration space while always making sure its inside the C_free space. This is done by setting up an artificial potential field, U, to guide the search.

The potential field is set up in a way that it attracts the manipulator to the goal and repulses it from the obstacles. The closer the robot manipulator gets to the obstacles the greater is the force repelling it. The repulsive force can be thought of the same force that occurs when two positive loaded magnets approaching each other. The force attracting the manipulator is greater the further away it is from the goal, and decreases as it approaches the final configuration, q_f . When the manipulator stops in its final configuration it has found a global minimum. The challenge of this method is to construct U such that only one global minimum exist and no local minima exists. This can be very difficult and sometimes impossible, which is a very big drawback of this method. Different extensions of the potential field method is developed that decrease the chance of getting stuck in local minima. Some of these extended methods are described in the sub sections below.

As described above the potential field U is the sum of the attractive field and the repulsive field

$$U(q) = U_{att}(q) + U_{rep}(q)$$

To be able to find the global minimum of U the problem can be treated as a optimization problem given a start configuration, q_s. One of the simplest method to solve the optimization problem is gradient descent. In this case the gradient of U can be considered as a generalized force acting on the robot in configuration space

$$\tau(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q)$$

in which τ is a vector of joint torques, if a revolute arm is considered. When this force or torque is acting on the robot it will follow the steepest descent toward its final configuration.

Calculating a potential field and finding its gradient in configuration space is quite difficult. The potential field and its gradients are therefore constructed in the workspace. All the attractive and repulsive forces acting on the links are found in workspace and then converted to torques acting on the joints

in the configuration spaces using the manipulator Jacobian matrix. When all the torques acting on one joint are found they are summarized together to find the final torque acting on that joint.

3.1.1 The Attractive field

The attractive field acts on each of the manipulator joints and will force the robot towards its final configuration. It should be calculated in such way that the further away from the goal a joint is, the greater the potential field is. An attractive potential field is calculated for each joint. Joint i is located at the origin of the i^{th} DH frame denoted by $o_i(q)$. An attractive field that fulfill the "increasing" requirement is a conic well potential which has a gradient with unit magnitude everywhere except of in the final configuration where it is zero. Unfortunately this means that it is not continuously differentiable and might lead to stability problems.

To solve this problem a function where the field increase quadratic with the distance from the goal is introduced. This field is called a parabolic well potential. One problem with this function is that it grows without bounds as q moves away from q_f which could give a very large initial force. A solution to this problem is to combine both the conic and the parabolic potential field. The parabolic potential attracts the manipulator when it is close to the goal, as shown in the first equation below, and the conic potential attracts when its far away. Such an attractive field could be defined by

$$U_{att,i}(q) = \begin{cases} \frac{1}{2}\zeta_i \parallel o_i(q) - o_i(q_f) \parallel^2, & \text{if } \parallel o_i(q) - o_i(q_f) \parallel \leq d \\ \\ d\zeta \parallel o_i(q) - o_i(q_f) \parallel - \frac{1}{2}\zeta_i d^2, & \text{if } \parallel o_i(q) - o_i(q_f) \parallel > d \end{cases} \tag{3.1}$$

where d is the distance that defines the transition from conic to parabolic well potential. ζ_i is a parameter used to scale the effect of the attractive potential and can be different for each joint. Typically a larger value is assigned to one of the joints to get some kind of "follow the leader" motion. The workspace attractive force is thus defined by

$$F_{att,i}(q) = \begin{cases} -\zeta_i(o_i(q) - o_i(q_f)), & \text{if } \parallel o_i(q) - o_i(q_f) \parallel \leq d \\ \\ -d\zeta \frac{(o_i(q) - o_i(q_f))}{\parallel o_i(q) - o_i(q_f) \parallel}, & \text{if } \parallel o_i(q) - o_i(q_f) \parallel > d \end{cases} \tag{3.2}$$

The gradient is well defined at the boundary of the two fields since at the boundary $d = \parallel o_i(q) - o_i(q_f) \parallel$ and the gradient of the quadratic potential is equal to the gradient of the conic potential $F_{att,i}(q) = -\zeta_i(o_i(q) - o_i(q_f))$.

3.1.2 The Repulsive field

The repulsive potential field is responsible for repelling the manipulator from the obstacles. It should be constructed in such way that when the manipulator approach an obstacle the repelling field increase to infinity such that the manipulator will never collide into an obstacle. In addition, when the manipulator is a certain distance from an obstacle it should not be affected by the repulsive field at all.

The attractive field was calculated for each of the joints on the manipulator. This is not sufficient for the repulsive field to avoid collision because other points on a link could be closer to an obstacle than the corresponding joint. Therefore for every link on the manipulator, closest points between the link and each obstacle in the workspace in a given radius need to be found. These closest points are then used to calculate the closest distance between a link and the obstacles. The alternating point on link i is called a floating point, $o_{float,i}$. One potential function that meets the criteria given above is

$$U_{rep,i}(q) = \begin{cases} \frac{1}{2}\eta_i(\frac{1}{\rho(o_{float,i}(q))} - \frac{1}{\rho_0})^2, & \text{if } \rho(o_{float,i}(q)) \leq \rho_0 \\ \\ 0 & \text{if } \rho(o_{float,i}(q)) > \rho_0 \end{cases} \tag{3.3}$$

where ρ_0 is the distance limit in which a link should be affected by an obstacle and $\rho(o_{float,i}(q))$ is the closest distance between a link and an obstacle. η_i is a parameter used to scale the effect of the repulsive potential. This value could vary for each obstacle. It could be small for obstacles near the goal to avoid the the repulsive field pushing the robot away from its final configuration. When this repulsive potential is given the negative gradient is determined as

$$F_{rep,i}(q) = \begin{cases} \eta_i(\frac{1}{\rho(o_{float,i}(q))} - \frac{1}{\rho_0})\frac{1}{\rho^2(o_{float,i}(q))}\nabla\rho(o_{float,i}(q)), & \text{if } \rho(o_{float,i}(q)) \leq \rho_0 \\ \\ 0 & \text{if } \rho(o_{float,i}(q)) > \rho_0 \end{cases} \tag{3.4}$$

in which $\nabla\rho(o_{float,i}(q))$ indicates the gradient evaluated at $o_{float,i}(q)$. If b is the point on the boundary of an obstacle that is closest to a point $0_{float,i}$ on link i then the distance between them is $\rho(o_{float,i}(q)) = \parallel o_{float,i}(q) - b \parallel$ and its gradient is

$$\nabla\rho(o_{float,i}(q)) = \frac{o_{float,i}(q) - b}{\parallel o_{float,i}(q) - b \parallel}$$

that is the unit vector directed from b toward $o_{float,i}(q)$.

3.1.3 Mapping Workspace Forces to Joint Torques

When the attractive, F_{att}, and repulsive, F_{rep}, artificial forces in workspace are found, they need to be mapped to artificial joint torques, τ in configuration space. This is done by using the manipulator Jacobian.

$$\tau = J_v^T F \tag{3.5}$$

where J_v includes the three top rows in the manipulator Jacobian. The three bottom rows are not used since workspace torqueses are not dealt with. When mapping the attractive forces the ordinary manipulator Jacobian corresponding to the DH table is used. When mapping the repulsive forces a dynamic manipulator Jacobian is used that corresponds to a DH table with varying origins. This has to be dynamic because of the $o_{float,i}$ that is changing due to where on the link the workspace force acts. It is very important to map the workspace forces to configuration torques before they are added together and not before.

3.1.4 Gradient Descent Planning

To solve the optimization problem the gradient descent algorithm can be used. This is a fairly simple version and other versions and extensions exists [SHV06]. The algorithm goes like this

Algorithm 1 Gradient Descent

1: $q^0 = q_s, i = 0$;
2: **while** $\| q^i - q^f \| > \epsilon$ **do**
3: $\quad q^{i+1} = q^i + \alpha^i \frac{\tau(q^i)}{\|\tau(q^i)\|}$
4: $\quad i = i + 1$;
5: **end while**
6: return$[q^0, q^1, ..., q^i]$

As seen from the algorithm it returns a vector with all the configurations from $q^0 to q^i = q_f$. Note that q^i is the i^th configuration in the loop and not the i^th component of the vector q. The α^i in the algorithm determines the step size of the i^th iteration. This value should be determined small enough such that the robot don't jump into any obstacles and large enough to making the algorithm efficient. i^th could be chosen on an ad-hoc or empirical basis. In optimization literature several systematics methods for

finding i^th can be found. The ϵ value is used because it is unlikely to find the exactly configuration as q_f. Therefore ϵ determine when the manipulator is close enough to let the algorithm terminate.

As mentioned the main disadvantage with the potential field planning method is the problem with local minima. One commonly used method to deal with the local minima problem is an extension of the algorithm above called randomized method. When the manipulator gets stuck in a local minimum the planner makes a random walk to escape the local minimum. Different methods for how to detect when the manipulator is stuck a local minimum and how to make the random walk to escape could be used, but are not dealt with in this paper because other methods are emphasized.

An intelligent choice of the tuning parameters ρ_0, ζ_i, η_i and especially α_i is crucial to avoid or reduce the local minima problem. For a robot with many DOF operating over large distances in a complex environment a perfect tuning of these parameters is very hard and often impossible. But, for small distances between two configurations the algorithm is more likely to avoid local minima and return a feasible solution. This motivates to try to combine the potential field method with the other path planners, such as the probabilistic roadmap planner described further below.

3.2 Cell Decomposition

A method that gives a representation of the free space is called exact cell decomposition. This method represents the free space by the union of simple regions called cells. The boundaries of the cells are often determined because of a physical reason. The reason might be a change of the closest obstacle or a change in line of sight to surroundings obstacle. If two cells have common boundary they are adjacent. To represents the combination adjacent cells an adjacent graph is used. The nodes correspond to cells and the edges between nodes connect adjacent cells. When the cell decomposition is done a path is found by first finding the cells that contain start and goal. When these cells are found a search through the adjacent graph is done to determine a feasible path.

Cell decomposition has the advantage that it can be used to achieve coverage. A coverage path planner determines a path where the robot or end effector moves through the entire free space by passing by all the cells. This feature would be useful if dealing with e.g. a floor cleaning robot or a painting manipulator. In this project such application is not considered which makes the advantage of the cell decomposition of less value.

There exist several methods for cell decomposition. One of the most popular is the Trapezoidal Decomposition which relies heavily on the polygonal representation of the planar configuration space. A more general method for decomposition is called the Morse Decomposition. This method allow for representations of non-polygonal and non-planar spaces. Methods that are based on visibility constraints are also often used.

The main drawback with cell decomposition methods is that they are not suitable for configuration spaces of higher order. When the configuration space is of higher order the cell composition methods get very complicated or even impossible to implement. Because of this, cell decomposition has not been studied in deeper detail in this project.

3.3 Probabilistic RoadMaps (PRM)

While the path planners mentioned in Section 3.2 use a pre-specified model of C_free the PRM planner use the fact that to check if a robot configuration is in C_free or not is a very cheap operation. PRM makes a lot of nodes/samples that describes random robot configurations, keeps the nodes not intersecting with any obstacles and discards the ones intersecting. The robot configurations are made by using a sampling technique in either a random selection or in a more deterministic way. Different methods have success in different environments [CLH05]. After this is done a roadmap is made by connecting the adjacent nodes using edges, which is the free path between the nodes. The edges are weighted using a so called "distance metric" which comes to use when the shortest path is found. Connecting of edges is done in a fine sampling to obtain a roadmap. All the operations mentioned above are part of the first phase, known as the learning phase, which is executed off-line.

The pseudocode for the learning phase is given in algorithm 2 where Δ represents the local planner using q and q' as input. It returns either a collision free path in C_free from q to q' or NIL if a path is not found. $dist$ is the distance function returning the distance between two given configuration. How the distance function is defined is presented in 4.5.

After the learning phase comes the query phase which is executed on-line. In this phase a path from the start to the goal is found in the following way. A given start node and goal node are attempt connected to the same connected component in the roadmap. If successful, then the shortest path is found between the start and goal by doing a search in the roadmap using some sort of a shortest path algorithm such as Dijkstra's algorithm or similar. An algorithm from [CLH05] describing the query phase is given in algorithm 3.

Algorithm 2 Roadmap Construction Algoritm

Input:

n: number of nodes to put in the roadmap

k: number of closest neighbors to examine for each configuration

Output:

A roadmap G = (V,E)

1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: **while** $\| V \| < n$ **do**
4: **repeat**
5: $q \leftarrow$ a random configuration in C
6: **until** q is collision free
7: V $\leftarrow$ V $\cup\{q\}$
8: **end while**
9: **for all** $q \in$ V **do**
10: $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
11: **for all** $q' \in N_q$ **do**
12: **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq NIL$ **then**
13: $E \leftarrow \cup\{(q, q')\}$
14: **end if**
15: **end for**
16: **end for**

Algorithm 3 Solve Query Algorithm

Input:

q_{init}: the initial configuration

q_{goal}: the goal configuration

k: the number of closest neighbors to examine for each configuration

G = (V,E): the roadmap computed by algorithm 2

Output:

A path from q_{init} to q_{goal} or failure

1: $N_{qinit} \leftarrow$ the k closest neigbors of q_{init} from V according to $dist$
2: $N_{qgoal} \leftarrow$ the k closest neigbors of q_{goal} from V according to $dist$
3: $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
4: set q' to be the closest neighbor of q_{init} in N_{qinit}
5: **repeat**
6: **if** $\Delta(q_{init}, q') \neq NIL$ **then**
7: $E \leftarrow (q_{init}, q') \cup E$
8: **else**
9: set q' to be the next closest neighbor of q_{init} in N_{qinit}
10: **end if**
11: **until** a connetion was succesful or the set N_{init} is empty
12: set q' to be the closest neighbor of q_{goal} in N_{qgoal}
13: **repeat**
14: **if** $\Delta(q_{goal}, q') \neq NIL$ **then**
15: $E \leftarrow (q_{goal}, q') \cup E$
16: **else**
17: set q' to be the next closest neighbor of q_{goal} in N_{qgoal}
18: **end if**
19: **until** a connetion was succesful or th set N_{qgoal} is empty
20: $P \leftarrow$ shortest path(q_{init}, q_{goal}, G)
21: **if** P is not emtpy **then**
22: **return** P
23: **else**
24: **return** FAILURE
25: **end if**

One of the greatest advantages of the PRM is that the main part of the computations is only done once and executed off-line, as long as the environment is static. The main part is the learning phase where the roadmap is made. The query phase is then executed on-line every time a new path planning query is specified. If the roadmap is to be used for several queries a great effort should be done to make a roadmap covering the entire C_free space with all the nodes connected in to one connected component, if possible. Another advantage is that sampled based planners can deal with many constraints such as closed-loop kinematics, stability constraints, energy constraints, contact constraints, visibility constraints and others[CLH05].

As described above the the PRM path planner consist of many sub algorithms. On of the main challenges, and most likely the most important one, is to build a roadmap consisting of only one connected component and that covers the entire C_{free} space. What combination of different sub algorithms that best provide this, depends on the features of the robot and what kind of environment it is supposed to be used in, [CLH05] and [RG06]. One combination might be very useful for one application but provide poor results for another application. A possibility is also to use several different methods in one sub algorithm to increase finding a feasible path in the graph. According to this it is hard to conclude which sub algorithms should be used without trying it on a detailed model of the real system. As a consequence of this the different sub algorithms in the Learning phase have been studied in closer detail in the following subsections. All the information in the following sub sections regarding Probabilistic Roadmap planners are provided from [CLH05] and [RG06] unless something else is mentioned

3.3.1 The Learning Phase

The process of constructing the roadmap can be divided in to several sub problems and to solve each sub problem there exists several optional methods:

- How to determine the density and distribution of configuration samples. e.g. Random, Grid, Halton, Cell-Cased, Bridge and Gaussian.
- Collision detection to determine if a configuration is feasible or not. E.g the GJK Separating axis algorithm or The Lin-Canny collision detection algorithm
- How to define a neighbor configuration. What method to use to calculate the distance between to configurations which can be used to determine if they are neighbors.

- Determine how to connect neighbors and which of them to connect. E.g. Component, Component-k, Visibility
- Choice of local planner to find paths between the node. Simple and fast local planner using interpolation or more powerful but slower planner, such as the potential field planner descibed earlier.

3.3.2 Sampling tecniques

The first step in developing a PRM planner is to make a lot of samples of the robot manipulator in different configurations in C_{free} space. The samples are stored as nodes in the roadmap. It is often desirable to get a reliable cover of the C_{free} space as possible while at the same time limit the number of nodes. Some reasons to limit the size of the roadmap are usually to reduce the storage space and save time in the learning phase. A small roadmap reduce the shortest distance search which will help reduce the query time.

Several sampling techniques are mentioned in the literature and they can be divided into two groups, uniform and non-uniform methods. The uniform methods are Random, Grid, Halton and Cell-based. The Random method simply choose the configuration parameters by random. In the Grid method the samples are distributed over a grid. The resolution is unknown in advanced and the grid is made by starting with a coarse grid. By halving each cell in the grid many times the resolution increase. Halton point set, which is used in discrepancy theory to obtain coverage of a region that is better than a grid, is also used with success to determine the samples. In the Cell-based approach random configurations are made inside decreasing cells in the workspace. First a random configuration is made. Then the workspace is divided into 2^3 sub cells. A new configuration is chosen inside every cell and then each cell is divided one more time and so on.

Non-uniform sampling methods are developed to add more samples around obstacles and in narrow passages. Some of these methods are Gaussian, Obstacle-based and Bridge Test. In the Gaussian sampling two random samples with a distance σ between them are considered at each time. Only if one of them is inside C_{free} and the other is outside C_{free} the free sample is added. The distance σ is determined by a Gaussian distribution. Obstacle-based sampling selects a sample by random. If it lie inside C_{free} its added to the graph. If not, a random direction is chosen and a step in that direction is made. It the new sample at the new point lie inside C_{free} it is added to the graph, if not the procedure continuous with increasing step size. The step initial size is chosen in advance. An extended version of this method is to discard a sample if it initially lie inside C_{free}. The Bridge Test selects two

random samples with a distance σ, chosen by Gaussian distribution, between them. Only if both samples lie outside C_{free} and if a third sample located equally between them lie inside C_{free}, the sample in C_{free} is added. While the first two methods intend to add more samples near obstacles the latter method aims to add more samples between obstacles, in narrow passages.

Two other non-uniform methods exist. One of them requires the closest distance to obstacles and the other requires in addition the penetration depth calculations. Since Robot Studio at ABB does not have any of these features the two sampling methods will have no priority in this project but are still worth mentioned. The first on, which is very expensive to use, is called Medial Axis and generate samples near the medial axis in C_{free}. All samples have at least two equidistant closest points on the obstacle. This results in a large clearance from the obstacles which help the roadmap to provide more robust paths. The other one is called Nearest Contact and generates samples on the boundary of C_{free}. If a random sample lie inside C_{free} it is discarded. If not, a new sample is found by moving in the direction of the penetration between the configuration and the obstacle. The configuration is then placed inside the boundary of C_{free}. It is important to place it at a certain distance from the boundary or else could the samples become very hard to connect.

In [RG06] a new version of the PRM, the Reachability Roadmap Method (RRM), is presented. This method is based on finding the reachability region of each node in the graph. It is shown that it generate one connected component covering the entire C_{free} space using few nodes. In addition it is very fast. A detailed description can be found in [RG06]. Calculating the reachability region for a node gets very complex for more than 3 dimensions and the RRM is therefore limited to 2 and 3 dimensional configuration spaces. Since the configuration space dealt with in this project has 9 dimensions the RRM is useless for this projects application.

In the different experiments done in [RG06] it turns out, not surprisingly, that the Bridge test performs well in environments with narrow passages, but moderately or even poor in less dense environments. In the environment in this project the robot will mainly operate around and between pipes and tanks. In the 9 dimensional configuration space some of those regions will probably appear as narrow passages, because many of the joints have very limited possibilities of motion in those regions. By this evaluation, both the Gaussian and Bridge sampling techniques seems to be good candidates to provide a reliable roadmap of the C_{free} space. Those two sampling tecniques are therefore chosen to be implemented, tested and investigated further in this project.

3.3.3 Collision Detection and Path Planning

To find out whether a configuration is inside the C_{free} space or not, collision detection is needed. Several algorithms doing this exist. In the project last fall the GJK distance calculation algorithm was implemented. A simpler version of this algorithm, The Separating Axis algorithm, does collision checking between two objects very quickly. This method is used for collision checking in the simulations done in this project. How the ordinary GJK algorithm works and the Separating Axis algorithm can be fond in [KAA07] or in [GB99].

A local path planner is needed to find out if there exists a collision free path between to nodes (configurations). When building a roadmap the nodes are often so close that a simple straight line path planner is sufficient to find a feasible path. In difficult areas where obstacles lie close or narrow passages has to be forced the demand of a more powerful, but also more expensive, path planner arise. The powerful path planner could be a potential field planner as described in Section 3.1. Below are some simple methods for testing if a path is collision free described.

Two typical methods that checks for collision along a path are presented. The Incremental or Interpolation method checks for collision by moving the manipulator incrementally along the path. For each step the configuration is checked for collision. Its important to determine the step size in a way that the entire path is known. In the Binary or Sub Division method the configuration at the middle of the path is checked for collision. If this is collision free both half sides is recursively tested. This is done until a collision occurs or the entire path is checked.

In practice it turns out that the Binary collision checking tend to perform better, [CLH05] and [RG06]. The reason to this is that the position in the middle is the one with greatest chance of intersect with the environment. This means that a the method detect a collision faster.

In the simulations in this project the Binary collision detection method is used as a local path planner. The local path planner is used to connect the nodes in the roadmap. It is deterministic and symmetric because the same path between two given nodes is found every time, independent of which of the two is the start node. In the experiments done on the real system at ABBs test rig, Robot Studio takes care of the collision checking and the local path planning. Robot Studio is more suited to do this because it has an accurate model of the robot and the processing equipment. The implementation on the real system is presented in detail in Chapter 6

3.3.4 Defining neighbors and connecting strategy

The nodes need to get connected to create a roadmap, or graph. In this part there are several aspects that should be taken into account to obtain a reliable graph in shortest time as possible. One aspect is how to determine suitable connection distance between to nodes. This distance should not be too long, nor to short. If the distance is to long the probability finding a collision free path will decrease and in addition the collision check for a long distance is expensive. If the distance is to short, way to many nodes are connected and thus many more samples is required.

Another important aspect is to determine how many connections should be tried. Trying too many connections could negative affect the running time while trying too few connections could result in a poor graph not connecting the entire C_{free} space.

The neighbor nodes are connected together in to a "connected component". It is desirable that the roadmap only consist of one connected component, but this is sometimes very hard to achieve or even impossible. A third aspect that has to be taken into account is how many connected components should a node endeavor to connect. In [RG06] some connection strategies are described. They are called Component, Component-k, Visibility, Visibility-k. In the Component strategy the nearest node in each component inside a given area are attempt connected. This is done because it is desirable to connect to several components. The Component-k strategy is very similar except that it is attempted to connect to the k-nearest nodes in each component. The k value is introduced because when there are a low number of components, an extra effort to connect a node is desirable.

The Visibility strategy is based on visibility sampling technique. The connections only connect to useful nodes. If a node fails to connect to any components it is labeled with u, for useful. The same accounts for a node that is connected to two or more components. If a node is connected to one component it is not labeled at all. It is observed that the number of useful nodes remains small, making it possible to try connect to all of them. The Visibility-k strategy is similar to the Visibility strategy, except of the k value which determines the number of nodes that should be considered for the usefulness-test, if they lay close enough.

Because the sampling time had very low priority for the application in this project, no explicit experiments were done to find an optimal connection strategy. From the experiments in [RG06] it turns out that the nearest-k node connection strategy performed well in most situations with k= ca 75. The maximum connection distance varied a lot regarding the environment and type of robot, but a long distance usually turned out to work better

than a short one. In fact it turned out that an infinity long distance did not perform much worse than the optimal value for each environment. Because of this the nearest-k with k=75 was chosen as connection strategy and the maximum connection distance was set to infinity.

Chapter 4

Robot Representation And Modeling

As mentioned earlier, when developing a path planner with collision avoidance the type of robot and environment it operates in are of great importance. A lot of the path planners comes to short when the dimensions of the configuration space and the complexity of the environment increase. Before a path planner could be developed, tested and simulated, models of the robot and the environment had to me made. The following chapter explains how the robot and the obstacles in the environment are modeled. Because the models were only supposed to be used for simulations, only geometric approximations were needed. What was more important was to take all the degrees of freedom of the robot in to account. Another issue that needed to be dealt with was how to determine the distance between two robot configurations. This is explained in the last sub section.

4.1 Forward Kinematics

In the unmanned oil platform research project by ABB and Statoil a 6 DOF robot is mounted up side down to a 3 DOF gantry crane. By this the entire robot system has 9 DOF. The 6 DOF robot is an ABB IRB 2400/16 robot. To keep track of the coordinate frame of each joint, forward kinematics and the Denavit-Hartenberg convention was used. The Denavit-Hartenberg table for the 9 DOF robot system is shown in Table 4.1. In the DH-table there are 10 joints. The first joint is a static joint and more like a "dummy joint". This joint was introduced because the base frame of the gantry crane is not located at the same position as the initial world frame in Matlab.

Joint	θ	d	a	α
1	$\frac{\pi}{2}$	34.30	13.50	$\frac{\pi}{2}$
2	$\frac{\pi}{2}$	d_x	0	$\frac{\pi}{2}$
3	$\frac{\pi}{2}$	d_y	0	$\frac{\pi}{2}$
4	0	d_z-14.30	0	π
5	$\theta_1^\star$	6.75	1.00	$-\frac{\pi}{2}$
6	$\theta_2^\star - \frac{\pi}{2}$	0	7.05	0
7	$\theta_3^\star$	0	1.35	$-\frac{\pi}{2}$
8	$\theta_4^\star$	7.55	0	$\frac{\pi}{2}$
9	$\theta_5^\star$	0	0	$-\frac{\pi}{2}$
10	$\theta_6^\star$	8.5	0	0

Table 4.1: DH table for the gantry crane and the ABB IRB 2400/16 robot

In Table 4.1 indicates $\theta^\star$ and d_x, d_y and d_z variables. By using the data in Table 4.1 the transformation matrix T_j^i between each joint can be calculated. T_j^i express the position and orientation of $o_j x_j y_j z_j$ with respect to $o_i x_i y_i z_i$, where o is the position of the origin and x, y, z gives the orientation of the basis vectors in each coordinate frame.

$$T_j^i = \begin{bmatrix} \cos\theta_j & -\sin\theta_j \cos\alpha_j & \sin\theta_j \sin\alpha_j & a_j \cos\theta_j \\ \sin\theta_j & \cos\theta_j \cos\alpha_j & -\cos\theta_j \sin\alpha_j & a_j \sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the four quantities $\theta_j,\ a_j,\ d_j,\ \alpha_j$ are parameters associated with the j^{th} joint or row in the DH table. The upper left 3x3 sub matrix in T_j^i is referred to as the rotation matrix R_j^i and the three first elements in the fourth column in T_j^i is the position of the frame origin o. The transformation matrix between two not adjacent frames is achieved by multiplying the adjacent transformation matrices in between. E.g. the transformation matrix from frame 0 to frame 4 is given by $T_4^0 = T_1^0 T_2^1 T_3^2 T_4^3$. Pleas note that a frame does not need to be located at the joint.

Figure 4.1 shows the coordinate frames and a sketch of the ABB IRB 2400/16 robot manipulator skeleton attached upside down to the gantry crane. The z axis is labeled and colored black and the red axis are the respective x-axis. In the bottom left corner is the initial world frame where all the other frames are related to. Figure 4.2 visualize how the coordinate frames are attached to the robot links for a given configuration. It is easy to see that the fourth z axis is the IRB robots initial frame. The DH-table, the sketch and the model of the robot have been made according to some data sheets provided by ABB. The data sheets are attached in Appendix A

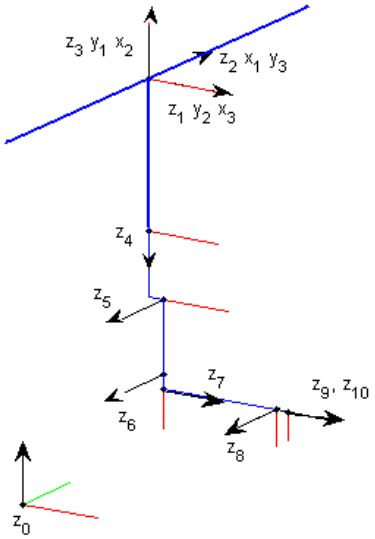


Figure 4.1: Skeleton of the ABB IRB 2400/16 mounted to the vertical bar on the cantry crane.

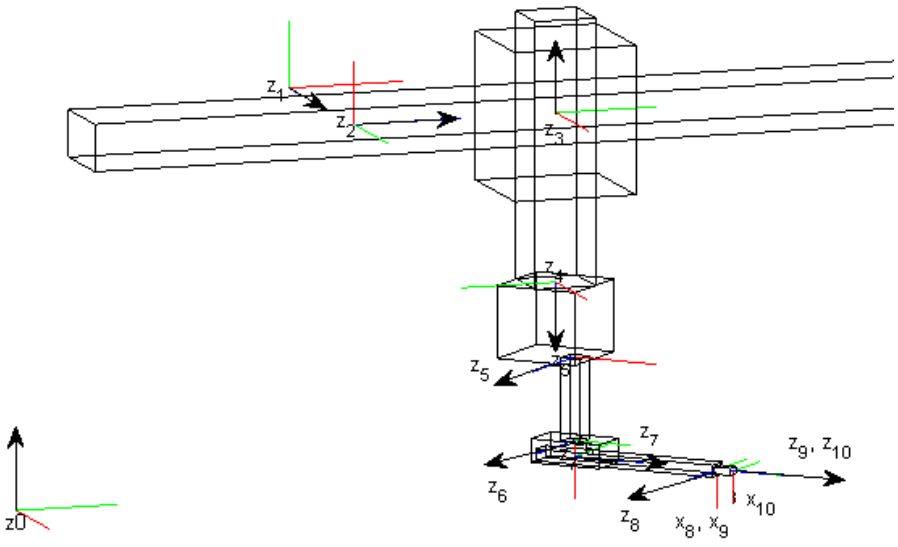


Figure 4.2: A skecth of the entire robot system and its respectve coordinate frames for a given configuration

4.2 Manipulator Jacobian

In order to map the forces acting on the links in workspace to joint torques in configuration space, the transposed manipulator Jacobian matrix J^T is used. The manipulator Jacobian is a mapping between the differential change of the joint variables and the differential change of the position and orientation of the end effector. The transposed manipulator Jacobian does the mapping in the other direction. The manipulator Jacobian from the base frame 0 to frame i is given by

$$J_i^0 = \begin{bmatrix} \mathbf{z}_0 \times (\mathbf{o}_i - \mathbf{o}_0) & \cdots & \mathbf{z}_{i-1} \times (\mathbf{o}_i - \mathbf{o}_{i-1}) \\ \mathbf{z}_0 & \cdots & \mathbf{z}_{i-1} \end{bmatrix}$$

where $\mathbf{o}_i$ is the origin of the i^{th} frame and $\mathbf{z}_i$ is the orientation of the $\mathbf{z}$ vector in the i^{th} frame. Thus, $\mathbf{z}_i$ is the first three elements of the third column in the transformation matrix T_i^0 and $\mathbf{o}_i$ is the first three elements of the fourth column in T_i^0. A manipulator Jacobian from the base frame to the end of each link had to be calculated and then transposed to map the force to torques. An algorithm taking care of this was implemented in matlab. Since no workspace torques but only workspace forces acting on the links is considered, the 3 last rows in the Jacobian was omitted.

4.3 Robot Modeling in Matlab

To be able to draw the robot ant the obstacles in Matlab, it was necessary to keep track of the position and orientation of each robot link and every part of the obstacles. This information was also used to do collision detection and to calculate the closest distances between the robot and the obstacles.

The robot manipulator links and the gantry crane links were modeled as several connected rectangular bounding boxes. These bounding boxes have their origins in the middle of each link. The transformation matrices provided from the DH-table gives the position and the orientation of each coordinate frame related to each joint and are not located in the middle of each link. To keep track of the bounding boxes of the links an extra set of transformation matrices, called "box transformation matrices", was calculated. The origin of each bounding box frame is located in the middle of the links, as seen in Figure 4.3. To calculate the bounding box frames the ordinary transformation matrices from the DH-table were used in relation with a matrix describing the measures of the bounding boxes. The bounding box frames were used by the GJK algorithm to calculate the closest distances

and to check for collision. They were also used to visualize the robot manipulator. Figure 4.3 shows the links bounding boxes and their coordinate frames.

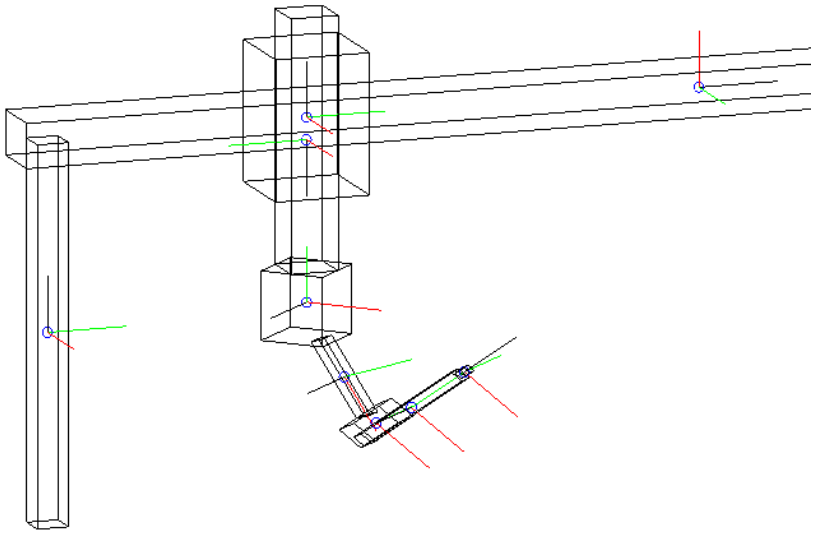


Figure 4.3: Bounding boxes and their coordinate frames

4.4 Modeling the Process Equipment

In the test rig at ABB in Oslo there is a contraption of different tanks connected together with several pipes. This is an imitation of typical processing equipment which can be found on oil platforms. The processing equipment is the obstacles that the robot has to avoid colliding into. Since the complexity of the obstacles is important when deciding on path planner approach, the processing equipment had to be taken into account. Unfortunately a detailed model of the processing equipment was not present in Matlab. Because of this an approximation of the processing equipment was made. The model that was made was also a contraption of tanks and pipes, just like the real model. This was done to add the same complexity to the model as for the real model.

The processing equipment where modeled as cylindrical bounding boxes. So called "obstacle transformation matrices" were calculated to determine the position and orientation of the obstacles frames. The "obstacle transformation matrices" were calculated by multiplying together the rotation matrices for each axis and by putting the translation vector in the fourth

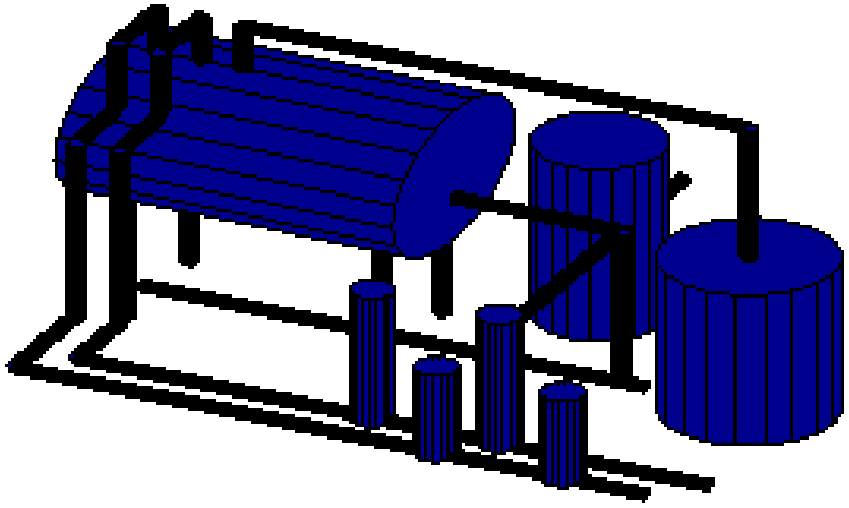


Figure 4.4: The processing equipment

column. The obstacles were independent objects and not connected by any kinematics chain. The rotation around each axis and position for each obstacle is given by a matrix $\mathbb{M}^{n\times 6}$ for n obstacles. The first three columns determines the Euler angle rotation around the x, y and z axis, respectively, and the three last columns determines the x, y and z position, respectively, for n origins provided by n obstacles. This matrix was used as input when calculating the obstacle frames.

In this project the modeling of the robot manipulator is limited to one bounding box per link. For implementation in a real world system one link should consist of many small boxes or spheres. This would give a more accurate modeling of the robot manipulator and increase the probability of obtaining the desired goal.

4.5 Distance Metric

The distance metric is a measure on how far two different robot configurations are from each other and should reflect the chance of collision between them. The longer distance the greater chance of collision. It is thus used to select which nodes one should attempt to connect using a local planner. Setting the distance metric is a non-trivial task and several different methods

exist [NA98]. The distance method used here is the same used in [RG06].

The distance metric between two configurations "q" and "r" is found by summarize the weighted partial distances between each of the DOFs $0 < i < n$ that describes the configurations, ie.:

$$d(q,r) = \sqrt{\sum_{i=0}^{n-1} [w_i d(q_., r_i)]^2} \tag{4.1}$$

How the calculation of the distance $d(q_i, r_i)$ is done depends on the joint:

- For translation: $d(q_i, r_i$ to $|q_i, r_i|$
- For rotation it is distingusihed between limited and periodic rotation. If the rotation is limited to less than 2π it is calculated in the same way as for translation. If the rotating joint can exceed 2π and is periodic the distance is set to $d(q_i, r_i) = min\left\{|q_i - r_i|, q_i - r_i + 2\pi, r_i - q_i + 2\pi\right\}$

The weight w_i in 4.1 describes how much influence each joint variable has to the final distance metric. A change in "theta one" has a bigger influence on the distance metric than a change in "theta six" because a greater part of the robot is moved and thus greater chance of collision. There is no straightforward method on how to set the weights on an articulated robot attached to a gantry crane. The weight for each joint was therefore set depending on how much of the robot manipulator moves when a joint variable is changed. As a measure the length from some of the joints to the end effector was used. For the 4th, 5th and sixth rotational joints the length from joint 5 to the end effector was used and for the 3rd rotational joint the length from joint 3 to the end effector. For the first and second rotational joints the length from joint 2 was used. For the three linear joints the length of the entire articulated robot was used. The weights are shown in the table below:

The weights in Table 4.2 were set according to linear values in meter and angular values in radians. This means that they had to be changed depending on what kind of input values used. In the Matlab simulations the linear input was made in decimeter, which meant that the linear weights had to be multiplied by 0.1 because of the 1:10 ratio. When it comes to the experiments on the real system the linear weights had to be multiplied with 0.001 because the input was given in millimeters. The angular weights had to be multiplied with $\frac{pi}{180}$ because the input was given in degrees.

Joint:	Weight
"Dummy joint"	0.0
Linear joint 1, x	2.16
Linear joint 2, y	2.16
Linear joint 3, z	2.16
Rotational joint 1, θ_1	1.54
Rotational joint 2, θ_2	1.54
Rotational joint 3, θ_3	0.84
Rotational joint 4, θ_4	0.085
Rotational joint 5, θ_5	0.085
Rotational joint 6, θ_6	0.085

Table 4.2: Weights for each joint in the robot system

Chapter 5

Optimal 3D path planner

5.1 Optimization criteria

Before starting to develop a path planner based on the PRM method, it is important to determine what kind of application the path planner is intended for. Different path planners can be optimized different, depending on what features and optimization criteria are emphasized for the given application.

The running time of the learning phase, the time it takes to make an adequate roadmap, is of great interest when the PRM is used in engineering. An example could be an engineer who wants to find out if a part can be removed from a car engine for maintenance without disassembly the entire engine. Off course the engineer does not want to wait longer than necessary to get the answer.

Reducing the number of nodes and the size of the roadmap is of great interested when the PRM is used in computer games and real-time applications. A small roadmap requires less memory which is always a goal when making computer games. Another important benefit is that a small roadmap reduces the query time which is important to not make the game halt.

In the application for this project some of these measures might be of less interest. The roadmap is not supposed to be made over and over again. Once it is implemented it should, hopefully, stay the same for a long time. This means that it does not matter if the learning phase takes 5 seconds or 5 hours. When it comes to memory this should not be a problem in practice as long as the roadmap does not become extremely large. But, the size of the roadmap might affect the query time, which is why it might be of interest. If one is dealing with query time in milliseconds it does not matter if it takes 5 or 60 milliseconds to find a path. But as soon as the query time increase

and one is dealing with seconds, or even minutes, it suddenly becomes of great importance for this application as well.

Priority number one for this projects application is to be able to move the end effector from one position to every other possible position on the platform. This is fulfilled by obtaining *Coverage* and $MaximalConnection$ for the graph G = (V, E) as defined in [RG06]:

Definition 5.1 (Coverage)**.** *G covers C_{free} when each configuration $c \in C_{free}$ can be connected using a local planner to at least on node $v \in V$.*

Definition 5.2 (Maximal Connection)**.** *G is maximally connected when for all nodes v',v" $\in$ V, if there exists a path in C_{free} between v' and v", then there exists a path in G between v' and v".*

The *Coverage* criterion ensures that it is possible to connect every possible start and goal configuration in C_free to at least one node in the roadmap. The $Maximal\ Connection$ criterion ensures that if there exists a path between a start and goal configuration in C_free, there should also exist a path in the roadmap G. *Connectivity* is a term used to explain how far or close a roadmap is to have $Maximal\ Connection$.

There exist certain methods to exact measure *Coverage* and *Connectivity*. Unfortunately they only deal with configuration spaces up to 3 dimensions [RG06]. How coverage and connectivity are measured in this project is described in 5.2. From now on the two criteria are referred together as the *Reachability* criterion.

According to the reasons listed above, the optimization criteria for the application in this project are given the following priority:

1. Reachability
2. Query time
3. Path length

5.2 Test set up

The sampling techniques used in the Probabilistic Roadmap method depends on configurations made at random. Because of this it is very hard, or even impossible, to derive analytical proofs showing which sampling method is optimal. And as mentioned earlier, different sampling techniques might be optimal for different robot and environments. To be able to decide on an

optimal sampling technique for ABBs application, empiric experiments for different sampling techniques were done. When deciding on an optimal sampling technique it is important to have an accurate model of the robot and the environment. Unfortunately this was not present so the approximated models described in Chapter 4 were used.

As mentioned in section 5.1 it is by now not known how to exact measure the *Coverage* and *Connectivity* for a 9 dimensional configuration space. Because of this, an alternative approach was used: 8 test configurations were made and for each roadmap generated the possibility of finding feasible paths between them was investigated. The 8 test configurations were made by visual inspection of the processing equipment in the environment. Some test configurations were meant to be easy to connect to the roadmap and others, located close and between many obstacles, where meant to be very hard to connect. When a roadmap was generated the 8 test configuration were attempted connected to the roadmap. The *Coverage* was measured by how many attempts succeeded. When the test configurations were connected to the roadmap, feasible paths between them were attempt found. The maximum of possible paths (successfully queries) were 7+6+5+4+3+2+1 = 28. The number of feasible paths found was a used as a measure of the *Connectivity*. The 8 test configurations used are shown in Figure 5.1. For simplicity only the 4 last links of the robot manipulator is diplayed.

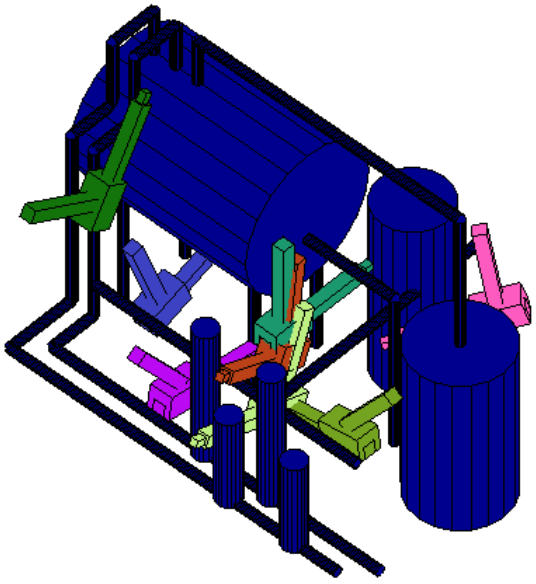


Figure 5.1: The 8 different test configurations used. The light green is test configuration nr 3 that turned out to be very difficult to cover.

All the experiments and simulations presented in this paper were executed

on a Intel(R) Pentium(R) D CPU 3.00 GHz 2.99 GHz with 2.00 GB of RAM.

5.3 Learning phase

As mentioned above there exists many different sampling techniques. In this part some of them are studied in more detail, implemented and tested. In the end they are all compared against each other. A method on how to improve the *Connectivity* of a roadmap is also described, implemented and test in the last subsection.

5.3.1 Gaussian Sampling

A sampling method mentioned in section 3.3.2 that aims to make samples close to the obstacles is the Gaussian Sampling technique. The sampling technique makes first a random sample. If it lies in C_{forb} a new sample is made a certain distance σ from the first sample. If the new sample lies in C_{free} it is stored in the roadmap. The distance σ between the samples is chosen according to a Gaussian distribution. The method is explained in detail in [VB99]. The pseudocode for the Gaussian Sampling technique is presented in Algorithm 4.

Algorithm 4 Gaussian Sampling Algorithm

```
1: repeat
2:    q ← a random configuration in C
3:    if q is NOT collision free then
4:       q' ← is a random configuration at length σ from q.
5:       if q' is collision free then
6:          Add q' as a node in the roadmap
7:       end if
8:    end if
9: until Desired number of nodes created
```

At line 4 in Algorithm 4, q' is chosen to be a random configuration at a given length from q. To determine a configuration at a given length from an initial configuration, the random direction vector from [RG06] was used. The random direction vector q_{rand} is made such that the distance between q and $q + q_{rand}$ is σ:

$$q_{rand} = \pm \frac{rnd_i * \sigma}{\sqrt{rnd \cdot w}} \tag{5.1}$$

where rnd is a vector of random values between 0 to 1 such that $\sum_i^{n-1} rnd_i = 1$. rnd_i is a random value for each DOF i. w is the weight vector described in section 4.5 and $rnd \cdot w = sum_{j=0}^{n-1}(rnd_j * w_j)^2$.

How to determine the distance σ, which describes how close to obstacels nodes are wanted, is non-trivial for a 9 DOF system. Experiments with different values of σ were done to determine the optimal distance. At first an experiment with great spread of test values used was executed. This gave some idea of what range the optimal value could be found in. In the first experiment $\sigma = [4\ 2\ 1\ 0,5\ 0,1]$ was used. This experiment showed that a very small σ, σ=0.1, gave best coverage. Even if $\sigma = 0.1$ gave best coverage, it resulted in many more connected components and therefore less successfully queries than for greater values. Not very surprisingly the sampling time increased with decreasing value of σ. These results corresponds to the fact that a small value of σ will make samples closer to obstacles which makes it more demanding to find a valid sample and to connect it to other nodes. Results from the first test is shown in Figure 5.2 and Figure 5.3. Please note that the maximum possible *Coverage* is 8 and maximum possible queries is 28.

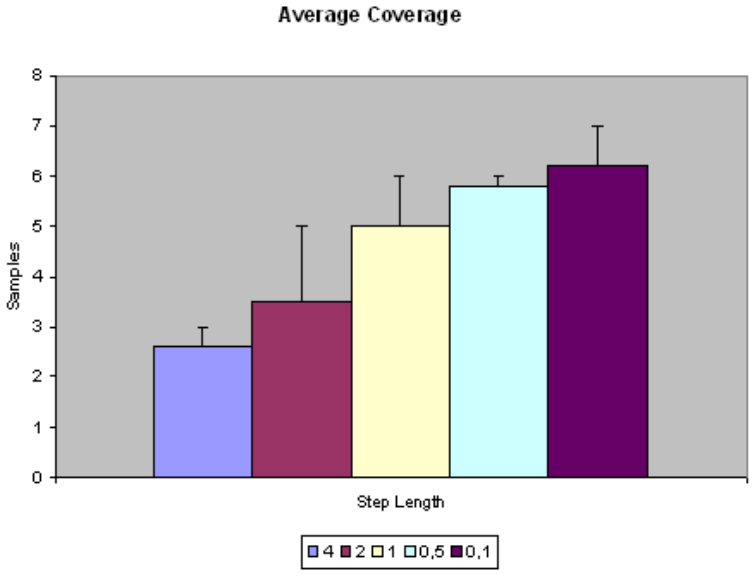


Figure 5.2: Average *Coverage* using different values of σ

Because *Coverage* was of highest priority when finding an optimal value of σ, new experiments were executed using values close to 0.1. The results from these experiments is shown in Figure 5.4 and Figure 5.5. Figure 5.2 shows that step length 0,15 gives best average *Coverage* with 6 samples and

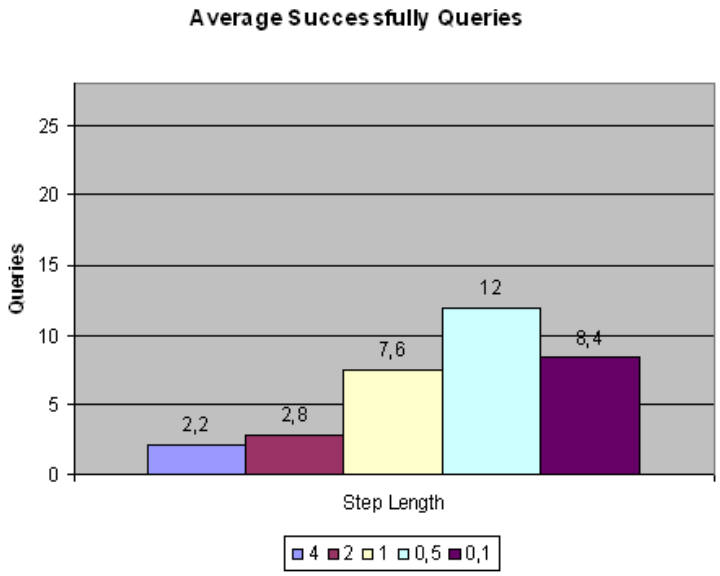


Figure 5.3: Average successfully queries using different values of σ

step length 0,1 gives second best average *Coverage* with 5,8 samples
But Figure 5.2 shows that step length 0,1 gives an average *Coverage* of 6,2 samples. The reason to these variations in the reults is due to the randomness of the method used, especially provided by the random direction vector 5.1. It would probably be more beneficial to run even more test for each step length to get a more reliable result. But because of limited time this has not been done. The maximal *Coverage* obtained when using step length 0,15 was 6, while the maximum *Coverage* when using step length 0,1 was 7. Because of this 0,1 was chosen as the optimal step length σ. Its worth mentioned that when step length was set to 0,01 the sampling time increased by a factor of 10 compared to the others. Despite this it did not improve the *Coverage*.

5.3.2 Bridge test

Another sampling technique mentioned in section 3.3.2 is the Bridge Sampling technique. This sampling technique aims to make samples between obstacles. At first the sampling technique makes a random sample. If it lies in C_{forb} a new sample is made a certain distance σ from the first sample. The distance σ between the samples is chosen according to a Gaussian distribution as in the Gaussian Sampling technique. If the new sample also lies in C_{forb} a third sample is made. The third sample is made between the two first samples. If this sample lies in C_{free} it is added to the roadmap. A

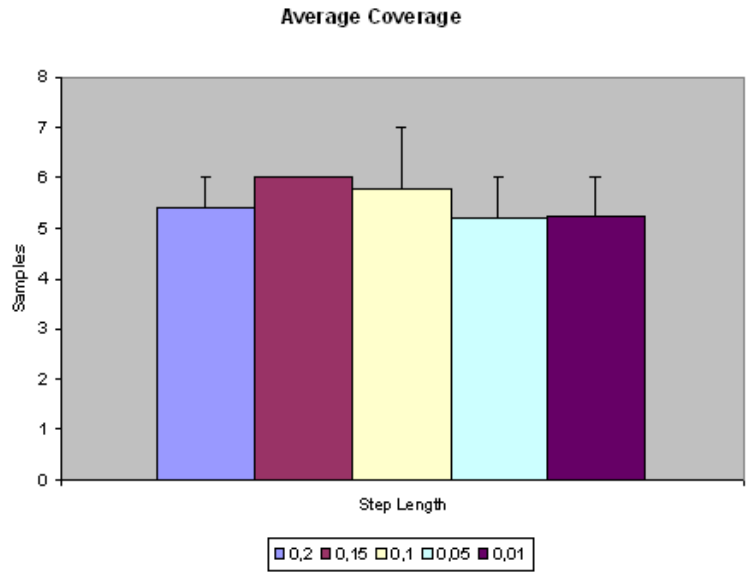


Figure 5.4: Average *Coverage* using different values of σ

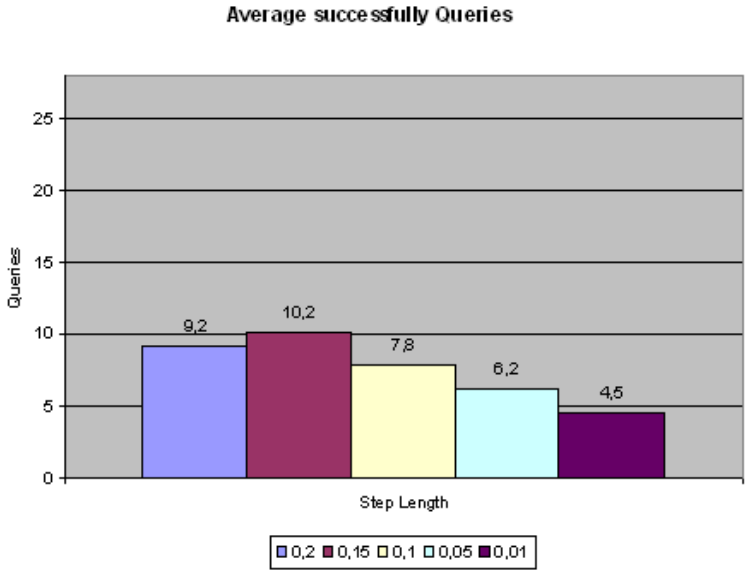


Figure 5.5: Average successfully queries using different values of σ

detailed explanation of this method can be found in [DH03]. Algorithm 5 shows the pseudocode of the Bridge sampling tecnique.

Algorithm 5 Bridge Sampling Algorithm

```
1:  repeat
2:     q ← a random configuration in C
3:     if  q is NOT collision free then
4:        q′ ← is a random configuration at length σ from q.
5:        if  q′ is NOT collision free then
6:           Set q″ to be a configuration between q and q′
7:           if q″ is collision free then
8:              Add q″ as a node in the roadmap
9:           end if
10:       end if
11:    end if
12: until Desired number of nodes created
```

The Bridge name describe the "'bridge"' that is made by the two first samples between the two forbidden areas of the configuration space. The distance σ determines the length of the bridge and how narrow passages the method endeavor to cover. Picture 5.6 shows how the bridge test succeed and fails for different length of σ. The green samples show where the bridge sampler succeed, and the red samples shows when it fails.

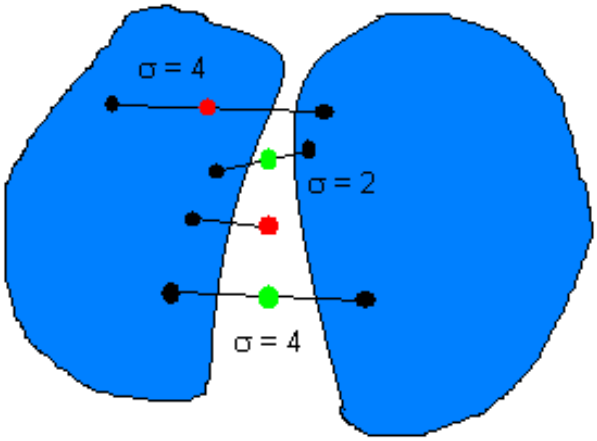


Figure 5.6: Four cases of the Bridge Sampling technique

Experiments with different values of σ were made for this method as well to

determine the optimal distance σ. Same test approach as for the Gaussian method was used by doing some pre testing and then do a final test for a given range of values. Like for the Gaussian method, it turned out that a decreasing value of σ gave best coverage but fewer successfully queries, more connected components and increasing sampling time. But under a certain value, the *Coverage* seemed to decrease again. Figure 5.7 and Figure 5.8 diplays the results from the experiments using a $\sigma = [3\ 2,5\ 2\ 1,5\ 1]$.

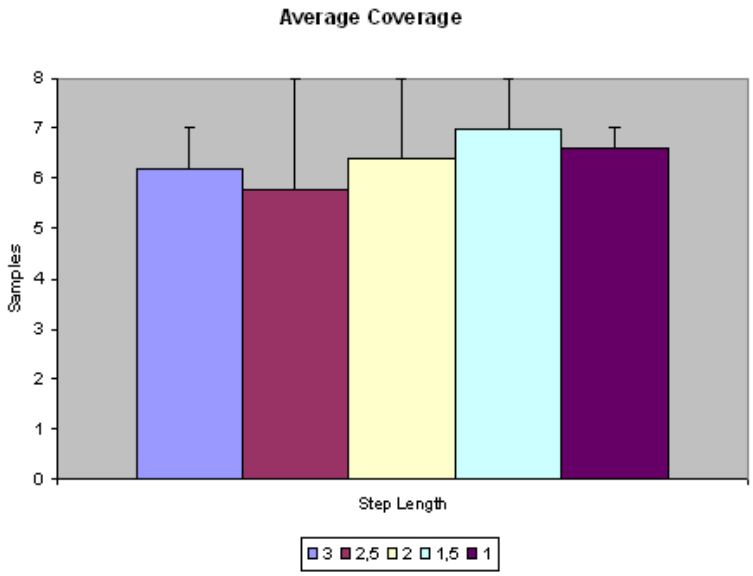


Figure 5.7: Average *Coverage* using different values of σ

The reason to the number of successfully queries decrease with decreasing value of σ is probably when the samples are made close to obstacles they get very hard to connect together. A remedy to this could be to make every tenth sample by random, not using the Bridge method. This will ensure that the roadmap will contain some nodes out in the open space which will most likely make it easier to connect the nodes together. In the following experiments this is done for both the Bridge and the Gaussian sampling techniques.

5.3.3 Grid Based

A version of the grid based sampling technique was also implemented and tested. The environment was partitioned into 48 cells in the x-y plane. The goal was to make 10 collision free configurations for each square. This is off course a lot easier when the robot is far away from the processing equipment

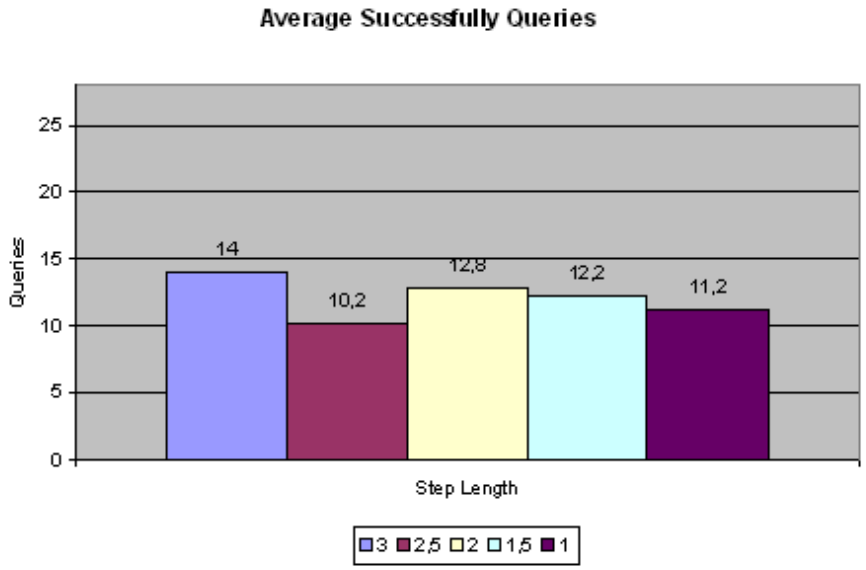


Figure 5.8: Average successfully queries using different values of σ

than when it is very close, where it can be impossible. Because of this the sampler was only given a certain number of tries for each square. If the sampler managed to make 10 valid nodes for each square, the total number would be 480 configurations. Since it is very likely that it will not manage to make 10 valid nodes for al the squares, the final number of nodes will probably be less. The test result of the Grid Based Sampling technique and comparison to the others is presented in 5.3.4

5.3.4 Comparison of test results

When the optimal step lengths for the Gaussian and Bridge sampling techniques were found the three sampling methods mentioned above and the ordinary Random Sampling method described in Section 3.3 were tested and compared against each other. Each sampling method was ran 10 times and *Coverage* and *Connectivity* were measured. The latter was measured to check if *Maximal Connection* was obtained. In addition the number of connected components and sampling time were measured. In each test 300 samples was made and the parameters mentioned above were measured for every 100 samples made. An exception is for the Grid test where 480 samples were made and the parameters measured for every 300, 400 and 480 samples. In the Gaussian and Bridge sampling every tenth sample was made by random to reduce the number of connected components and thus

hopefully increase the *Connectivity*. The result of the different sampling techniques is shown in Tables 5.1, 5.2, 5.3, 5.4.

	100 (300)	200 (400)	300 (480)	Max Value
Random	3,8	4,4	4,5	5
Grid	4,5	4,9	4,9	6
Gaussian	4,5	5	5,5	7
Bridge	5,1	6,6	7	8

Table 5.1: Comparison of the average coverage obtained from the different sampling techniques

	100 (300)	200 (400)	300 (480)	Max Value
Random	5,5	7,6	8	10
Grid	7,1	8,8	8,8	10
Gaussian	3,8	6,9	9,3	15
Bridge	4,7	12,2	15,2	21

Table 5.2: Comparison of the average number of succeeded queries obtained from the different sampling techniques

	100 (300)	200 (400)	300 (480)
Random	1,3	1,5	1,6
Grid	5,5	4,1	4,1
Gaussian	17,9	21,7	21
Bridge	20,5	16,8	16,8

Table 5.3: Comparison of the average number of connected components obtained from the different sampling techniques

The *Coverage* obtained from the Random Sampling technique was poor. When 100, 200 and 300 nodes were made the average *Coverage* was 3,8, 4,4 and 4,5. The last two numbers shows that adding even more nodes will much likely not increase the *Coverage* significantly. The query results and the numbers of connected components shows that the roadmap is more or less connected into the same component. This is not very surprising since the roadmaps generated do not contain any difficult nodes covering the narrow passages.

From experiments done with Grid Based Sampling it turned out that it managed to make 10 nodes in each square. By this, 480 nodes were made in each test. But from the analysis done, it turned out that the Grid Based Sampling technique performed worse than the Bridge and the Gaussian sampling techniques. The average coverage from 10 tests was 4.9 where only one test managed to cover 6 of the test nodes.

	100 (300)	200 (400)	300 (480)	SUM
Random	7,8	8,4	8,4	24,6 min
Grid	9,6	8,4	6,6	24,1 min
Gaussian	28,8	27,6	28,8	1h 25,2 min
Bridge	39	37,6	37,2	1h 51 min

Table 5.4: Comparison of the average sampling time for each sampling technique

It turns out that the Bridge test performed best when it came to *Coverage* and *Connectivity*. The Bridge test was the only one that managed to cover all the 8 test configurations, but it only happened 2 times out of 10. The minimum cover was never under 6.

3 of the 8 test configurations were located in difficult areas between pipes and tanks. This is what typically would appear as narrow passages in configuration space. The other test configurations were made close to the processing equipment but with good clearance from one side. All the sampling technique managed to cover the 5 "easy" test configurations, but it was when covering the 3 difficult some of them failed. Not very surprisingly the Gaussian performed well but the Bridge performed better because the Bridge aims to cover narrow passages and not only close to obstacles as the Gaussian. By this it is concluded that of the 4 sampling techniques tested, the Bridge Sampling technique with a "bridge" length of 1,5 in combination with a Random sampling tehcnique gives best *Coverage* and *Connectivity* of the C_{free} space.

Tabel 5.2 shows that even if all the 8 test samples were covered it did not imply that *Maximal Connectivity* was achieved. This is due to the fact that the roadmap was not connected into one big connected component. By investigating the size of the connected components it turned out that the majority of the samples was connected together in on big connected component, while some samples were single and not connected to any else. A method to reduce the number of connected components is described and investigated in Section 5.3.5. Table 5.3 shows that for some of the methods the number of connected components is less for 300 samples than for 200 samples. This seems logical since a high number of nodes provide a denser roadmap with better connectivity. The effect of a roadmap containing a very high number of nodes is tested further below.

In general, one of the disadvantages with the Bridge Sampling technique is that it is slower than the others. But this could maybe be compensated for if fewer samples are needed to obtain satisfactorily coverage. This is not investigated since sampling time has low priority in this project.

5.3.5 Connection Sampling

The results from the sampling technique testing shows that a greater challenge than cover the entire C_{free} space is to connect the roadmap into one big connected component. This means that after the sampling is done, it is necessary to make an effort to connect the connected components together. This is called "Connection Sampling".

The reason why the PRM fails to connect all the nodes into one big connected component is that some of the nodes are placed in difficult areas such as narrow passages. The challenge is therefore to detect nodes in these regions and then manage to connect the components they belong to together. In [CLH05] it is proposed to associate every configuration q with a heuristic measure of the difficulty of the region around q expressed by a positive weight $w(q)$. Thus, the larger $w(q)$ is the more difficult is the region it lies in. The weights are normalized such that the sum of all configurations in the roadmap is one. The weights can therefore be calculated only once in a given period, and its not possible to calculate a weight for every new node added.

According to [CLH05] there are several methods to define the heuristic weight $w(q)$ and a method that are found to work well in practice is function 5.2

$$w(q) = \frac{\frac{1}{deg(q)+1}}{\sum_{q' \in V} \frac{1}{deg(q')+1}} \tag{5.2}$$

where $deg(q)$ is the number of configurations to which q is connected.

During the testing of different samplings methods it turned out that nodes in difficult regions belonged to connected components only containing one or a few nodes. This gave an idea that the size of the connected components could also be used as an adequate measure to detect difficult regions. Because of this the nodes was first sorted according to the size of their respective connected component. Then the nodes in each connected component were sorted by its difficult weight (5.2). So first of all was the node with highest difficult weight in the smallest connected component attempt connected to another connected component. If it failed, the node with next greatest difficult weight in the same connected component was tried. If all nodes in a connected component were tried out without managing to connect to another connected component the method moved to the next smallest connected component. If there still are connected components only containing one or

a few nodes after the connection sampling is executed, one should consider deleting them from the roadmap because they are not very useful. This has not been done in the implemented method.

When a sample lying in a difficult region is found, its configuration q needs to be expanded to attempt connect it to another connected component. To do this the following method was used. The method is inspired from the Expansive-Spaces Trees (EST) and the Rapidly-Exploring Random Trees (RRT) descibed in [CLH05]. A new configuration $q*$ in the neighborhood of q is made by moving a random step with a given step length. If there exists a collision free path between q and $q*$, $q*$ is temporary stored. If there also exists a collision free path between $q*$ and a configuration in another component, the components are merged and $q*$ is added to the roadmap. If the latter case fails n number of times, then neighbor configurations $q**$ to the neighbors $q*$ are found and attempt connected. When finally two components are connected, the involved neighbors should be added to the roadmap. The method continues to make and connect neighbors until maximum 10 neighbors are connected. If it fails to connect after adding 10 neighbors, all the 10 neighbors are discarded.

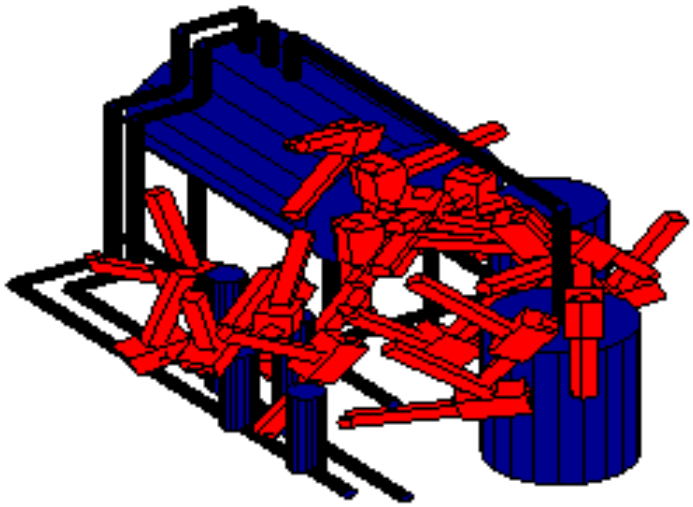


Figure 5.9: The configurations beloning to the connected components with less than 5 nodes. Only the 4 last links of the robot manipualtor are displayed for simplicity.

The Connection Sampling terminates when it either manage to connect all the nodes into one connected component, or when it has tried all the nodes in the small connected components and the nodes with highest weight in

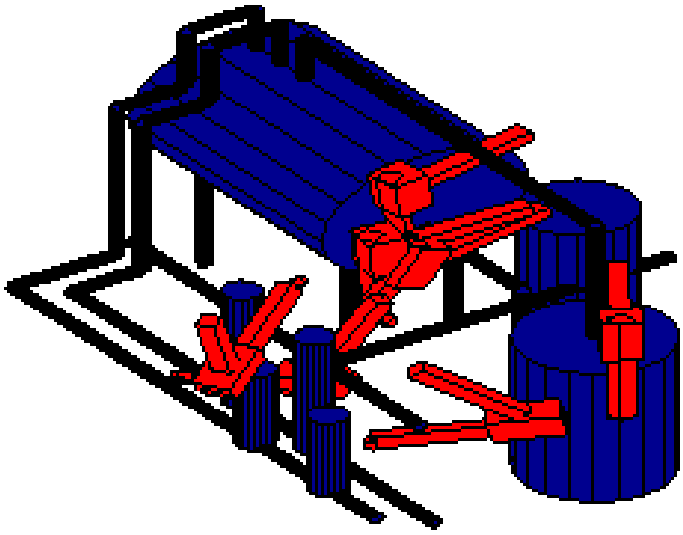


Figure 5.10: The configurations beloning to the connected components with less than 5 nodes after the Connection Sampling method was executed

the big connected component. By choosing other termination criteria the execution time could probably be reduced but still achieving the same result. But studying this in detail has not been prioritized.

To test the "Connection Sampling" method a new roadmap was made. In Section 5.3.4 several roadmaps containing 300 nodes were made to find the optimal sampling strategy. In those roadmaps a sample was added to a connected component when it managed to connect to another node in that connected component. It did not attempt to connect to other nodes in the same connected roadmap because it does not improve the *Reachability* in the roadmap. But, when finding the shortest path through the roadmap it is beneficial that a node is connected to many other nodes in the same connected component to avoid unnecessary detours. It is also easier to distinguish which nodes are located in difficult areas and not because nodes in difficult areas have few neighbours. For the following test a new roadmap was therefore made where new nodes attempt to connect to max 75 other nodes in the same connected component. To test the effect of a big roadmap the new roadmap was made with 1000 nodes.

It took 22 hours and 6 minuets to generate the big roadmap. A sampling analysis was done and it turned out that all the 8 test configurations were covered, but only 21 of 28 queries succeeded. The 7 last queries failed because it was not possible to connect test configuration 3 to the same con-

nected component as the other test configurations. The roadmap consisted of 17 different connected components, where one big connected component with 980 nodes contained the majority of the nodes.

In the Connection Sampling a difficult node is expanded by moving one step in a random direction. This is the same as done in the Gaussian Sampling technique. In section 5.3.1 it turned out that step size 0.1 gave best coverage in narrow passages. It is therefore most likely that the same step size will provide best improvements in the Connection Sampling method. To find out if this was true and to test the performance of the Connection Sampling method, several tests with different step lengths were executed. The step lengths used were 0.5 and 0.1. The Connection Sampling method was ran 5 times for each step length.

Test	1	2	3	4	5
Number of Connected Components	3	1	3	2	3
Sucessfull Queries	21	28	21	28	28
Average execution time	2h 44min	3h 45min	2h 54min	5h 1min	3h 11min

Table 5.5: Results of Connection Sampling using step length 1

Unfortunately the result from the test using step length 0.1 is not available since an error occurred during testing. The test had run for several hours when the incident happened and it was not possible at that time to run a new test. But the test with step length 0.5 terminated successfully. It turned out that in 1 of 5 cases it succeeded to connect all the connected components into one big connected component. In 3 of the 5 cases all the 28 queries were successful. This shows that the Connection Sampling improved the *Connectivity* in the roadmap and that *Maximal Connection* most likely is possible to achieve. The average execution time was 3 hours and 31 minutes but it varied a lot and the shortest time was 2hours and 44 minutes and the longetst time was 5 hours and one minute. The results are shown in Table 5.5. It is not known if using step length 0.1 would provide a higher success rate, but there is no doubt that it would spend more time. But, since the Connection Sampling is performed off-line the execution time is not a big issue. (A video showing the path starting in test confgiuration 3 and ending up in test configuration 5 is named hardPathSmooth.avi and can be found in the attached files)

5.4 Query phase

In this section the work regarding the query phase and optimization of the query time will be presented. The query phase consists of three steps: 1. Connect a start and goal node to the roadmap 2. Find the shortest path through the roadmap. 3. Smoothing the path and decreasing the path length. The execution of part 1 has been studied in detail, but the main contribution to optimization has been done in part 3. In part 2 is Djikstras algorithm used to find the shortest path through the roadmap. This is a well known and very fast algorithm and has not been emphasized to improve.

5.4.1 Connecting start and goal node

When a query is given the first step is to connect the start and goal nodes to the roadmap. The start node would typically describe the current configuration of the robot while the goal node represents the desired final configuration. It is very important that the start and goal nodes are connected to the same connected component. If they are connected to different connected components no feasible path between them can be found. Because of this the start and goal nodes are only attempt connected to the same connected component, starting with the largest. If they fail the second largest connected component is tried out, and so on. The distance metric function is used to find the closest nodes in the roadmap for both the start and the goal nodes. A modified version of the connection part in Algorithm 3 in Section 3.3 is presented in Algorithm 6

The procedure in Algorithm 6 was implemented and tested on the roadmap with 1000 nodes mode in section 5.3.5. For simplicity the test configuration 1 was always used as start configuration and feasible paths to the other test configurations were attempt found. For each query different parameters was measured. The results are shown in Table 5.6.

The first row lists the number of the test configurations to where feasible paths were found. As seen, a feasible path to test configuration 3 was not found. In the second row is the number of attempts needed to find a node that coould be connected to the goal node. Most of them needed only one, except of the last test configuration that needed 2 tries. The connection time used by the local planner to verify a feasible path between the connected nodes is listed in row 3 and 4. Since the same configuration was used as start node every time, this value is almost constant. The values from row 3 and 4 are then summarized in row 5.

Algorithm 6 Connect Start and Goal nodes

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
G = (V,E): the roadmap computed by algorithm 2

Output:

G = (V,E): updated roadmap containing the start and goal nodes

1: $CC \leftarrow$ the connecetd components in G sorted by decreasing order
2: cc is the largest connected component in CC
3: **for** length(CC) **do**
4: $N_{qinit} \leftarrow$ the k closest neigbors of q_{init} in cc according to $dist$
5: $N_{qgoal} \leftarrow$ the k closest neigbors of q_{goal} in cc according to $dist$
6: set q' to be the closest neighbor of q_{init} in N_{qinit}
7: **repeat**
8: **if** $\Delta(q_{init}, q') \neq NIL$ **then**
9: $startConnected \leftarrow true$
10: **else**
11: set q' to be the next closest neighbor of q_{init} in N_{qinit}
12: **end if**
13: **until** $startConnected$ or the set N_{init} is empty
14: set q'' to be the closest neighbor of q_{goal} in N_{qgoal}
15: **repeat**
16: **if** $\Delta(q_{goal}, q'') \neq NIL$ **then**
17: $goalConnected \leftarrow true$
18: **else**
19: set q'' to be the next closest neighbor of q_{goal} in N_{qgoal}
20: **end if**
21: **until** $goalConnected$ or the set N_{qgoal} is empty
22: **if** $rtConnected$ **and** $goalConnected$ **then**
23: $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
24: $E \leftarrow (q_{init}, q') \cup E$
25: $E \leftarrow (q_{goal}, q'') \cup E$
26: **return** G
27: **end if**
28: set cc to be the next connected component in CC
29: $startConnected \leftarrow false$, $goalConnected \leftarrow false$
30: **end for**
31: **if not** $startConnected$ **or not** $goalConnected$ **then**
32: **return** Connection attempt failed
33: **end if**

Test config	2	4	5	6	7	8	
Failed goal connection attempts	0	0	0	0	0	1	
Goal Connection	8,27	2,37	8,36	8,90	4,44	8,51	seconds
Start Connection	1,91	1,90	1,90	1,92	1,91	1,90	seconds
SUM	10,19	4,27	10,26	10,82	6,34	10,40	seconds
Calculating distances	0,11	0,11	0,11	0,11	0,11	0,11	seconds
Djikstras	0,60	0,68	1,06	0,45	0,42	0,82	seconds
Entire Query	**10,90**	**5,07**	**11,43**	**11,38**	**6,87**	**11,34**	seconds
Connecting	93,5	84,3	89,8	95,1	92,4	91,8	%
Djikstras	5,5	13,5	9,2	3,9	6,0	7,3	%

Table 5.6: Comparison of the average coverage obtained from the different sampling techniques

At the start of every query the distance from the start and goal configurations to all the nodes in a connected component are calculated and stored in two arrays. These arrays are then sorted with increasing distance. The execution time of this step,which depends on the number of nodes in the connected component, is listed in row 6. Even if it was almost 1000 nodes in the connected component used, the execution time was very low compared to the other steps in the query phase.

When a start and goal configuration was connected to the roadmap Djikstras Algorithm was used to search through the neighbour graph to find the shortest path. The time Djikstras algorithm needed to find the shortest distance is listed in row 7.

As seen from row 8 in Table 5.6 the time spent by the local path planner to connect when a valid node is found is the most time consuming part in the query time.

A similar test was done to a roadmap containing 400 nodes. In this test 5 of 7 queries succeeded, which is a worse result than for the previous test. It also turned out that in some queries the planner spent more time finding a suitable node to connect the goal node to. When connecting the 8th test configuration for example, it needed 27 tries before it finally succeeded. For the 26 unsuccessful tries the local planner spent in average 0.12 seconds pr attempt to detect a collision, and 3.12 second in total. Djikstra algorithm spent 0.2465 finding the shortest path between test configuration 1 and 8. This is about 30 % of the time, or 0.58 second less than the same query in the large roadmap. The planner spent 2.88 and 6.56 second to verify a feasible path when connecting the start and goal nodes. When adding up all the steps the final query time is 12.86 seconds, which is 11.48 % more

Test config	2	5	6	7	8	
Failed goal connection attempts	5	0	1	0	26	
Average collison detection	0,06		0,11	0	0,12	seconds
Sum failing	0,25	0	0,11	0	3,12	seconds
Goal Connection	12,74	6,67	6,74	6,71	6,56	seconds
Start Connection	2,84	2,92	2,92	2,90	2,88	seconds
SUM connection	15,58	9,59	9,66	9,61	9,44	seconds
Calculating distances	0,05	0,05	0,05	0,05	0,05	seconds
Djikstras	0,04	0,29	0,05	0,19	0,25	seconds
Entire Query	**15,92**	**9,93**	**9,87**	**9,85**	**12,86**	seconds
Connecting	97,9	96,6	97,8	97,5	73,4	%
Djikstras	0,2	2,9	0,5	1,9	1,9	%

Table 5.7: Comparison of the average coverage obtained from the different sampling techniques

than for the same query in the larger roadmap.

As seen from the results above the most time consuming part in the query phase is verifying a feasible path, by the local planner, between a node in the roadmap and to the start or goal node. How long it takes to verify a feasible path depends on the resolution of the binary path planner. An increasing step size results in a faster path planner with poorer resolution and thus less accurate. The step size used in the simulations will not affect the result in practice because then the local path planning will be taken care of by the local planner in Robot Studio.

From the second test it turned out that trying out many nodes before finding a feasible path, might be time consuming if the number of tries is high. The number of unsuccessful tries was lower in the large roadmap than in the smaller roadmap.

The time used calculating the distances and finding the shortest path decreased for a smaller roadmap. But, these steps are the least time consuming steps and have therefore less impact on the query time than the two others steps mention above.

According to the observations mentioned above a high number of nodes in the roadmap does not necessarily affect the query time negatively. If the connection method succeed in connecting the query configuration to the first configuration tried, this will positively affect the query time. The probability that the local planner succeed on first try seems to increase with the number of nodes in the roadmap. Speeding up the local path planner and ensuring

that the most suitable node in the roadmap is attempt connected first will improve the query time most.

In the results presented above the query time for the queries failing were not presented. The query time for the query failing in the large roadmap was 65 seconds, and ca 35 seconds for the two queries in the small roadmap. In line 4 and 5 in Algorithm 6 k is used as an upper limit of connections tried in one connected component. In the tests executed above k was set to infinity, and this is why the query time varies for the failing queries. The value k determines how fast an unfeasible query is detected. By using the same value independent of number of roadmap size. As mentioned earlier in other sections, $k = 75$ has tunred out to be a suitable value to limit the number of tries. As seen from the results above, this value should be great enough since the query needed most tries needed 27 tries before it succeeded. New test was executed with $k = 75$. The new query times for the failing queries was now 17.71 for the large roadmap, and 10.06 and 19.14 for the small roadmap. This time could be reduced even further by reducing k. The drawback by doing this is an increasing chance of missing valid queries.

5.4.2 Backup Procedure

As mentioned in 5.1 priority number one is to achieve as good *Reachabiliy* as possible. Even if is made a great effort in the Learning Phase to obtain this, situations might occur when a query fails. To cope with this problem a new method called the "Backup Procedure" is introduced. If a query fails it is possible to run the Backup Procedure that aim to make the query succeed. The Backup Procedure relies on the same approach used in the Connection Sampling. If a query fails because it only was possible to connect the start node to the largest connected component in the roadmap, the Backup Procedure endeavours to connect the goal configuration to the largest connected component by building a tree from the goal node and towards any of the nodes in the largest connected component. The tree is built in the same way as the trees from the difficult nodes in the Connection Sampling method.

To test the performance of the Backup Procedure it was run 10 times on the big roadmap containing 1000 nodes. It tried to connect test configuration 3 to the largest connected component. It succeeded every time but the execution time varied a lot as seen in the graph in Figure 5.11. The average time was 172,23 seconds while the minimum and maximum execution time were 14,69 and 523,18 respectively.

As seen from the results the time it takes for the Backup Procedure to succeed varies a lot. Because the method heavily relies on random configurations, it is hard to predict how much time it will require from time to

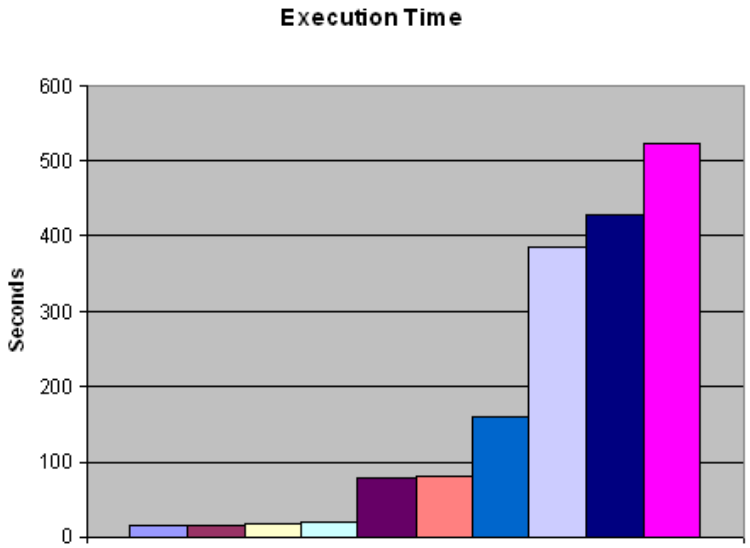


Figure 5.11: Execution time of the Backup Procedure

time. From section 5.4.1 it is known that it takes from 10 to 20 seconds to verify that a query fails. How long it is acceptable to wait for the Backup Procedure depends on the application. What are the alternatives and what are the consequences of a query failing? Maybe a less random approach could make the Backup Procedure easier to predict and more convenient to use.

5.4.3 Path Smoothing

Djikstras algorithm returns the shortest path through the roadmap according to the weights calculated by the distance metric. Due to the randomness in the methods used when generating the nodes the path might be unnecessary long and crooked. After a path is found it is possible to do some path smoothing which remove unnecessary movements and decrease the path length. There exists different techniques doing this [RG06] and [CLH05]. Some of the techniques mentioned in [RG06] and [CLH05] is implemented, tested and analysed in this section.

The main purpose with doing path smoothing is to save time when actually moving the robot from start to goal. Because the path smoothing is executed on-line the execution time is of great importance. In the end one has to consider if the path smoothing algorithm actually reduce the overall query

time. When performing unsmooth movements the robot is exposed to unnecessary torques and forces. Another benefit of executing path smoothing is therefore to decrease the wear and tear on the robot system.

The simplest path smoothing technique only decrease the number of waypoints from start to goal, while more advanced methods operates on the entire path from start to goal. In the following a path is defined by n nodes with v_{init} and v_{goal} as start and goal nodes, respectively. Below are some definitions presented that is used when describing the different path smoothing techniques.

Definition 5.1 (Node path N)**.** *A node path N is a series of node* v_0,...,v_{n-1}*, such that the local path LP*$[v_i, v_{i+1}]$ *are collision-free.*

Definition 5.2 (Adjacent configurations)**.** *The configurations* π_0,...,π_{n-1} *are adjacent configurations if the distance* $d(\pi_i, \pi_{i+1})$ *is at most a predetermined step.*

Definition 5.3 (Discrete Path Π)**.** *A discrete path* Π *is a series of adjacent configurations* π_0,...,π_{n-1}*.*

Definition 5.4 (Discrete Local Path LP)**.** *A discrete local path LP*$[\pi', \pi'']$ *is a series of n interpolated adjacent configurations* π_0,...,π_{n-1} *on the local path between π' and π".*

The first method implemented and tested was a simple path smoothing technique which aim to remove redundant nodes. A node v_i is redundant if the local planner finds a collision free path between v_{i-1} and v_{i+1}. The path is checked by using a greedy approach. The greedy approach first tries to connect the v_{init} and v_{goal}. If it fails it then tries to connect v_{init+1} and v_{goal} and so on. E.g if v_{i+5} manages to connect to v_{goal}, then v_{i+5} is set as a temporary goal node and then the loop starts over again and tries to connect v_{init} to the temporary goal node. This is continueed until the entire path is searched through. Algorithm 7 shows the pseudo code of this method.

While the Node Pruning algorithm only deals with redundant nodes, the next method take the entire discrete path Π into consideration. The next method is called Shortcut and is described in Algorithm 8. The Shortcut method takes to random configurations from the path, π_a and π_b and use interlpolation to find out it there is a valid path between the two configurations. If a valid path exists, then the old path between π_a and π_b is replaced with the new path. Figure 5.13 shows how the initial path is smoothed.

In the Shortcut method all the DOFs are interpolated by the local planner between π_a and π_b. In [RG06] a new technique called Partial Shortcut is introduced. This path smoothing method only interpolates one of the

Algorithm 7 Node Pruning

1: $v_{goal} \leftarrow v_n$
2: $N_{new} \leftarrow \{v_{goal}\}$
3: $i \leftarrow 0$
4: **repeat**
5: **if** LP$[v_i, v_{goal}] \in C_{free}$ **then**
6: $v_{goal} \leftarrow v_i$
7: $N_{new} \leftarrow N_{new} \cup \{v_{goal}\}$
8: $i \leftarrow 0$
9: **else**
10: $i \leftarrow i + 1$
11: **end if**
12: **until** $v_{goal} == v_0$
13: **return** N_{new}

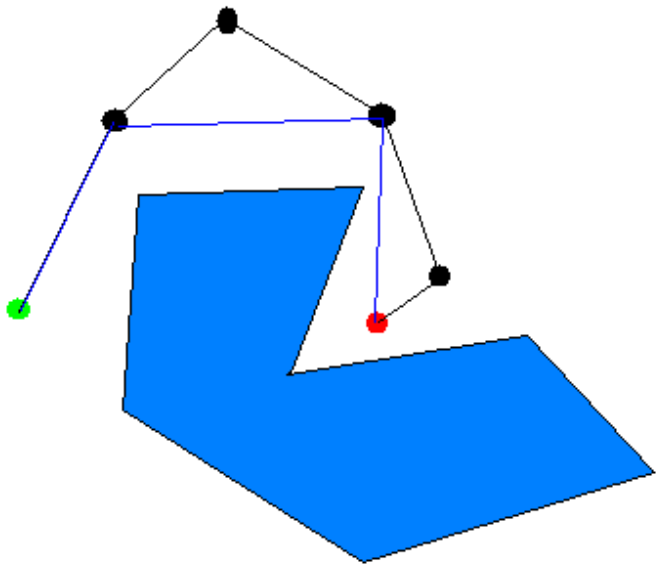


Figure 5.12: Black path is intial path, while blue path is path with redundant nodes removed.

Algorithm 8 Shortcut

repeat
  number of configurations $n \leftarrow |\Pi|$
  $a, b \leftarrow$ two random indices $0 \leq a + 1 < b < n$
  $\Pi' \leftarrow \pi_0, ..., \pi_{a-1}$
  $\Pi'' \leftarrow \pi_a, ..., \pi_b$
  $\Pi''' \leftarrow \pi_{b+1}, ..., \pi_{n-1}$
  if $\mathrm{LP}[\pi_a, \pi_b] \in C_{free}$ **then**
    $\Pi \leftarrow \pi' \cup LP\,[\pi_a, \pi_b] \cup \Pi'''$
  end if
until x number of tries
return Π

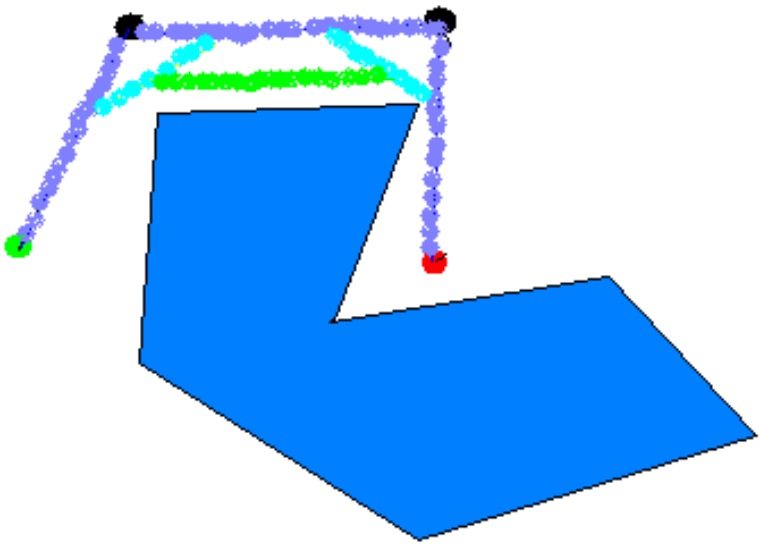


Figure 5.13: The path being smoothed step by step.

degrees of freedom between configuration π_a and π_b. The rationale is that sometimes only one or two DOFs do redundant movements, while the moves in others DOFs are necessary to avoid collision. For example when moving a small cube around a big cube, the linear movements are necessary not to crash into the big cube, while the rotational moves are in most cases redundant. The Partial shortcut method is described in Algorithm 9.

Algorithm 9 Partial Shortcut

1: **repeat**
2: $f \leftarrow$ a random degree of freedom
3: number of configurations $n \leftarrow |\Pi|$
4: $a, b \leftarrow$ two random indices $0 \leq a + 1 < b < n$
5: $\Pi' \leftarrow \pi_0, ..., \pi_{a-1}$
6: $\Pi'' \leftarrow \pi_a, ..., \pi_b$
7: $\Pi''' \leftarrow \pi_{b+1}, ..., \pi_{n-1}$
8: $m \leftarrow |\Pi''|$
9: **for** $\pi_i'' \in \Pi''$ **do**
10: $\pi_i[f] \leftarrow$ Interpolate($\pi_0[f], \pi_{m-1}[f]$, i/(m-1))
11: **end for**
12: ValidatePath(Π'')
13: **if** $\Pi_i'' \in C_{free}$ **then**
14: $\Pi \leftarrow \pi' \cup \Pi'' \cup \Pi'''$
15: **end if**
16: **until** x number of tries
17: **return** Π

The chance that a random DOF f is chosen is given according to its weight w. When only interpolating one of the DOFs it could occur that the distance between two adjacent configurations is greater than the valid step size. Because of this the path has to be validated, (see line 12), such that the maximal step size never is exceeded.

In the Shortcut and Partial Shortcut method index a and b are found by random. A new version of Shortcut, called Deterministic Shortcut, is hereby introduced, that choose index a and b in a more deterministic way. The idea is that the chance of smoothing the movements is greatest where the robot manipulator changes direction i.e. around the waypoints. Because of this index a is always smaller than the index of a given node, v_i, and index b is always greater. a and b are always chosen with the same distance d from the given node, v_i. Because it is desirable to try smoothing the path as much as possible as early as possible, d is large in the beginning and then decreased by 1 for each iteration. The algorithm terminates when it has reached a minimum distance $minDist$ or alternatively when a given time constraint is exceeded. The pseudo code is presented in Algorithm 10.

Algorithm 10 Deterministic Shortcut

```
1: d ← maxDist
2: repeat
3:    i ← d
4:    number of configurations n ← |Π|
5:    repeat
6:       if  π_i ∈ N then
7:          a ← i − d
8:          b ← i + d
9:          i ← b
10:         Π′ ← π_0, ..., π_{a−1}
11:         Π″ ← π_a, ..., π_b
12:         Π‴ ← π_{b+1}, ..., π_{n−1}
13:         if  LP[π_a, π_b] ∈ C_free then
14:            Π ← π′ ∪ LP [π_a, π_b] ∪ Π‴
15:            N ← N ∪ π_a ∪ π_b
16:         end if
17:      else
18:         i ← i + 1
19:      end if
20:   until i = n − d
21:   d ← d − 1
22: until d = minDist
23: return  Π
```

The maximum and minimum distances $maxDist$ and $minDist$ were determined by doing some experiments. Determine $maxDist$ and $minDist$ is a trade-off between path improvements versus time costs. The greater the $maxDist$ is the greater is the improvement done by the local planner between two configurations. But the greater the $maxDist$ is, the more time is wasted on unsuccessful local path planning. It turned out that there was very little to earn in a greater $maxDist$ than 20. The smaller $minDist$ becomes the greater is the chance of finding a feasible local path, but the smaller is the improvements of the path length. Testing showed that when $minDist$ was smaller than 3 the length improvements was very small compared to time costs.

When doing a new search through the literature it turned out that the idea of the Deterministic Shortcut method was not new. In [DH99] a method called Adaptive Shortcut also pick a and b on each side of a waypoint, but using a different approach. When this was discovered there was no time to implement and test the Adaptive Shortcut method.

As mentioned above the main goal with path smoothing is do decrease the path length. When calcuatling the legnth it is distingusihed between rotational and translational DOfs. The path length funciton is the same as used in [RG06]:

$$d(\Pi) = d_r(\Pi) + d_t(\Pi)$$

where

$$d_r(\Pi) = \sum_{i=0}^{n-2} \left[d_r(\pi_i, \pi_{i+i})\right]^2 \quad and \quad d_t(\Pi) = \sum_{i=0}^{n-2} \left[d_t(\pi_i, \pi_{i+i})\right]^2$$

Let $q = \pi_i$ and $r = \pi_{i+1}$. Then, for all k rotational DOFs $0 \leq j \leq k$ and for all $(l-k)$ translational DOFs $k \leq j \leq l$:

$$d_r(q,r) = \sqrt{\sum_{j=0}^{k-1} \left[w_j d(q_j, r_j)\right]^2} \quad and \quad d_t(q,r) = \sqrt{\sum_{j=k}^{l-1} \left[w_j d(q_j, r_j)\right]^2}$$

where the partial distances $d\left(q_j, r_j\right)$ are calculated in the same way as in Section 4.5 and w_j is the weights listed in Table 4.2.

The performance of the path smoothing methods mentioned above were investigated by testing each method on 6 successfully queries in the roadmap containing 1000 nodes. Since the Node Pruning method is deterministic, it was only tested one time for each query. The paths returned from the Node Pruning method were then used when the other path smoothing methods were tested. The Shortcut and Partial Shortcut methods, which rely on random variables, were first tested to find out how much they could decrease the length of the paths. By studying the methods it turned out that they had converged when random values for a and b where tried 500 times. Because the methods are random each method was ran 30 times on each query to find the average path length improvements. To find out the maximal path length decreasing performance of the Deterministic Shortcut, the method was executed with $maxDist$ and $minDist$ set to half the number of nodes in the original path and 0, respectively.

Original length	**16,05**	**17,58**	**32,36**	**13,41**	**13,64**	**21,93**	**length**
Node Pruning	23,78	11,48	8,53	16,27	9,86	6,13	%
Shortcut	55,21	41,88	36,73	48,06	25,03	20,86	%
Partial Shortcut	52,39	38,29	33,74	45,50	26,44	17,59	%
Determined Shortcut	58,97	37,23	38,73	43,20	24,96	19,20	%
Node Pruning	7,48	4,59	9,91	4,44	5,70	7,03	seconds
Shortcut	64,94	69,97	76,08	43,36	156,96	81,43	seconds
Partial Shortcut	232,85	438,81	753,46	294,21	302,18	548,33	seconds
Determined Shortcut	5,58	23,37	67,64	4,08	23,17	37,22	seconds

Table 5.8: Comparison of the maximal path length decreasing

The empirical testing showed that different mehods performed diferent on different paths. Even if they perofrmed different, the differences were not very big, less than 5% for most of the paths. The Partial Shortcut method came worst out of the testing. The results are listed in Table 5.8 where the best results are in red. Because the execution times in the experiments above were unacceptable large, the path length decreasing performance versus execution time was therefore compared in the next experiment. The Deterministic Shortcut method was therefore executed with its optimal $minDist$ and $maxDist$ values presented above. In addition a maximum time limit was used. If the method had not terminated in 20 seconds, it was forced to terminate. The execution time the Deterministic Shortcut method used for each query was then used as maximum time limits when the Shortcut and Partial Shortcut methods were tested. This was done to make the performance of the methods easier to compare. The Shortcut and Partial Shortcut methods were executed 100 times for each query. The average path length reduction in prosent was then measured. The results from the tests are shown in Table 5.9.

Original length	**16,05**	**17,58**	**32,36**	**13,41**	**13,64**	**21,93**	**length**
Node Pruning	23,78	11,48	8,53	16,27	9,86	6,13	%
Shortcut	30,09	21,47	27,04	29,02	16,70	12,92	%
Partial Shortcut	25,42	13,02	13,77	17,20	15,53	6,31	%
Determined Shortcut	54,97	27,98	30,38	42,27	22,55	15,02	%
Node Pruning	7,48	4,59	9,91	4,44	5,70	7,03	seconds
Shortcut	1,72	8,64	22,51	2,90	7,44	15,07	seconds
Partial Shortcut	1,77	9,00	23,52	3,20	7,44	15,53	seconds
Determined Shortcut	1,33	8,11	22,18	2,64	6,95	14,64	seconds

Table 5.9: Comparison of performance when using time constraints

This experiment showed that when time is an issue the Deterministic Shortcut path smoothing method outperformed the random path smoothing methods: Shortcut and Partial Shortcut. Figure 5.14 and 5.15 shows the robot manipulator moving from test configurations 1 to test configuration 5. Figure 5.14 shows the initial path while Figure 5.15 shows the path when in it first is smoothed with Node Pruning and then Deterministic Shortcut. It is hard to see how the robot manipulator actually moves, but it is easy to se that the path is shorter and smoother in Figure 5.15. For simplicity only the robot arm is displayed and the gantry crane is omitted. Animation showing the entire 9 DOF system moving along the path can be found in the avi files attached.

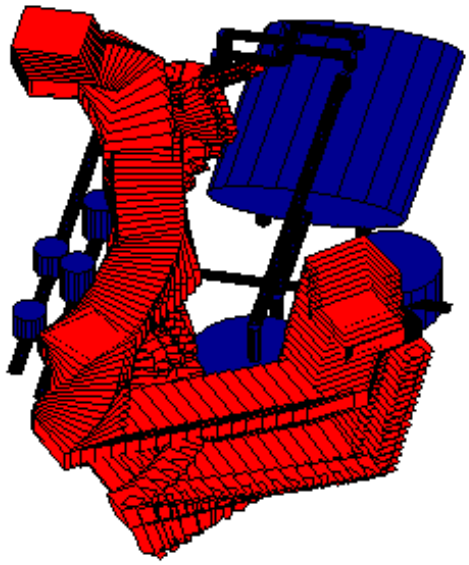


Figure 5.14: Intial pat with many redunant movements

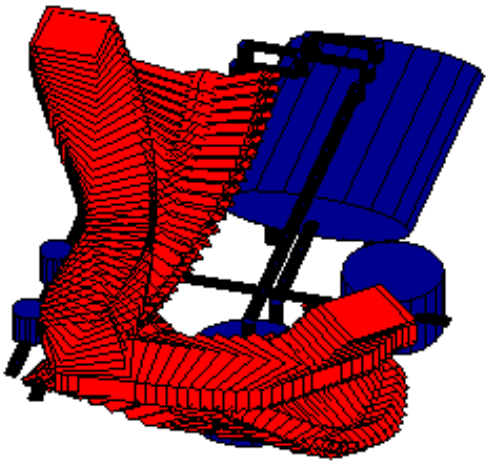


Figure 5.15: Path smoothed and length decreased by 42,27% by the Node Pruning and Deterministic Shortcut methods

5.5 Robust path

It is very important that the path returned by the path planner is collision free. Because the path planner relies on a static model of the robot and the environment it is necessary to ensure that the path has a certain "clarity" to the obstacles. This will reduce the robots chance of colliding in case an unnoticed change in the construction of the robot or the environment occurs. By increasing the distances to the obstacles the path becomes more "robust". There exists different methods to make the path more robust.

One method is called the "medial axis" which ensure that a robot configuration is placed with equal distances between each link and the obstacles. The Medial axis method is mentioned as a sampling method in Section 3.3.2. This method requires the calculation of the closest distance between each link and the obstacles. Since making a robust path is out of the scope of this project, the method has not been implemented and tested.

Another simpler method that will ensure a certain clearance between the robot and the obstacles is to scale the size of either the robot, the obstacles, or both the robot and the obstacles. This method will ensure a safety zone between the robot and the obstacles depending on the scaling.

5.6 Local planner

In the work presented in the sections above a simple but very fast local planner is used to connect the nodes. The drawback of this local planner is that it can only move each joint linearly using interpolation. This reduces its chance finding a feasible path. A more powerful planner is able to move each joint independent and in a non-linear fashion. The disadvantage of a more powerful path planner is that it is computationally demanding and therefore very slow. An example of a powerful local path planner is the Potential Field Planner described in Section 3.1. This powerful path planner is based on knowing the closest distances from the robot manipulator links to the surrounding obstacles at any time. This feature is not supported by Robot Studio and a PRM using the Potential Field planner is therefore not likely to be implemented and tested on the real system. Despite this the use of the Potential Field path planner is investigated. If use of this local path planner improves *Coverage* and *Maximal Connectivity* an effort to develop the distance calculation feature to Robot Studio might be of interest to ABB.

In [KAA07] a 3 DOF potential field planner was implemented and tested. It was shown that it could find a path among dense obstacles if the distance between the start and goal configurations was small enough, and likely to fail over longer distances. To make the 3 DOF potential field planner from [KAA07] handle both the 6 DOF articulated robot manipulator and the 3 DOF linear gantry crane together it was extended from 3 to 9 DOF.

As mentioned above the potential field planner is very slow compared to the binary path planner, but way more sophisticated and has higher chance of finding a path between two configurations. When studying the 9 DOF potential field planner operating among the processing equipment it turned out, as expected, that it had big problems finding a path between two configurations when the distances became to large.

The potential field planner was therefore implemented as a backup planner to the simple binary planner and used only if the simple planner failed. Because the potential field planner is very likely to fail over longer distances, it is only used if the distance between to configurations is under a certain limit. To calculate the distance the metric distance mentioned earlier was used. By trial and error the distance limit was set to 3. Even more time could be used finding an optimal distance limit, but this has not been prioritized.

It was also noticed that when the goal configuration was close to many obstacles, the last part of the path, close to the goal configuration, was very time consuming and very hard or even impossible to complete. The reason

to this is probably all the attractive and repulsive forces acting on the robot manipulator making it shaking or collide when it is close to its goal and many obstacles. To deal with this problem and to make the planner faster the local path planner changes back again from Potential Field back to the binary collision detection planner when the manipulator is close to the goal. If the distance from current configuration to goal configuration is under a certain limit the planner tries to succeed with the binary planner. If it succeed it terminates, if not the limit is decreased and it continues using the powerful planner to try getting closer. Because the Potential Field planner is not symmetric a collision free path needed to be found for the robot manipulator moving both ways.

The effect of using a combination of both a powerful and simple local path planner was investigated by running several tests. The test set up was the same as described in section 5.2 and used in section 5.3. A roadmap was made 10 times by using the Bridge sampling technique and for each time was *Coverage*, *Connectivity* and sampling time measured. The results were compared with the results from the Bridge Sampling tests in Section 5.3.4 where only a simple planner was used. The results from the tests are shown in Tabel 5.10.

Number of nodes	100	200	300	Max Value
Coverage Potential	5,7	6,4	6,8	8
Coverage Simple	5,1	6,6	7	8
Potential used when connecting nodes	15,3	35,8	53,4	
Simple used when connecting node	68	149,9	232	
Queries Potential	6,8	13,8	18	21
Queries Simple	4,7	12,2	15,2	21
Sample time Potential			6 h 47 min	
Sample time Simple			1h 51 min	

Table 5.10: Comparison of a PRM using both a Simple and Potential Field as local planners and a PRM only using a simple local planner

As seen from row 4 and 5 the Potential field planner was used in average 53.4 times when connecting the nodes in the graph and the simple planner 232 times. Despite of this it turned out that using the Potential Field planner as a backup local planner did not improve the *Coverage* compared to only using a simple local planner. Row 6 and 7 shows that the *Connecetivity* was slightly improved by using a more powerful local planner. But, the maximal number of successful queries was still not more than 21, meaning that the 3rd test configuration was still not able to connect to the large connected component. The measuring of number of connected components went wrong

in this test and is therefore not showed. But some pre testing showed that the roadmap generated by using the Potential Field planner as backup planner did decrease the number of connected components. The reason why a powerful local path planner did not improve the roadmap significantly is not known for sure. One reason could be that all the parameters in the 9 DOF Potential Field planner was not sufficient tuned. Spending more time on tuning the Potential Field planner could maybe have improved the results. But, a high dimensional configuration space C is the curse of the Potential Field planner, and it is not sure that there are much room of improvements.

Because of these indifferent results and the fact that using the Potential Filed planner requires calculations of the closest distances between obstacles and robot links, it is not recommended to use in this application. In addition the sampling time was more than three times greater when using the Potential Field as local planner than when only using the simple local planner which shows that the powerful local planner is way slower than the simple one.

5.7 Connect to point

As mention, the work presented above is focused on finding a path between a start and goal configuration. In a typical situation in real life it is more likely that the goal is not described by the entire goal configuration, but only of the position and orientation of the end effector. In the local binary path planner used above both the start and goal configuration are needed because it interpolates between each joint position. If the goal is only determined by the position and orientation of the end effector, the other joint parameters of the goal configuration is not known. Another local path planner is therefore needed when only goal position and orientation is specified.

In the Potential Field path planner the robot manipulator is attracted to its goal configuration by attractive forces depending on the distance between each joint position in the start and goal configuration. Which joints or points on the robot manipulator to use as reference points to calculate the attractive forces is optional. Using more points restricts the motion of the robot. Using only the end effector as reference point will provide great freedom to the robot manipulator movements and how the goal configuration could be. But, guiding the entire robot manipulator using only one reference point is not straight forward.

At first a version of the Potential Field path planner mention above which only used position and not orientation as input was implemented. The manipulator Jacobian was used to convert the attractive force acting on the end effector to torques acting on each of the rotational joints on the 6 DOF

robot manipulator. The repulsive forces were calculated as usual as in the ordinary Potential Field planner described in Section 3.1. By this there were no attractive forces but only repulsive forces acting on the linear joints on the 3 DOF gantry crane.

The query itself was executed the same way as in Algorithm 6. Except this time the distance from the goal position and the end effector position of the configurations was used as sorting criteria when the goal position was attempt connected to the roadmap.

The planner described above was tested on a roadmap containing 400 nodes generated by the Bridge Sampling technique. The 8 test queries described in Section 5.2 was used. But this time only test configuration 1 was used as start configuration, while the end effector positions from the 7 other test configurations were used as goal positions. Reaching exactly the end position is often very hard because of the nature of the Potential Field planner. Because of this an error limit of length 2 was accepted. If the planner failed to reach its goal position area by using more than 40 iterations, it was aborted and a new configuration node in the roadmap was used as initial configuration.

It turned out that the planner managed to succeed all the 7 queries. 5 of the queries were solved at first try and the query time was 10.04, 5.72, 4.47, 5.17 and 3.15 seconds. One query was solved after 2 tries with a query time of 13.3 seconds. One query time turned out to be very hard and needed 9 tries and 190.34 seconds to succeed. So, after all the test turned out quite well. There could be several reasons to this. One is the generous error limit used, and another could be that even if the goal configurations are inside a difficult area and hard to reach, the goal position of the end effector might be outside this, and quite easy to reach from another angle. The latter leads to the last reason, the fact that end orientation is not taken into account.

To make the test harder the goal error limit was decreased to 1. When connecting to the goal position only the 15 closest end effectors were tried. This limit was set because the probability of success decrease with number of tried. Now it became much harder for the planner to succeed. Only 4 of 7 queries succeeded. The connection time for those succeeding was still low: 14.39, 5.05, 6.43 and 2.97. The main reason to some failed is that some of the goal positions are close to obstacles where the repulsive forces are very large. This makes the robot shake back and forward as mentioned earlier in Section 5.6.

To cope with this problem the repulsive force factor η was decreased when the robots end effector was close to its goal. This resulted in 7 of 7 queries succeeded. In addition to this, the query time for some of the hard queries

had had decreased significantly compared to the very first test. Now the maximum query time was 91.61 seconds for the worst, compared to 190 seconds from earlier.

When the error limit was decreased even more, to 0.5, which corresponds to 5 cm in the real world, the planner again struggled to connect to all the goal points. The reduction of the η was not enough. A modification of the method above was therefore done. When the end effector reach inside a limit of 1, the η was set to zero. This means that no repulsive forces acted on the robot. To decrease the chance that the robot moved past the goal and bumped into any obstacles the step length α was reduced by 50%. By doing these changes the planner performed better and the worst time was actually reduced to 57.86 seconds.

For goal error limits under 0.5 the planner struggled to connect to many goal points. Some modifications were done to try improve this, but with no greater success.

As mentioned above, the local path planner has so far only dealt with goal position and not taken goal orientation of the end effector into account. Adding the orientation as a requirement will most likely add the demand to the path planer. The orientation was taken into account by adding an extra goal point , goal point 2, as reference point. The goal point used so far was located at the desired goal point of the end effector. Goal point 2 was located where the start of the end effector would be to get the desired orientation. By using these to points the orientation of the end effectors Z axis was specified. An extra attractive force was therefore calculated between this new point and the origin of the 9th coordination frame of the robot manipulator.

The planner would succeed if both reference points on the robot manipulator were close enough to their goal positions. To make sure that the end effector was pointing in the right direction along the Z axes the distance from frame 10 to goal point 1 had to be less than the distance from frame 10 to goal point 2.

Now the query became harder for the planner to solve. I turned out that only 3 of 7 query succeeded. The query time did also increase for some of the successful queries. The query with greatest time spent 113.90 seconds, while the one with least time spent 13.82 seconds to solve the query.

A deeper study of a planner based on the ideas above could probably improve the results even more. The object of the work presented in this subsection was just to briefly investigate one approach that could solve a query with a given goal position and orientation. An expansion of the planner's range of application could be when the entire path for the end effectors position and

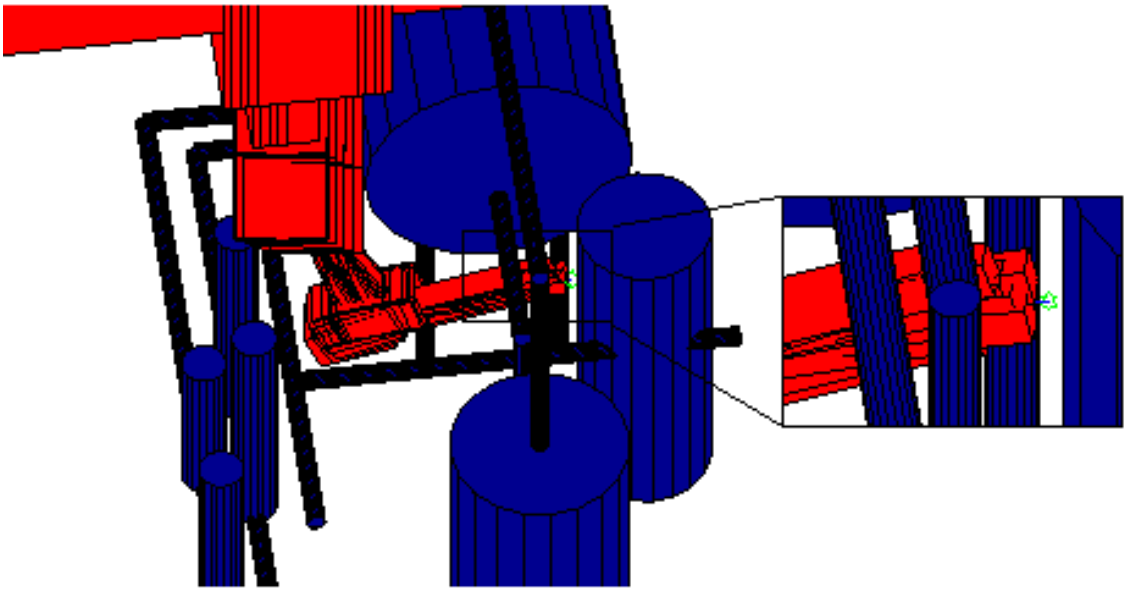


Figure 5.16: The end efector reaching its desired goal position and orientation.

orientation is specified. Then it could guide the end effector from sub goal to sub goal.

Another idea how to connect to a specified goal position and orientation that not relies on the potential field method and knowing the shortest distances between the robot and the obstacles is a method where inverse kinematics is used. By using the inverse kinematics of the robot manipulator a goal configuration could be made that has the desired end effector position and orientation. If the goal configuration is collision free it is valid and could be used. If not, a new configuration needs to be made. Because the robot manipulator is redundant there exist several joint parameter combinations that give the same end effector position and orientation. The challenge is to find the one that is collision free. In [DH99] a path planner using this approach has been implemented and tested with great success. Because of limited time this last method has not been investigated any further.

The path planner in this paper finds the path from a given start configuration to a given goal configuration. According to ABB it is also desirable to make a path planner that generates a path from start to goal where the position and orientation of the end effector is specified along the entire path. An idea is to use the last method mentioned above and presented in **??** to

generate dense waypoints with the desired position and orientation of the end effector along the path. Then a local planner could be used to move the robot manipulator between the waypoints. Because this was outside the scope of this project the idea has not been investigated further.

Chapter 6

Implementation at ABB

One of the main goals in this project was to implement the developed path planner at ABB in Oslo and try it out on the real robot system. As a consequence of this it was always emphasized to make the robot model and the experiments as close to reality as possible. To avoid discrepancies it was necessary to cooperate close with ABB during the project.

6.1 Differences between real system and model

The robot model in Matlab was made according to some robot data sheets and drawings of the real robot system provided by ABB. When the model in Matlab was tested against the model in Robot Studio, it turned out that there were some discrepancies between the two models. The real robot has a parallel bar between joint 2 and joint 3. Because of this parallel bar the robot has a feature that keeps the angle of the third link relative to the horizontal plane constant if a change in joint 2 is made. Because this feature was not mentioned in the data sheets, the feature was not taken into account when the robot was modelled in Matlab.

In the Matlab model the robot gets its joint angle input for joint 3 relative to link 2. Because of the parallel stabilisation Robot Studio prefer to get the joint angles input for joint 3 relative to a horizontal line. Because of this the joint limits for joint 3 moves up and down depending on its orientation. At first it was expected that the discrepancies mentioned above would cause big problems when the two programs should start cooperate. Fortunately it turned out that the discrepancies could easily be dealt with. According to the robot data sheets the joint limits for joint 3 are always -60 to 65 degrees relative to link 2. So, as long as the random joint angles for joint 3 generated

in Matlab stayed inside these limits the moving joint limits in Robot Studio were not exceeded either. Before the joint value could be transferred to Robot Studio it needed to be converted to the format RS used, i.e. relative to the horizontal plane. This was done by adding the joint 2 angle to the joint 3 angle. E.g. if joint 2 and 3 was 30 and 35 degrees, respectively, in Matlab, they would be 30 and 65 degrees in Robot Studio.

6.2 Implementation and implementation issues

The path planner implemented in Matlab used only a simple model imitating the real test rig used by ABB in Oslo. Before the path planner could be tested on the real system a detailed model of the real system was required. At ABB they had a detailed model of both the robot system and the processing equipment in Robot Studio. Robot Studio is an offline robot programming tool, which enables the user to develop and test robot programs in a 3D environment. The reason why a detailed model of the robot system and the processing equipment is needed is to do collision detection. This can be done in Robot Studio for both one single configuration and between to configurations, given the joint variables as input. Because of this it was determined to let Robot Studio do the collision detection required by the path planner.

Next challenge was to make Matlab and Robot Studio cooperate. The standard RS program comes with an Application Programming Interface (API), which enables developers to extend the functionality of Robot Studio by building custom software applications. All custom developed code must be written as an AddIn, which is loaded into Robot Studio at start up. As all path planning algorithms were developed in the Matlab, and the Robot Studio API uses the Microsoft .NET framework, a suitable way of integrating both environments needed to be found. The Matlab Builder NE toolbox was used to convert the Matlab functions into dynamic link libraries (dll's), which can then be called from within the .NET environment. This approach was originally chosen because there would be no need to re-write the algorithms and therefore providing a faster way to implement the path planning algorithms.

In order to access the compiled dll's, the class must be instantiated. This was possible within a console application and a light version of the path planning algorithm, where simulated collision detection events were used, was tested successfully.

Problems arose within the AddIn application when used in Robot Studio. Robot Studio uses a standard entry procedure to read all objects used to

program the custom AddIns. There appears to be a major conflict between the objects generated from the Matlab Builder NE toolbox and this standard entry procedure. The cause of this conflict is still unknown.

Because of this inherent problem with software integration, it was decided to re-write all Matlab functions in the .NET environment. This is a much more time consuming task, but the likelihood of success is much higher as .NET developed AddIns have been successfully deployed previously.

The manner in which Robot Studio deals with collision events caused some problems with the developed path planning algorithms. Whilst the Matlab functions returned a TRUE or FALSE to a collision event, Robot Studio returns a CollisionStarted or CollisionEnded event for each link or object involved in the collision. The decision logic therefore had to be changed so that for each configuration, all collision events were counted. This ensured that the collision checker returned more accurate results.

Because of these unexpected technical challenges the implementation of the path planner at ABB in Oslo got delayed and was therefore not finished within deadline. As a consequence of this, the tests on the real system could not be executed. The staff at ABB will continue to work on the implementation so that the path planner can be tested in near future.

6.3 Communication between programs

Since collision detection is an important and time consuming part in the path planning, the communication between the two programs had to be very fast. Table 6.1 describes how the data is meant to be sent back and forward between the two programs when making a roadmap. The flow in step 1 depends on the sampling technique used. In the example in Table 6.1 a simple random sampling technique is used for simplicity. For other sampling techniques, like the Bridge or the Gaussian, the flow in step 1 would be different. The flow in step 2 to step 5 would not change because they are independent of the sampling technique used.

.NET framework		Robot Studio
1. A random configuration (RC) is made.		
	Send RCs joint variables for collision check →	
		Check if RC is colliding
	← Return GO/NOGO	
if GO: **2.** Store RC as a node in the roadmap. **3.** Try connect RC to the closest node (CN) in the roadmap.		
	Send RC and CNs joint variables for collision check →	
		Do a collision check between RC and CN using interpolation.
	← Return GO/NOGO	
if GO **4.** Store the distance between RC and CN and add RC to CNs connected component.		
elseif NOGO **5.** Return to 3 using the second closest node. Repeat k times or until GO is returned.		
end **end**		
REPEAT until n number of nodes are made		

Table 6.1: Communication between .NET framework and Robot Studio

Chapter 7

Final Discussion and conclusion

In this paper the development of an optimal 3D path planner for a 9 DOF robot manipulator with collision avoidance is described. The path planner is based on the well known Probabilistic Roadmap method. The path planner is optimized for a 9 DOF ABB Robot manipulator moving within a limited and static environment with processing equipment, consisting of many pipes and tanks, as obstacles. The requirement that the robot manipulator should be able to move between and close to the obstacles is emphasized. The path planner is optimized as described below.

In the off-line Learning Phase the path planner was optimized due to *Coverage* and *Maximal Connectivity*. To fulfil these two criteria an appropriate sampling technique was required. Different sampling techniques were therefore studied, implemented and tested. It turned out that the greatest challenge was to cover the narrow areas in configurations space where the robot manipulator only can do small movements without crashing. Because of this the Bridge Sampling technique in combination with the Random Sampling technique provided best results and managed to cover all the test configurations.

When it came to *Maximal Connectivity* this was harder to achieve. Results showed that even if all the configurations were covered, it was not possible to connect all of them to the same connected component. It was especially the 3rd test configuration that was hard to connect. Because of this a method that improved the *Connectivity* called Connection Sampling was implemented and tested. The idea of this method is well known. The Connection Sampling methoh provdied *MaximalConenctivity* 3 ot of 5 times. The Connection Sampling method did decrease the number of connected

components in the roadmap every time which implies better *Connectivity*. The Connection Sampling method developed is therefore very useful, but not optimal.

The different steps in the Query Phase were studied in detail. An analysis showed that the most time consuming part was executing the local path planning when connecting the start and goal configurations. Sorting the nodes by distance and finding the shortest path using Djikstras algorithm had little impact on the query time. Many attempts to connect to different nodes increased the Query Time. Testing showed that the number of tries to connect decreased with an increasing number of nodes in the roadmap. A large roadmap does therefore not necessarily increase the query time, but rather decrease it compared to a smaller roadmap. Since Robot studio will be used in practice to do the local path planning, attempts to improve this part was not prioritized.

The optimization of the Query time was done in the path smoothing part. Some existing and a new developed method were implemented and tested. The new developed method, Deterministic Shortcut, is a modification of one of the existing methods, the Shortcut method. While the Shortcut method endeavour to make shortcuts with random lengths and at random places along the path, the Deterministic Shortcut methods endeavours to make "long" shortcuts first, and then smaller, around the waypoints along the path. Empirical testing showed that the Deterministic Shortcut method decreased the path length considerably more than the other techniques for a given time constraint.

A so called "Backup Procedure" was also implemented. The procedure is executed when a query fails. Experiments showed that the Backup Procedure manages to accomplish the queries, but the execution time varied a lot. Because of this it needs improvements to be useful in practice.

Use of the Potential Field planner as a more powerful local planner was also investigated. It turned out that use of a more powerful local path planner did not improve *Coverage* or *Connectivity*. The reason to this could be the limited performance of the Potential Field planner used due to insufficient tuning. If more time was spent on tuning and improving the Potential Field planner the result might have been improved.

In a real situation it is more likely that the goal is specified by a goal position and orientation instead of the entire goal configuration. Thus a local planner placing the end effector at a specified position in a specified orientation was developed and tested. This local planner, called the End Effector Planner, is a modified version of the Potential Field planner where the only attractive force is applied on the end effector and attracts it to its final position and

orientation. The End Effector Planner worked satisfactorily when only the goal position was specified, but struggled when the goal orientation also was specified.

To be able to test it on the real robot system at ABB in Oslo the path planner was attempt incorporated into Robot Studio. But due to unexpected issues it was not succeeded to finish the implementation within deadline. The issues were not related to the path planner program itself but to some unexpected behaviours from Robot Studio that needed to be dealt with. According to the staff at ABB it is just a matter of time before the implementation is done and experiments on the real system can be executed.

According to the results presented in this paper it is concluded that the Probabilistic Roadmap method is a suitable approach to obtain 3D path planner with collision avoidance for the 9 DOF robot manipulator. The *Coverage* can be optimized by using the Bridge Sampling technique in combination with a Random Sampling technique. The *Conectivity* can be improved by using Connetion Sampling, which could result in *Maximal Connection*. The path length can be optimized using the Deterministic Shortcut method.

Appendix A

Matlab Program

In this Chapter a brief explenation to the implemented Matlab code is presented. Only the purpose of the main functions are described.

The main file needed to be executed is PRM_Bridge. In this file are certain important parameter specified, such as the distance weight vector (and the joint Limits). This file calls a function, makeBridgeSamples_nearest_k, that generates samples and construct the roadmap. The makeBridgeSamples_nearest_k return a matrix called Samples, and a list of structure called Components.

The Samples matrix is a 11xN matrix that stores all the nodes in the roadmap. One column represents one node. Column 1 represents Node nr 1. Column 2 represents Node nr 2 and so on. The 10 first rows are the joint parameters and the 11th row describes which connected component the node belongs to.

The "'Components"' is a list of structures. Each item in the list is a structure describing one connected component. Each structure contains one list called "Node" and one matrix called "NB". The "Node" list contains all the node numbers in the connected component and is realted to column number in the Samples matrix. The "NB" matrix is a neighbour matrix describing the edges which connect the nodes together in the connected component. Zero represents no connection between two nodes while a value greater than zero represents a connection and the length between two nodes. The structures related to the Samples matrix in Table A.1 are listed below. The component structures show that node 1,2,3,6 and 7 are connected together in one component, while node 4 and 5 are connected together in another one. Node 8 has not managed to connect to any other nodes, and has therefore created its own component.

1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
x	x	x	x	x	x	x	x
y	y	y	y	y	y	y	y
z	z	z	z	z	z	z	z
θ_1	θ_1	θ_1	θ_1	θ_1	θ_1	θ_1	θ_1
θ_2	θ_2	θ_2	θ_2	θ_2	θ_2	θ_2	θ_2
θ_3	θ_3	θ_3	θ_3	θ_3	θ_3	θ_3	θ_3
θ_4	θ_4	θ_4	θ_4	θ_4	θ_4	θ_4	θ_4
θ_5	θ_5	θ_5	θ_5	θ_5	θ_5	θ_5	θ_5
θ_6	θ_6	θ_6	θ_6	θ_6	θ_6	θ_6	θ_6
1	1	1	2	2	1	1	3

Table A.1: Samples matrix containing 8 nodes

$$\text{Structure 1: Node} = \begin{bmatrix} 1 & 2 & 3 & 6 & 7 \end{bmatrix} \text{NB} = \begin{bmatrix} 0 & 1.45 & 4.32 & 0 & 0 & 0 & 0 \\ 1.45 & 0 & 0 & 0 & 0 & 0 & 4.53 \\ 4.32 & 0 & 0 & 0 & 0 & 0.76 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.76 & 0 & 0 & 0 & 0.2 \\ 0 & 4.53 & 0 & 0 & 0 & 0.2 & 0 \end{bmatrix}$$

$$\text{Structure 2: Node} = \begin{bmatrix} 4 & 5 \end{bmatrix} \text{NB} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.64 \\ 0 & 0 & 0 & 1.64 & 0 \end{bmatrix}$$

$$\text{Structure 3: Node} = \begin{bmatrix} 8 \end{bmatrix} \text{NB} = \begin{bmatrix} 0 \end{bmatrix}$$

When a new valid node is made, attempts to connect it to other nodes are executed. The order the new node tries to connect to the other nodes is determined by the distance between them. Obviously the closest nodes are tried out first. Because the chance of finding a feasible path between two nodes decrease as the distance between them increase, the distance and the number of nodes can therefore be limited. When a valid path is found the new node is connected to the node and its connected component by calling the "*connectNodeABB_K*" function. If the new node fails to find a valid path to other nodes, it makes a new component by calling the "*connectToEmptyComponentABB*" function.

Appendix B

Robot data sheets

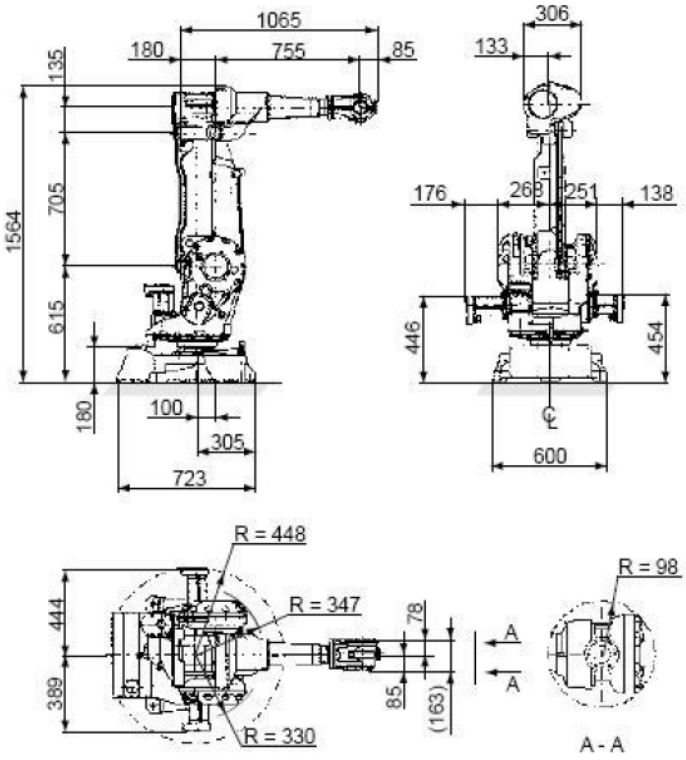
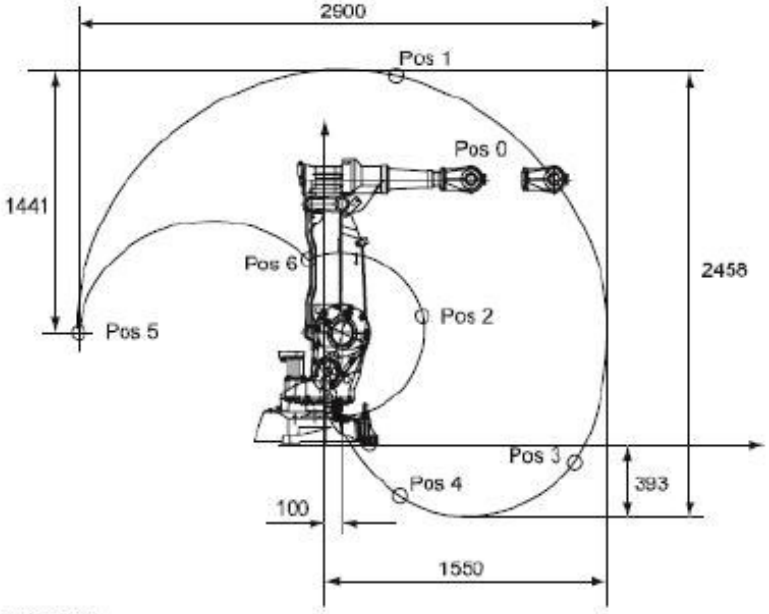


Figure B.1: Robot Data Sheet

IRB 2400/10, /16

The working area is the same for both floor and inverted (suspended). Positions are located at wrist center.



Pos.	X	Z	Angle axis 2	Angle axis 3
0	855	1455	0	0
1	360	2041	0	-60
2	541	693	0	65
3	1351	-118	110	-60
4	400	-302	110	18.3
5	-1350	624	-100	-60
6	-53	1036	-100	65

Figure B.2: Robot Data Sheet

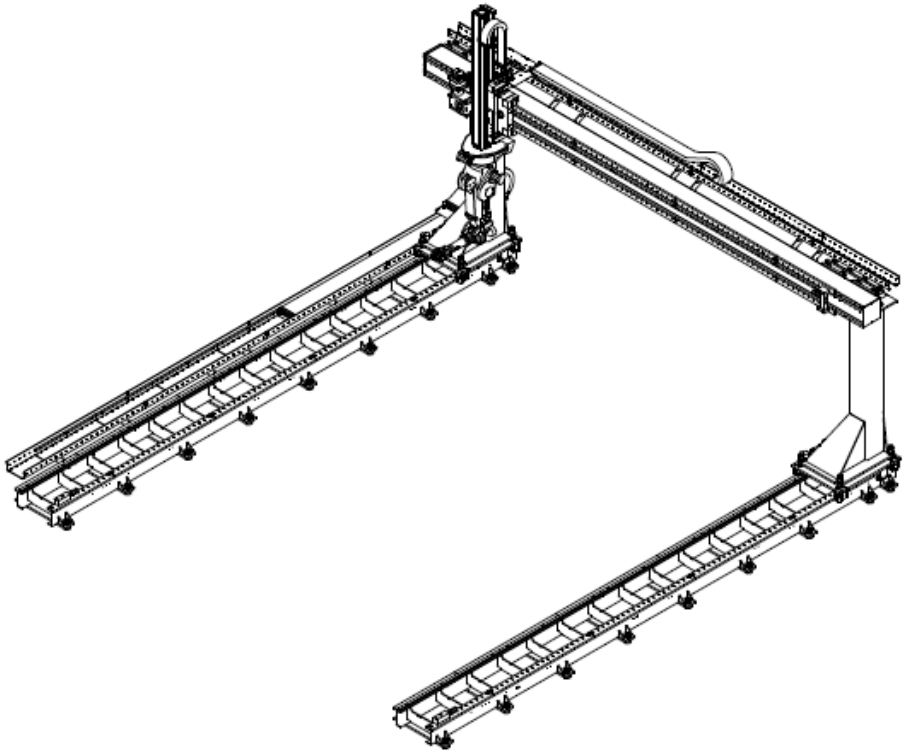


Figure B.3: Gantry Crane

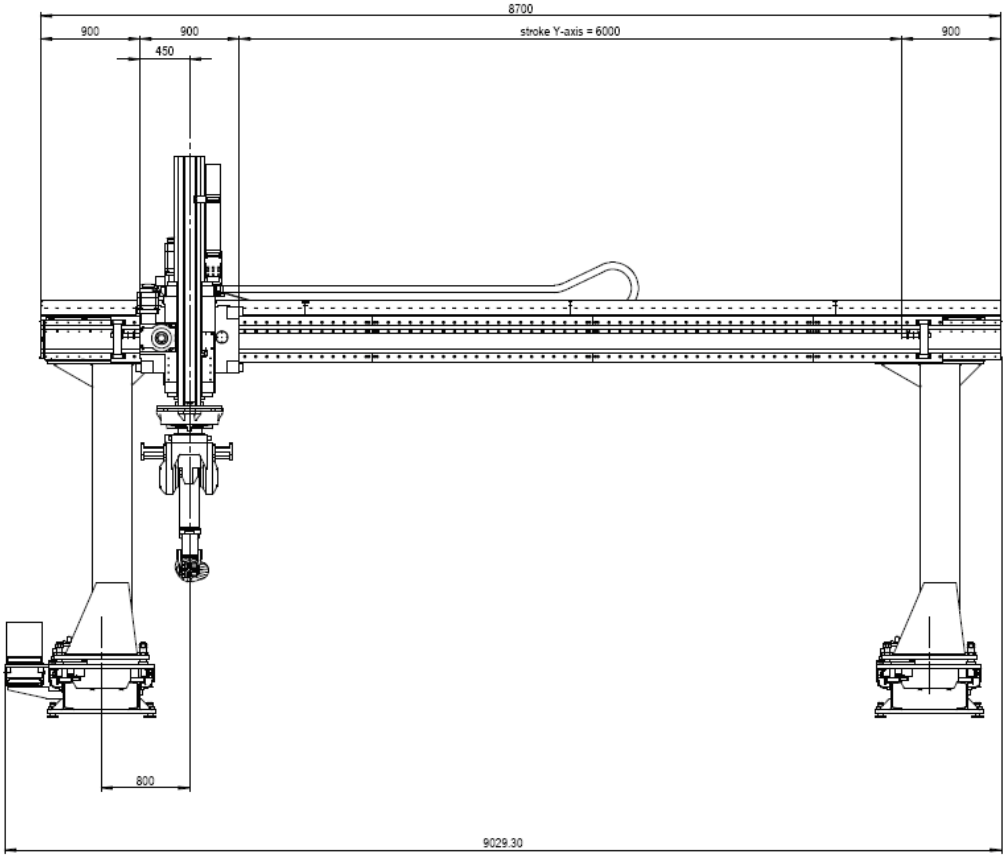


Figure B.4: Gantry Crane

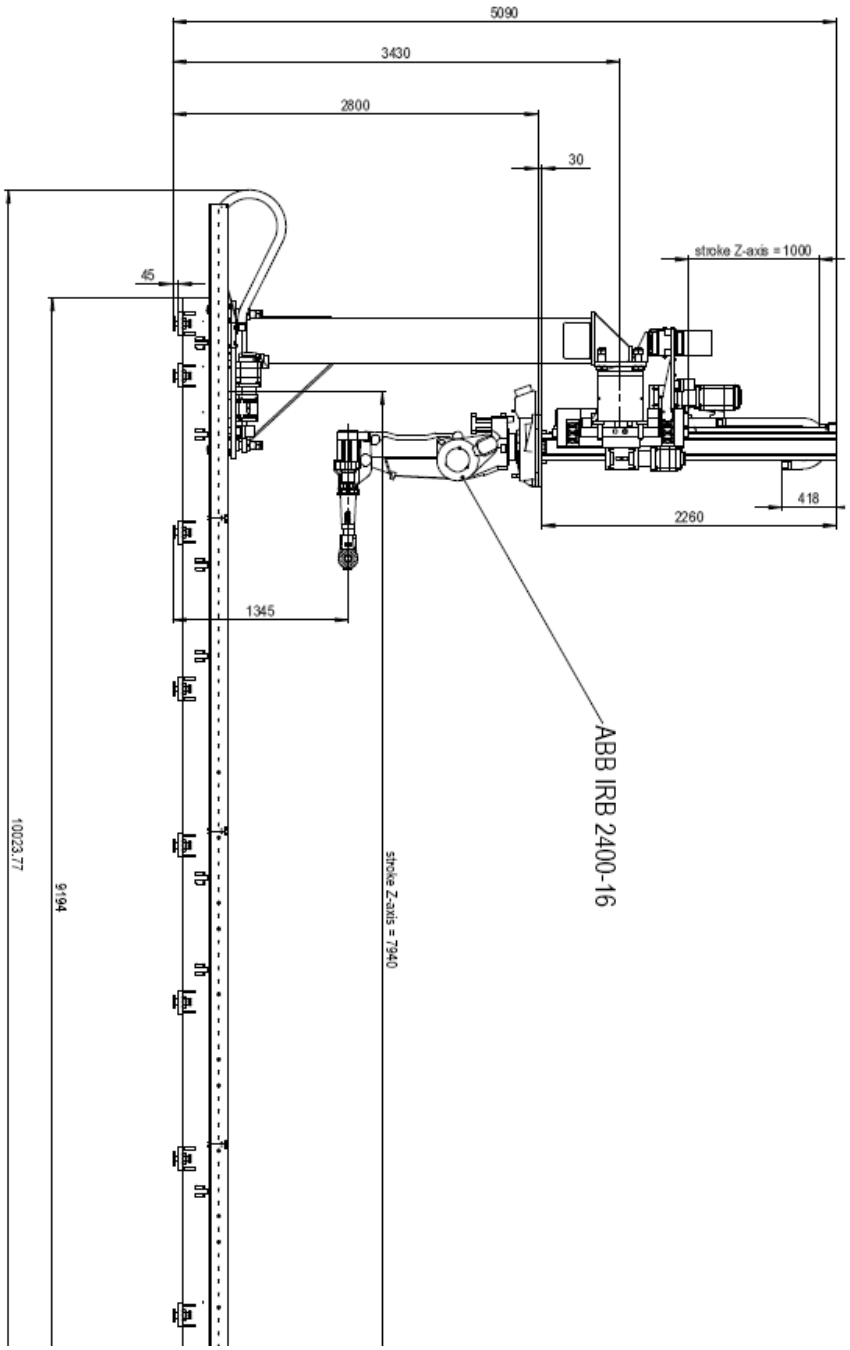


Figure B.5: Gantry Crane

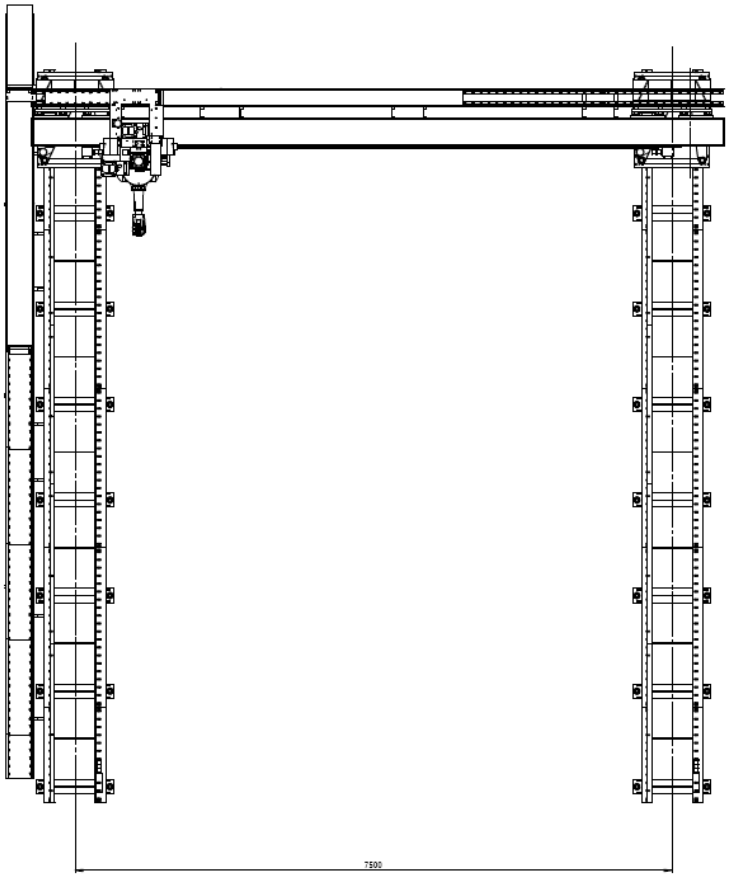


Figure B.6: Gantry Crane

Bibliography

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Trans. Robotics Automation*, 4(2):193-203, April 1988.

[LC91] Lin M. C. and Canny J. F. (1991) A fast algorithm for incremental distance claculation. In *Proceedings of International Conference on Robotics and Automation*, pages 1008-1014. IEEE

[MC93] Alistair Mclean, Stephen Cameron. Snaked-based path planning for redundant manipulators. In int. Conf. Robotics and Automation, volume 2, pages 275-282, Atlanta, May 1993

[MC94] Alistair Mclean, Stephen Cameron. Effective Path Planning and Collision Avoidance for Redundant Manipulators, Presented at *International Conference on Advanced Robotics and Computer Vision*, Singapotre 1994

[GD98] Kamal Gupta, Angel P. Del Pobil *Practical motion planning in robotics* Wiley, 1998

[NA98] Nancy M. Amato, O. Burchan Bayazit, et. al. Choosing Good Distance Metrics and Local Planners for Probabilistic Roadmap Methods. In *IEEE International Conference on Robotics and Automation*, pages 630-637, 1998.

[VB99] V. Boor, M.H. Overmars, and A.F van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 1018-1023, 1999.

[DH99] D.Hsu, J.-C. Latombe, and S.Sorkin. Placing a robot manipulator amid obstacels for optimized execution. In *IEEE International Symoposium on Assembly and Task*, pages 280-285, 1999.

[GB99] Gino van den Bergen. *A Fast and Robust GJK Implementation for Collision Detection of Convex Objects* July 6, 1999

[DH03] D. Hsu, t.Jiang, J. Reif and Z. Sun. The bridge test for sampling narrow passaages with probabilistic roadmap planner. In *IEEE International Conference on Robotics and Automation*, pages 4420-4426, 2003.

[GB04] Gino van den Bergen. *Collision Detection in Interactive 3D Environments*, Elsevier, San Francisco 2004

[CLH05] Howir Chuset, Kevin Lync, Set Hutchinson et al. *Principles of Robot Motion, Theory Algorithms, and Implementations*, MIT Press, 2005

[SHV06] Mark W. Spong, Seth Hutchinson, M. Vidyasagar. *Robot Modeling and Control*, Wiley, 2006

[RG06] Roland Jan Geraerts, *Sampling-based Motion Planning: Analysis and Path Quality*, Ph.D. thesis. Utrecht University. 2006.

[KAA07] Kristoffer Aasland, *3D Path Planning For a Robot Manipualtor With Collision Avoidance*, Project work fall, 2007, NTNU