Comparative Analysis of the Inverse Kinematics of a 6-DOF Manipulator

A Comparative Study of Inverse Kinematics for the 6-DOF Saab Seaeye eM1-7 Manipulator with Non-Conventional Wrist Configuration

Master Thesis, Spring 2023

Anton Larsson
Oskar Grönlund

ISRN:	**LiTH-ISY-EX–23/5597–SE**
Supervisor:	**Luca Claude Gino Lebon**, ISY
Examiner:	**Svante Gunnarsson**, ISY
External Supervisor:	**Gert Johansson**, Saab Dynamics

Linköping University
Division of Automatic Control
Department of Electrical Engineering
SE-581 83 Linköping, Sweden

Linköping University
Division of Automatic Control
Department of Electrical Engineering
SE-581 83 Linköping, Sweden

LINKÖPING
UNIVERSITY

Abstract

This report presents various methods for solving the inverse kinematic problem for a non-conventional robotic manipulator with 6 degrees of freedom and discusses their respective advantages and disadvantages. Numerical methods, such as gradient descent, Gauss-Newton and Levenberg-Marquardt as well as heuristic methods such as Cyclic Coordinate Descent and Forward and Backward Reaching Inverse Kinematics are discussed and presented, while the numerical methods are implemented and tested in simulation. An analytical solution is derived for the Saab Seaeye eM1-7 and implemented and tested in simulation. The numerical methods are concluded to be easy to implement and derive, however, lack computational speed and robustness. At the same time, the analytical solution overcomes the same issues, but will have difficulties in singularities. A simple path planning algorithm is presented which plans around singular intervals, making it viable to use the analytical solution without encountering problems with singularities.

Acknowledgement

We would like to express our gratitude to all parties involved in this project. We have encountered a lot of competent people both at the whole Saab concern as well at Linköping University who have shared their valuable input with us.

There are however a few persons' contributions we would like to particularly highlight.

Firstly, we would like to thank our supervisor at Saab Seaeye Joseph Molyneux-Saunders. Your engagement in the work aided us in understanding the technical aspects of the manipulator, but also how the results of this work could/should be implemented and used in real-world applications.

Secondly, we would like to thank our examiner Svante Gunnarsson and supervisor Luca Claude Gino Lebon at Linköping University. The feedback you granted us during this time made us feel comfortable with the direction of the work which in turn helped us reach our final results.

Lastly, we would like to thank the people who have assisted us at Saab Dynamics. Our manager Jonas Wallberg for making us feel welcome and helping us get settled in at the company. Jonathan Olofsson for your engagement as well as your aid in software-related issues that we stumbled upon during our work.

However, our deepest gratitude goes to our Saab Dynamics supervisor, Gert Johansson. Without your valuable input and sincerer desire for us to succeed, this project would not have been possible. Your enthusiasm and positive energy have motivated us to put extra effort into the work which ensured a great final result.

June 25, 2023
Anton Larsson & Oskar Grönlund

Nomenclature

Abbreviations

AUV	Autonomous Underwater Vehicle
CCD	Cyclic Coordinate Descent
DH	Denavit-Hartenberg
DHP	Denavit-Hartenberg Parameters
DLS	Damped Least Square
DOF	Degrees of Freedom
FABRIK	Forward and Backward Reaching Inverse Kinematics
HW	Hardware
IK	Inverse Kinematics
RHR	Right Hand Rule
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
$SE(3)$	Special Euclidean Group
SIM	Simulation

Miscellaneous

Base Frame	Frame of the manipulator base
Docker	Platform that allows applications to be packaged and run in isolated environments.
Gazebo	3D simulator for robotics
Pose	Position **and** orientation of an object/frame
RViz	3D visualizer for ROS
Tool Frame	Frame of end tool

Contents

1 Introduction

The introductory section of this report serves to provide a comprehensive foundation for the reader to understand the context of the research presented in this thesis. This section will begin by offering an overview of the field of robotic manipulators.

Subsequently, a descriptive analysis of the problem this thesis addresses will be presented. The purpose of this section is to provide a clear and thorough understanding of the problem being addressed and to set the stage for the subsequent sections that will present the proposed solution and the results of the research.

Lastly, the objective of the report is briefly addressed so that it is made clear what the main purpose of the proposed work is, as well as an overview of the robotic manipulator's current state.

1.1 Background

Underwater sea operations and maintenance work are essential for a variety of industries, including oil and gas, renewable energy and construction. However, such operations can be treacherous and potentially hazardous. Traditionally, the method of carrying out such operations has been to use divers equipped with all necessary equipment. This method poses a lot of dangers as the diver must endure and navigate through rough conditions, such as strong underwater currents, low visibility, extreme pressures and dangerous surroundings. Taking this into account means that divers sometimes may risk their lives for the sake of carrying out maintenance work.

However, due to the novel development of remotely operated vehicles (ROVs) and autonomous underwater vehicles (AUVs), these operations can often be carried out in a much safer and more efficient way, effectively reducing the risk associated with underwater operations. However, entirely using an ROV or AUV will oftentimes not be enough as it will require a robotic manipulator for carrying tools and equipment. These tools are often specially designed for the task at hand, which can be valve turning, bolt tightening, welding and pipe cutting, which in turn requires the manipulator to be equipped with multiple degrees of freedom (DOF). A manipulator is often controlled by an operator above sea level and there exist multiple methods of controlling the manipulator.

The subject of robotic control is a well-researched subject which involves the whole process of construction and design of the hardware to the actual control system, which controls the robot's movement. A robotic manipulator can consist of multiple joints controlled by actuators, making it possible for the manipulator to move in space. There exist mainly two approaches for controlling the manipulator's movement through space:

- Joint control
- Cartesian control

During joint control, an operator adjusts each joint until the end manipulator is at the desired pose. A Cartesian-controlled manipulator, however, requires only the manipulator's reference pose in Cartesian space while the joints adjust themselves to reach the desired pose. In order for a robotic arm to calculate each joint's reference angle to reach this position, the so-called inverse kinematic problem must be solved.

Inverse kinematics can be described as the opposite of forward kinematics, which answers the question: What is the end manipulator's pose, given some defined joint angles? The inverse kinematic answers the question: given a defined goal pose for the end effector, which joint angles are required to reach that goal pose? The inverse kinematic problem is a non-linear problem and not an easy problem to solve, as the complexity increases with the number of joints. It is therefore interesting to examine how the inverse kinematic problem can be solved for a 6-DOF robotic manipulator. There are multiple problems that arise when working with inverse kinematics, most prominently perhaps the fact there may exist multiple solutions, infinitely many solutions (referred to as singularity) and no appropriate solution at all.

1.2 Problem Description

In this thesis, the main objective is to develop and implement a robust (meaning it should be well-behaved and generate angles within joint limits, solutions close to the current configuration as well as not diverge in the proximity of singularities) inverse kinematic control scheme on a robotic manipulator which is equipped with 6 joints and 6 DOF. The algorithm should be able to control the manipulator with position control, meaning the input should be a pose and the output joint angles. Rather than implementing an arbitrary algorithm, this thesis aims to compare different methods and techniques to find an overall solution that gives a satisfactory behaviour of the manipulator whilst providing the best computational speed among other comparable methods. Hence, the problem description of the master thesis can be summarized by the following questions:

- What different techniques exist for solving the inverse kinematic problem for a 6-DOF robotic manipulator and how do they ensure numeric stability?
- How do different solving techniques perform in regard to crucial metrics, such as computational time and stability?
- In the case of an inverse kinematic algorithm, more often than not, there exist multiple solutions for the joint angles to reach a cartesian coordinate with the end effector. How do you choose an appropriate solution?
- When a goal point is found, the manipulator needs to follow a certain path to move from the initial state to the desired goal state. How can a feasible path be generated, with respect to singularities and computational time?

The main objective of the thesis is to develop a functioning algorithm for solving the inverse kinematic problem, making it possible to control the 6-DOF robotic manipulator in the Cartesian space with either velocity control or position control.

1.3 Manipulator Descriptions

The robotic manipulator on which this work is based is the all-electric Saab Seaeye eM1-7, a 6-DOF robotic manipulator intended for underwater usage. Although the eM1-7 is designed to be capable of a variety of different uses, a majority of the current intended tasks include maintenance on offshore oil and gas wells, while mounted to an underwater ROV. This maintenance is not seldom carried out on a depth of up to 3000 metres and includes delicate work on for example manifolds, Christmas trees[1] and wellheads. This intended work puts an extremely high priority on the repeatability and robustness of the control algorithm, as a mishap could result in catastrophic consequences.

Contrary to its competitors, the manipulator consists only of electric actuators, instead of hydraulic or pneumatic actuators, which is considered industry standard. The main idea for this is to achieve more accurate control and versatility as well as decrease oil leakages (which otherwise may lower the performance of the actuator as well as result in hefty fines for the company).

As underwater currents and other phenomena have an effect on the ROV's position, the manipulator should also be able to handle an unstable (moving) base reference, however, that will be beyond the scope of this thesis. Further on, the manipulator end tip should be able to handle a jaw, making it critical to be able to control not only the end position of the tool frame but also the orientation of the tool frame, this will be referred to as *pose*.

The manipulator consists of three elevation joints, two azimuth joints as well a wrist rotate joint. In a zero joint angle configuration, meaning that the encoders of each joint are set to zero, an elevation joint adjusts the upward movement of the manipulator (also referred to as *pitch*), an azimuth joint rotates the manipulator around in the ground frame (*yaw*) and the wrist joint rotate the tool frame itself (*roll*).

The design of the eM1-7 does not comply with the industry standard, described in [1], which means that the so-called "Pieper's solution" is not applicable when deriving the analytical solution for the inverse kinematic. This could result in numerical solutions being the only suitable solution for the inverse kinematic, making it even more relevant to investigate the suitability of those.

The main part of the project will be carried out, and validated/evaluated in a simulator which realizes a simulation as close to real-world scenarios as possible. The system architecture will briefly be described in the section below

1.4 System Architecture

The framework for controlling the manipulator is based on ROS Noetic, providing a fast and robust interface for development with C++ and Python code, as well as a CAN interface for controlling the manipulator's joints. The system architecture is structured in such a way it should be possible to control either a simulation model or the hardware model. The simulation is visualized in Gazebo and the CAN interface is simulated with a Linux kernel module "vcan" which can be read about here [2]. Furthermore, in order to control the manipulator in simulation, a GUI is provided and needs to be run on a separate PC connected via an Ethernet cable to the simulator computer. However, the GUI only enables control of specific joint positions and not end effector control in cartesian space, making it difficult to control the manipulator.

[1]A Christmas tree is a set of valves, spools and other mechanisms to control the flow from an oil well

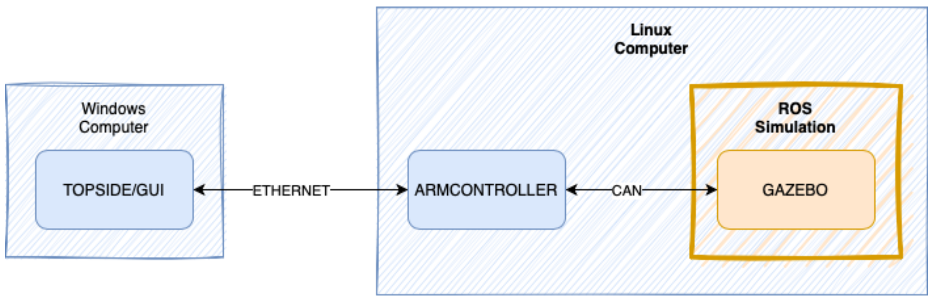


Figure 1: Caption

This architecture enables fast implementation and testing. If executed correctly, the system should be able to control the hardware without any major changes to the design and implementation of code and frameworks.

2 Related Work

This section provides an overview of existing research relevant to the current study. As previously stated, one key factor of this report is the evaluation and implementation of different algorithms that ensure safe travelling in the workspace without encountering any issues with regard to singularities (or rather dampening the impacts of singularities). On this subject, there is a lot of research work done, hence why it is key to compare the algorithms before full-scale implementation is started.

A lot of research dates back to the late 20th century about this matter and is based on numerical methods. What is often stated is however that numerical methods do not prove to be fast enough, which has made analytical methods a far more popular choice for solving the inverse kinematic problem. The drawback of an analytical solution, proven historically, is that it lacks a general solution as the closed-form expression is based on the design of the robotic manipulator. As an analytical solution (closed-form expression), historically has been proven to be faster than numerical methods, limited research into the use of numerical methods for real-time robotic systems has been published. Furthermore, that means that there may be no analytical solution possible if the manipulator is designed poorly.

The means to solve the Jacobian, which is needed in numerical methods, involves classic optimization such as *Gradient Descent*, *Gauss-Newton* and *Levenberg-Marquardt*. These methods are for non-real-time systems, as well as for graphical animation researched and established. They have proven to be capable of handling solutions in singularities but may lack solutions in the neighbourhood of singularities.

There also exist multiple heuristic methods which are favourable when working in computer animations where the accuracy of the manipulator is not deemed as important as the visuals. This is also a well-researched subject, but the implementation of these methods into real-time manipulators is not as well established.

Further on there exists multiple ways of computing and generating paths between two different poses. These methods often use interpolation between the poses, but generally do not take singular regions (which can be problematic) into account.

3 Method

This section provides a detailed description of the process and steps involved in finding the inverse kinematic control and subsequently how the implementation of said control scheme will be actualized.

The main aim of this chapter is to present a clear and concise explanation of the methodology and to provide the reader with a comprehensive understanding of the considerations that must be taken into account when implementing the solution. However, before the formulation of the inverse kinematic is derived, extensive research in the field of robotics is conducted. This includes well-established theories as well as more modern theories and research on the subject. Additionally, extensive research into the functionalities of Docker, ROS and MoveIt is conducted in order to realize the potential of these tools.

As the main part of the manipulator currently communicates over ROS, the final algorithms must be written in C++ and be able to be implemented by the tools presented. Furthermore, the MoveIt platform [3] is a platform developed for ROS, containing a lot of functionalities for robotic control. For this thesis, however, the algorithms developed will be included as plugins, making it possible to analyse the performance of the written algorithms in an otherwise functioning environment. This also implies that the "Topside" GUI and "Armcontroller" will be bypassed in the initial phase.

3.1 Mathematical Model

This part focuses on defining the inverse kinematics of the robotic manipulator. The manipulator's geometric properties act as a basis when defining the DHP:s (Denavit-Hartenberg parameters) for the manipulator, and subsequently also the frame transformations and Jacobian. This will mainly be based on the theory provided by John J. Craig in [4] and sets the framework for the forthcoming work. The code is initially written in Python as a proof of work and will, if proven functioning be translated into C++. The computed joint angles are at this stage compared to a simulated manipulator created with Peter Corke's "Robotics Toolbox" for Matlab, which not only presents a mathematical comparison but also a visual interpretation which can be used to validate the correctness.

3.2 ROS and MoveIt Implementation

Initially, at this step, a Docker image and container will need to be defined. Docker can be compared to a virtual machine, but presents multiple advantages. Docker containers share reassures with the host machine and the images (of which the containers are run) are much more easily exported, which means that other users only will need to install the docker image in order to run the simulation. The docker container should contain all software needed for the simulation to run smoothly. When the container is functioning, the simulation environment will need to be set up using ROS Noetic and the files provided by Saab Seaeye. These parts should make it possible to control the manipulator through joint control, either with a provided GUI or publishing topics for desired joint angles.

At this step, MoveIt will be configured and installed, making it possible to control the manipulator in cartesian space with the aid of MoveIt's pre-configured inverse kinematics solvers. The solvers we developed should be implemented as plugins, thereby the system will be structured according to Figure 2. Notice that MoveIt will provide the inverse kinematics plugin with the latest joint angles, making it possible to use that information as an initial guess for the numerical algorithms. MoveIt will additionally provide specific joint velocity for the joint's movement.

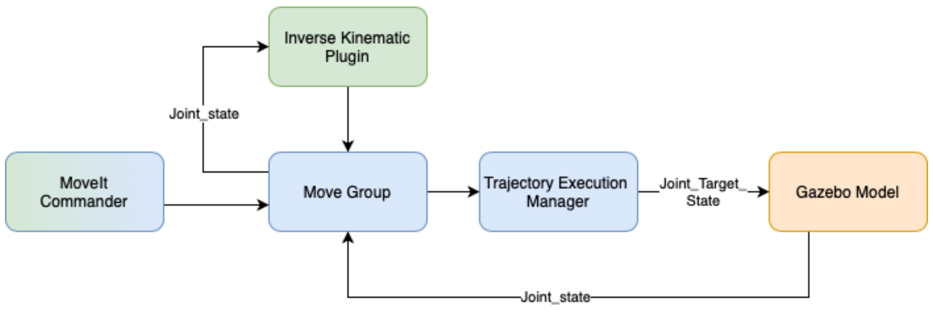


Figure 2: Simplified view of how the simulations will be configured. Green indicates developed by us and blue indicates MoveIt packages, while orange is simulation. Additionally, this architecture is all encapsulated inside the Docker container.

At this stage, the MoveIt package is configured and set up. MoveIt provides an interface which should make it possible to interact with the manipulator using a pre-defined inverse kinematics algorithm. When this setup is complete, the developed algorithms are implemented as plugins to MoveIt. They are then validated in simulation.

3.3 Path-Planning

The path planning will be developed for two reasons. Firstly, it makes it possible to control the manipulator without using MoveIt. Secondly, it opens up possibilities for creating a singular avoiding path, which may increase the convergence for the inverse kinematics algorithms. Initial testing of this will be carried out in Python and in Peter Corke's "Robotics Toolbox" for Matlab, making it possible to validate the algorithm.

4 Theory

In this section, theoretical descriptions of some subjects are gathered. These are the subjects which are deemed crucial for understanding the more advanced concepts later on in the report.

4.1 Coordinate Frames & Transforms

One returning aspect when discussing kinematic relations (not limited to manipulators) is coordinate frames and how one goes about describing some coordinate in another frame. This process can also be seen as finding the transformation between two different coordinate systems.

More precisely, frame transformations refer to the process of mapping vectors or points from one coordinate system to another systematically. Frame transformations involve the use of linear transformations, represented by matrices, to describe the orientation and position of one coordinate system relative to another. The matrices used to perform these transformations are known as rotation matrices and translation matrices, which then can be neatly described together within what is called a homogeneous transformation matrix.

4.1.1 Rotation Matrices

There exist multiple methods of representing the orientation of a body in space, all with different shortcomings and advantages. This includes Euler angles, quaternions and *rotational matrices*, which mainly will be used in this work.

A rotation matrix, represented by a 3×3 matrix, defines the linear transformation that maps vectors from one coordinate system to another. i.e. a rotational matrix describes how a frame is oriented relative to a reference frame. Let's call the B, the rotation matrix describing how frame B is oriented relative to frame A would be denoted ${}^{A}_{B}R$.

In turn, the rotational matrix is constructed by three separate rotations around each individual axis, the angles of which the rotational matrices represent can be seen in Figure 3.

$$ {}^{A}_{B}R = {}^{A}_{B}R_x \, {}^{A}_{B}R_y \, {}^{A}_{B}R_z \tag{1}$$

$$ {}^{A}_{B}R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \tag{2}$$

$$ {}^{A}_{B}R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{3}$$

$$ {}^{A}_{B}R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The resulting matrix columns can each be interpreted as a representation of each of the axes relative to the origin frame, i.e. the first row of the matrix denotes the orientation of the x-axis in frame B relative to frame A and so forth. As described in [5], it is known that due to the cosine rule the angle between the old and new x-axes (and similarly for y and z) is in fact the diagonal element corresponding to the axis (for x it's the first, y the second and for z the third element of the diagonal).

Effectively, this results in a simple way of representing a vector ${}^{B}P$ expressed in the coordinate system of **B**. Multiplying the vector ${}^{A}P$ with the rotational matrix ${}^{A}_{B}R$, results in:

$$ {}^{A}P = {}^{A}_{B}R\, {}^{B}P \tag{5}$$

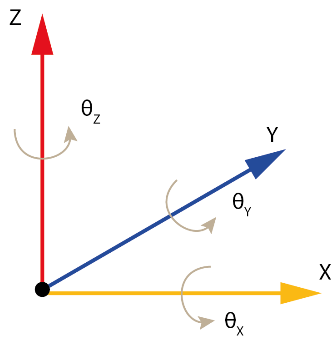


Figure 3: Visual representation of the rotation around each axis

4.1.2 Translation Matrices

By the previously defined rotational matrices, rotational relations between two different coordinate frames can be described. If the transformation of a coordinate frame includes some sort of displacement instead one wants to use what is called a translational matrix.

Aa explained by John J. Craig [6] translational matrix describes the displacement of frame A relative to frame B when their orientation coincides, but their respective origins do not. In order to describe this relation, a vector ${}^{A}P_1$ is used. It describes how the origin of B is positioned relative to the origin of A and can be represented by a 3×1 vector.

Let's denote a vector ${}^{B}P$, expressed relative to frame B. Spong and Vidyasagar [4] explain that in order to represent this vector in the coordinate system of frame A, it is just the case of adding them.

$$ {}^{A}P = {}^{B}P + {}^{A}P_1 \tag{6}$$

4.1.3 Homogeneous Transformation

In the previous two sections, the transformation of a coordinate frame was described when it occurred only as either a displacement or a rotation. In the case of robotic manipulators, the coordinate transform almost exclusively will need to account for both relative orientation and relative position. One way to represent a transformation of a complete rigid-body configuration is by using a homogeneous transformation matrix representation. According to Kevin M. Lynch and Frank C. Park [7], the set of all real Homogeneous Transformations matrices is called the "Special Euclidean Group" or SE(3) which these types of transform oftentimes are denoted as.

Essentially, the SE(3)-transform is a combination of the previously described rotational and translational transforms, and it can be denoted in the following manner:

$$T = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

The bottom row of the matrix is essentially only included to make the matrix square and simplify matrix operations.

When working with the SE(3)-transforms to properly describe the manipulator, it usually exists multiple transforms in sequence within a kinematic chain, to properly describe these relations one can use the fact that the SE(3)-transforms works in the same way as rotational matrices in multiplication, i.e.:

$$^{0}_{N}T = \prod_{i=1}^{N} {}^{i-1}_{i}T \tag{8}$$

4.2 Singularities and Workspace Limitations

When working with robotic manipulators, it is crucial to understand singularities within the manipulator configuration space and how these affect the resulting movements of the manipulator.

Essentially, a robotic manipulator is in a singular configuration (or point) when the configuration limits the possible movements of the arm, i.e. there is a loss of degrees of freedom. In a practical sense, there are tell-tales (when studying the real manipulator movement) of when a singularity has occurred: examples are when the manipulator appears to be unable to move in a given direction or when some joints move extremely quick.

Where these singular points are located within the workspace is heavily dependent on the configuration of the manipulator. However, there is a common way of categorizing singularities within the workspace, mainly dividing between what is called boundary and interior singularities.

A boundary singularity can essentially be seen in Figure 4 where the manipulator is stretched out to its fullest. This singularity is characterized by an impairment of the end effector's movement. The main occurrence of these singularities is elbow singularities. Essentially the manipulator will stretch the arm to its maximum reach, i.e. to the workspace boundary. This point can be explained as the transition between two configurations yielding the same end-effector pose, more precisely elbow-up and elbow-down configurations[2], within the singular point these configurations will look identical.

Interior singularities, often denoted as wrist singularities and shoulder singularities, are categorized as when movement is impaired whilst the endpoint of the manipulator is located within the boundary of the manipulator workspace.

[2]Elbow-up/Elbow-down configuration is exactly like it sounds, the elbow joint of the manipulator can be either in an upright or downright position whilst finding the same final solution for the end-effector.

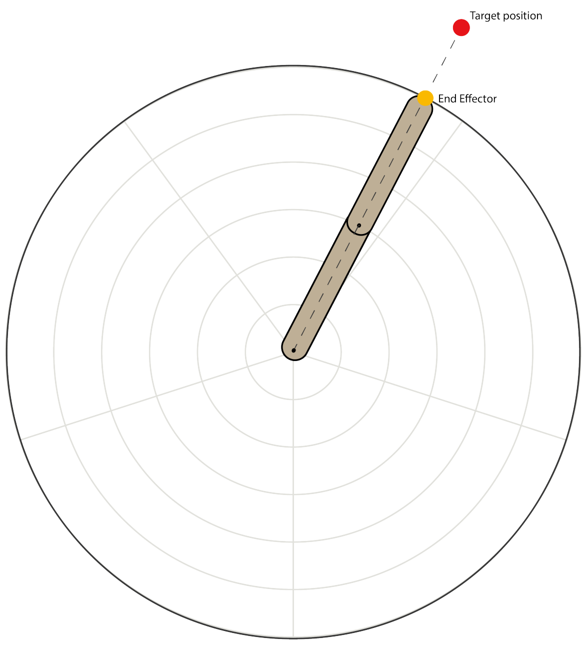


Figure 4: Illustration of the workspace for a 2 DOF robotic manipulator in 2D space

A more mathematical description of the singularities that occurs for the robotic manipulator gets linearly dependent rows, i.e. the rank of the Jacobian drops below the #DOF for the manipulator, seen in Equation (9).

$$\text{rank } J < \#DOF \tag{9}$$

This means that the velocities of the robot's joints cannot be uniquely determined from the velocity of the end-effector. In other words, the Jacobian matrix loses one or more degrees of freedom at the singularity. As described by M. W. Spong and M. Vidyasagar in [4], a common practice to find out whether the manipulator is in a singular configuration is to study the determinant of the Jacobian. If a singular configuration is present, the determinant of the Jacobian will result in the value zero, this implies that the Jacobian matrix is singular which in turn implies that the manipulator is in a singular configuration.

$$\det(J) = 0 \tag{10}$$

5 System Model

This section includes a kinematic representation of the manipulator, an introduction to the Denavit-Hartenberg convention as well as a numerical approximation of the system.

5.1 Kinematic Representation

The representation of a robotic manipulator can be described in a multitude of ways. For example, it can be described as explained in the introduction via its joint configuration or with specific names that describe the kinematic relations in the manipulator (ex. Elbow Manipulator). For the purpose of this report, it is most beneficial to study the kinematics representation of the manipulator, since this is a necessary foundation for ultimately deriving the inverse kinematics. There is a multitude of ways in which a manipulators kinematics can be represented, hence it is of high importance to understand what's required from a kinematic model to be deemed satisfactory.

In the paper [8] by Ruibo He et al, three basic requirements are defined that a kinematic-parameter identification should meet:

- Completeness: A complete model must have enough parameters to describe any possible deviation of the actual kinematic parameters from the nominal values.
- Continuity: Small changes in the geometric structure of the robot must correspond to small changes in the kinematic parameters. In mathematics, the model is a continuous function of the kinematic parameters.
- Minimality: The kinematic model must include only a minimal number of parameters. The error model for the kinematic calibration should not have redundant parameters.

There are a plethora of proposed models that satisfy these requirements, some examples are Hayati model[9], Veitschegger and Wu's model[10] and the "Complete and Parametrically Continuous" (CPC) model[11].

While these requirements (and the subsequent models) present a good way to evaluate a kinematic representation it slightly overlooks discrete alternatives (since continuity is required), which can significantly lower the complexity of the representation whilst still providing satisfactory results. Actually, the most widely used kinematic representation of manipulators, the Denavit-Hartenberg convention, is not continuous but does satisfy the requirements of completeness and minimality.

Denavit-Hartenberg convention may not present a continuous way to describe the kinematics, but it is advantageous in other ways, namely:

- Simplicity: Derivation of the DH parameter method is simpler than its counterparts.
- Computational Weight: Due to the discrete nature of the DH-parameter method its calculation process is simpler, which in turn leads to less computational effort if used in iterative optimization algorithms.
- Versatility: The method is highly versatile and can disregard the structural order and complexity of the robot.

- Well-documented: Due to the fact that DH-parameterisation is the most used method it has been vigorously tested and documented.

For this master thesis, the model that was chosen was the Denavit-Hartenberg convention. The method was chosen based on the aim of this master thesis, since the thesis is looking to find inverse kinematics solutions both numerically and analytically it is of great benefit that the kinematics model is computational "light". Furthermore, seeing as the DH convention is widely used, it makes for a more trustworthy end result which is crucial if the algorithms were to be implemented on hardware in the future.

For a more in-depth description of the method one can refer to most introductory literature on robotics, some examples are the textbooks written by John J. Craig [6] and Saeed B. Niku [12].

5.2 Denavit-Hartenberg Convention

This subsection will describe in detail how the DH-Convention can be used to construct a kinematic representation of the manipulator. This procedure is done in several steps, which will be presented in the order in which they are to be executed.

5.2.1 Assignment of the Joint Coordinate Frames

The initial part of deriving the mathematical description of the manipulator according to DH convention is to define the relevant coordinate frames for the manipulator.

As described by B. Spong and Vidyasagar [4] a robotic manipulator consists of multiple joints and links in sequence. The joints allow for rotational movement around the joint axis. As a manipulator consists of multiple joints and links in sequence, we need to describe their relative movement with the help of frame transformations. This means that one frame for each link must be defined (referred to as link frames). Assigning these frames to the links and denoting them can be done in multiple ways, however in order to comply with the DH convention, we will have to follow a specific set of rules.

The first frame, more commonly known as the base frame, is usually designated as the reference frame and is placed at the base of the mechanism (hence the name) as seen in Figure 5. Every following frame i should thereafter be constructed according to the following principles described by B. Spong and Vidyasagar [4]:

- The Z_i-axis should be placed in the positive direction of the joint rotation decided by the rotational RHR (right-hand rule), which states that if the thumb of the right-hand points along the direction of the line of action, the fingers curl in the direction of the positive axis.
- The X_i-axis should be defined along the common normal of Z_i and Z_{i-1}, pointing away from the previous frame. Should both axis be parallel, there is an infinite amount of common normals between Z_i and Z_{i-1}, thus the placement of X_i is arbitrarily as long as it is perpendicular to both Z_i and Z_{i-1}.
- The Y_i-axis should complete the frame in accordance with the RHR, which states that if the thumb of the right-hand points along the direction of the X-axis, the index finger points along the direction of the Y-axis and the middle finger points along the Z-axis. Since the X-, and Z-axis are already defined, this will yield the direction of the Y-axis.

- If the following rules are satisfied, the placement of the frame origin should be elementary.

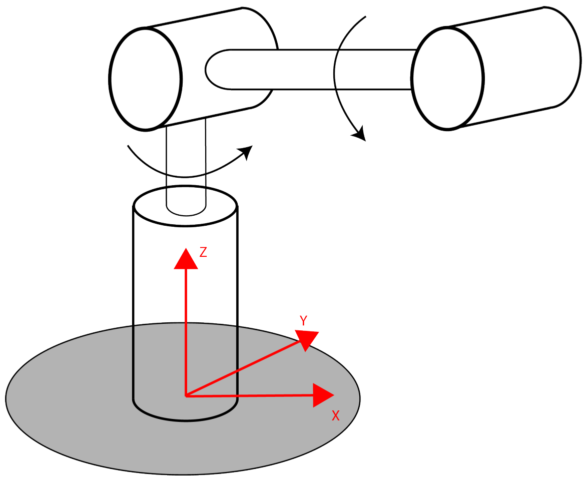


Figure 5: Illustration of how the base frame is placed on a generic manipulator.

This method of constructing the frames within the DH-Convention scope is valid in most cases, there is however a special case which often presents itself, which is when two sequential rotational axes intersect one another. In this case, the previously mentioned rules do not apply. When defining a frame for intersecting rotational axes, we instead want to take the following principles into consideration which are presented in B. Spong and Vidyasagar [4]:

- The Z_i-axis should be defined as previously mentioned, which creates an intersection with the Z_{i-1} axis.
- The X_i-axis should be defined as a normal to the plane containing the Z_i and Z_{i-1} axes. This presents multiple options since the axis can be defined as normal to the plane with two different directions. Which direction is chosen depends on the situation (i.e. which direction leads to simpler calculations moving forward for the specific manipulator type), although both are valid.
- The Y_i-axis should complete the frame in accordance with the RHR.
- The origin of this frame should be placed in the origin of the last frame, this placement is important to enable satisfy derivation of the DHP moving forward.

Lastly, at the end of a robotic manipulator, it is presumed that we will have some sort of tool attached, for example, a gripper. It is common practice that the origin of the last frame (oftentimes recognized as the tool frame) is placed in the centre of the tool, meaning that in the case of a gripper, it lies in the centre of the "fingers" of the gripper. Of course, this will differ slightly depending on the application (i.e. which tool is used).

In accordance with the described convention, the frames of the robotic manipulator can be seen in Figure 6.

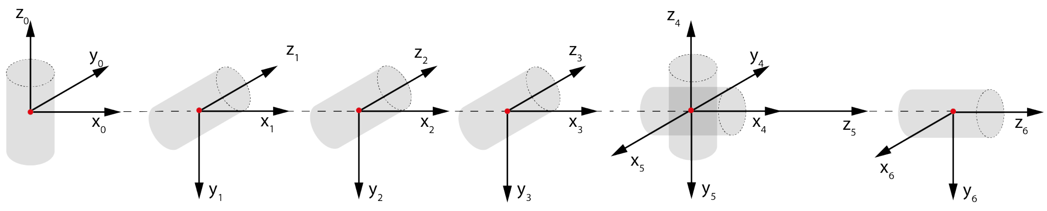


Figure 6: Illustration of how the frames were placed for the robotic manipulator

5.2.2 Finding the DH-Parameters

By making the assumption that the frame definition in the previous section was executed properly, then the DH-Convention provides an intricate way to describe the kinematics of the robot links by the use of four parameters.

This is one of the major advantages of the DH convention, that it is possible to describe the full orientation of each manipulator link with fewer parameters than what one would find in other common conventions. In the standard convention, orientation is represented as a displacement vector which includes three components (x, y, z) and a rotational matrix including the three Euler angles. This is equivalent since we have a considerable amount of freedom in choosing the origin and coordinate system of each link frame, even though the frame i still needs to be rigidly attached to the link i (this method of choosing the frames was explained in the previous section). Thus, by choosing the frame and origin in a clever way, the number of parameters needed can be decreased. In turn, this results in a more compact representation of the manipulator, in the case of 6-DOF manipulators DH-convention yields only 24 independent variables instead of the 36 independent variables present in the standard approach. For a full-scale proof of this concept, please refer to the textbook "Robot Dynamics and Control" [4] by M. W. Spong and M. Vidyasagar.

The four parameters which are present in the DH convention can be described in the following manner:

- θ_i = Joint angle from X_{i-1} to X_i about Z_{i-1}, referred to as joint variable
- α_i = Angle from Z_{i-1} to Z_i about X_i
- r_i = Distance between Z_{i-1} to X_i along Z_i
- d_i = Joint distance, distance between Z_{i-1} to X_i along Z_{i-1}

In addition to the regular DH-Parameters, a supplementary variable has been added which denotes the joint-angle offset in the zero-configuration. What this describes is that when a specific joint angle is applied to the joint, it needs to be reduced by a pre-set angle which is already set in place to align the robot to its "rest" configuration.

By extract the described parameters, from the previously defined coordinate frames (Figure 6), the manipulator can be described according to Table 1:

i	θ [rad]	α [rad]	r [m]	d [m]	Offset [rad]
1	θ_1	$-\frac{\pi}{2}$	r_1	0	0
2	θ_2	0	r_2	0	0
3	θ_3	0	r_3	0	0
4	θ_4	$\frac{\pi}{2}$	r_4	0	0
5	θ_5	-$\frac{\pi}{2}$	r_5	0	$-\frac{\pi}{2}$
6	θ_6	0	r_6	0	0

Table 1: Table of derived DH-parameters, some parameters not shown due to secrecy

5.2.3 Transformation Matrices

Furthermore, the DH-Convention introduces a direct way of finding the Homogeneous transformation matrices that describe the kinematic linkages of the manipulator, given that the previously described steps have been accomplished. By inserting the values for each of the manipulator's joints into the matrix shown in Equation (11)[3]:

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & s\theta_i c\alpha_i & s\theta_i s\alpha_i & r_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & d_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11} $$

By inserting the DH-parameters (previously denoted in Table 1) the following transformations between each of the manipulator frames are achieved:

$$ {}^{0}_{1}T = \begin{bmatrix} c\theta_1 & 0 & 0 & c\theta_1 r_1 \\ s\theta_1 & 0 & c\theta_1 & s\theta_1 r_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12} $$

$$ {}^{1}_{2}T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & c\theta_2 r_2 \\ s\theta_2 & c\theta_2 & 0 & s\theta_2 r_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13} $$

$$ {}^{2}_{3}T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & c\theta_3 r_3 \\ s\theta_3 & c\theta_3 & 0 & s\theta_3 r_3 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14} $$

$$ {}^{3}_{4}T = \begin{bmatrix} c\theta_4 & 0 & s\theta_4 & c\theta_4 r_4 \\ s\theta_4 & 0 & -c\theta_4 & s\theta_4 r_4 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15} $$

$$ {}^{4}_{5}T = \begin{bmatrix} c\theta_5 & 0 & -s\theta_5 & c\theta_5 r_5 \\ s\theta_5 & 0 & c\theta_5 & s\theta_5 r_5 \\ 0 & -1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16} $$

[3] $c = cosine$ and $s = sine$

$$
{}^{5}_{6}T = \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & c\theta_6 r_6 \\ s\theta_6 & c\theta_6 & 0 & s\theta_6 r_6 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}
$$

By using the relation between each of the manipulator frames the relation between joint angles and cartesian positions can be defined via the means of multiplication. The resulting matrix is more commonly known as the Forward Kinematics.

5.3 Numerical Approximation via the Jacobian

While the forward kinematics equations relate the position of the end-effector to the joint angles, they do not directly provide information about the velocities of the end-effector and how they are related to the velocities of the joints. This part is crucial in calculating the inverse kinematics of a manipulator since by relating the velocities of the end-effector to the velocities of the joints, it realizes the possibility to iteratively adjust the joint angles until the desired position and orientation are achieved (which is the used practice when applying different forms of numerical inverse kinematics). One way to describe this relation approximately is from the Jacobian.

The Jacobian is a mathematical concept used to analyse the relationship between two sets of variables. Specifically, it represents the derivative of a vector-valued function with respect to variables used as inputs. E.g, if you have a function that takes multiple variables as inputs and produces a vector as output, the Jacobian matrix describes how the elements of the output vector change with respect to the elements of the input vector. This also means that each element in the Jacobian can be seen as a partial derivative
The Jacobian can thus be used to find the relation between the velocities of the end-effector and the velocities of the joints. By taking the partial derivatives of the forward kinematics equations with respect to the joint angles, the Jacobian matrix describes how small changes in the joint angles correspond to changes in the position and orientation of the end-effector.

To understand how the Jacobian relates to solving the inverse kinematics in a numerical sense, it is important to note that the kinematic relations previously derived are of a non-linear nature. Hence, finding the correct joint-angle configuration from a specific pose of the end-effector essentially consists of finding the solution to a non-linear optimization problem.

Non-linear optimization problems have long been solved via the use of numerical methods. There exist multiple different numerical methods, but the majority uses Taylor expansions in order to get a first or second-order system seen in Equation (18).

$$
f(x) = f(a) + f^{'}(a)(x-a) + \frac{f^{''}(a)}{2}(x-a)^2 + \mathcal{O}((x-a)^3) \tag{18}
$$

By using the Taylor expansion of a non-linear function, you effectively end up with an approximation of the function. We will see further that some numerical methods only make use of the first-order derivative, while some make use of the second-order derivative. In order to derive the general formula we will for now continue with the first-order system.

$$
f(x) = f(a) + f^{'}(a)(x-a) + \mathcal{O}((x-a)^2) \tag{19}
$$

In this context (i.e. non-scalar) and with correct notation, the Taylor expansion would look accordingly:

$$f(\theta_d) = f(\theta_i) + \left(\frac{\partial f(\theta)}{\partial \theta}\right)\bigg|_{\theta_i} (\theta_d - \theta_i) + \mathcal{O}((\theta_d - \theta_i)^2) \tag{20}$$

Where θ_d denotes the desired state while θ_i denotes the current state which the linearization is made around. Solving for $(\theta_d - \theta_i)$ Equation (18) a while excluding the higher order terms one will end up with:

$$(\theta_d - \theta_i) = \left(\frac{\partial f(\theta)}{\partial \theta}\right)^{-1}\bigg|_{\theta_i} (f(\theta_d) - f(\theta_i)) \tag{21}$$

And simplifying the notation yields Equation (22) where the Jacobian Inverse presents itself as the relation between the joint angles to the end effector pose.

$$\Delta\theta = J^{-1}(\theta)\Delta X \tag{22}$$

In the equations, ΔX is the error vector for the desired pose and current pose, while the Jacobian represents the following matrix:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \cdots & \frac{\partial x}{\partial \theta_n} \\ \frac{\partial y}{\partial \theta_1} & \cdots & \frac{\partial y}{\partial \theta_n} \\ \frac{\partial z}{\partial \theta_1} & \cdots & \frac{\partial z}{\partial \theta_n} \\ \frac{\partial \theta_x}{\partial \theta_1} & \cdots & \frac{\partial \theta_z x}{\partial \theta_n} \\ \frac{\partial \theta_y}{\partial \theta_1} & \cdots & \frac{\partial \theta_y}{\partial \theta_n} \\ \frac{\partial \theta_z}{\partial \theta_1} & \cdots & \frac{\partial \theta_z}{\partial \theta_n} \end{bmatrix} \tag{23}$$

Which (in the field of robotics) can be derived in the following manner:

$$J = \begin{bmatrix} R_{i-1}\begin{pmatrix}0\\0\\1\end{pmatrix} \times (d_n - d_{i-1}) \\ R_{i-1}\begin{pmatrix}0\\0\\1\end{pmatrix} \end{bmatrix} \tag{24}$$

In Equation (24) R_{i-1} represents the rotational matrix while d_{i-1} and d_n represents the current and final end effector position respectively.

As it proclaims to the linearization it is required to take the inverse of said Jacobian. This action presents some issues with numerical stability in the case where the Jacobian is non-invertible. This is true for a Jacobian which describes the relations between cartesian- and joint velocities of a robotic manipulator when the rank of the Jacobian is less than the degrees of freedom (rank $J < \#DOFs$). This is known as a *singularity* (The effect of singularities in the field of robotic manipulators is explained more in-depth previously in Section 4.2).

6 Numerical Concept

The numerical concepts all strive to minimize the error between the target pose and the current pose according to Equation (25). This is achieved by iterating through configurations of joint angles in a controlled manner. The iterating process as well as the method of approximating the Jacobian inverse is what sets the different algorithms apart. The problems with the Jacobian inverse are explained in greater detail in Section 5.3 and the different methods of computing the Jacobian inverse can be seen in the coming sections.

$$\begin{aligned} \underset{\theta}{\text{arg min}} \quad & ||T_{end} - H(\theta)|| \\ \text{subject to} \quad & \theta \geq \theta_{min}, \\ & \theta \leq \theta_{max} \end{aligned} \tag{25}$$

Where T_{end} is the end effector's desired pose and $H(\theta)$ is the calculated pose, given current joint angles, θ_{min} and θ_{max} are the minimum and maximum allowed joint angles for each joint respectively

The iterative process is in general the same for all three analysed methods. Figure 7 presents the process by which the algorithm iterates through poses. The yellow coloured boxes indicate in which steps the different algorithms differ from each other.

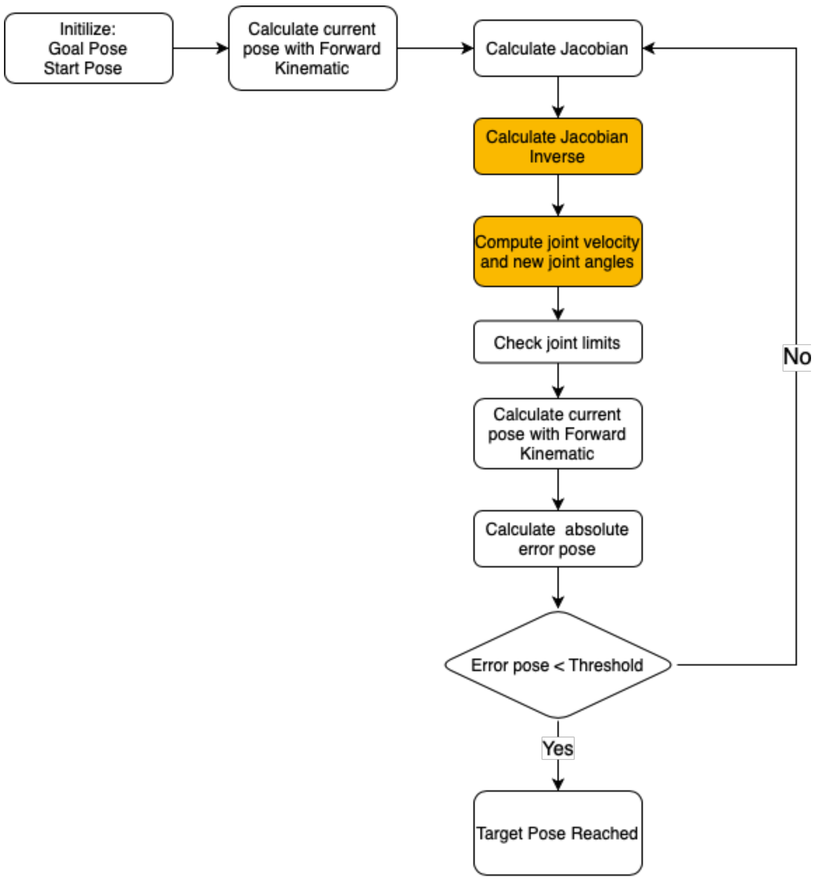

Initilize:
Goal Pose
Start Pose
Calculate current pose with Forward Kinematic
Calculate Jacobian
Calculate Jacobian Inverse
Compute joint velocity and new joint angles
Check joint limits
Calculate current pose with Forward Kinematic
Calculate absolute error pose
Error pose < Threshold
No
Yes
Target Pose Reached

Figure 7: Flowchart for a numerical algorithm

The steps can be summarized accordingly:

1. The goal pose is initialized by the user input. This can for example be a specific pose in space or a pose in a trajectory. The start pose can essentially be chosen arbitrarily, however, it is generally speaking often set as the last known pose for the end effector.
2. The current end-effector pose is calculated with the forward kinematics.
3. The Jacobian is calculated according to Equation (24).
4. The inverse, or rather an approximation of the Jacobian inverse, is calculated depending on which algorithm/method is used.
5. New joint angles are calculated according to the specific method.
6. The joints are adjusted to the maximum/minimum allowed value if they are above/under their limits
7. The temporary pose for the calculated joint angles is calculated.

8. The error pose is calculated and the norm of it is used as the absolute error.
9. A check whether the calculated pose is close enough to the goal pose is done.
    (a) If the pose is not close enough to the goal pose, the Jacobian is again calculated for the new joint angles and the process repeats
    (b) If the current pose is deemed close enough to the goal pose, the algorithm is finished.

Additionally, it should be noted that all numerical methods are prone to get stuck in local optima, which makes them sensitive to their initial guesses of joint angle configurations. This means that one has to be certain to specify a good initial guess in order to gather a satisfactory result.

6.1 Dependence on the Jacobian Inverse

As mentioned in Section 5.3, the Jacobian presents a way to relate the velocities in the joint space to the cartesian space. Although, the problem is not that simple since the manipulator may find itself in singular configurations in many regions within the valid workspace. Singular configurations result in the Jacobian inverse becoming numerically unstable (according to Section 5.3), thus a way to approximate the Jacobian inverse is required to have a feasible solution to the numerical methods. There are two approaches which often are used to approximate the inverse of the Jacobian which are the Moore-Penrose pseudo inverse and the Jacobian Transpose.

- Jacobian Pseudo Inverse
$$\Delta\theta = J^{\dagger}\Delta X \tag{26}$$
- Jacobian Transpose
$$\Delta\theta = J^{T}\Delta X \tag{27}$$

6.2 Gradient Descent

The gradient descent method (in robotics, often referred to as *Jacobian transpose method*) is a numerical approach to solve a set of n non-linear algebraic equations with n unknowns. It was used in order to solve the inverse kinematic problem in 1984 by both Balestrio [13] and Wolovich [14]. The basis for this method lies in the original optimization algorithm gradient descent but with some clever adjustments.

The gradient descent method only makes use of the first order derivative of the Taylor expansion (see Equation (20)) and can, in general, be formulated according to Equation (28), where n is the *n-th* iteration, x is the optimization variable, $\nabla f(x_n)$ the gradient of the optimization function and α the learning rate (which is a tune-able design parameter for the gradient descent algorithm).

$$x_{n+1} = x_n - \alpha\nabla f(x_n) \tag{28}$$

In order to calculate the gradient, we can make use of the Jacobian transpose, J^T and the residual, $r(x)$ (can also be denoted as $f(x)$ and also referred to as the optimization function) according to Equation (29). This is achieved by realising that the Jacobian transpose results in a matrix whose columns correspond to the gradient vectors to each residual with respect to the parameters. By multiplying the Jacobian transpose with the residual we essentially end up with the gradient vector.

$$x_{n+1} = x_n - \alpha J^T r(x) \tag{29}$$

The gradient descent method solves the inverse kinematic problem by iteratively adjusting the joint angles in the direction of decreasing error between the desired and actual end-effector positions. This is illustrated in Figure 8 in a case of only one joint. To perform this, the method makes use of the transpose of the Jacobian matrix (hence the name) to map the error between actual and desired end-effector positions into the joint-angle space. For the purpose of inverse kinematics, x_n and x_{n+1} would denote joint angles, while the residual, $r(x)$ would denote an error vector for the current pose and the desired pose.

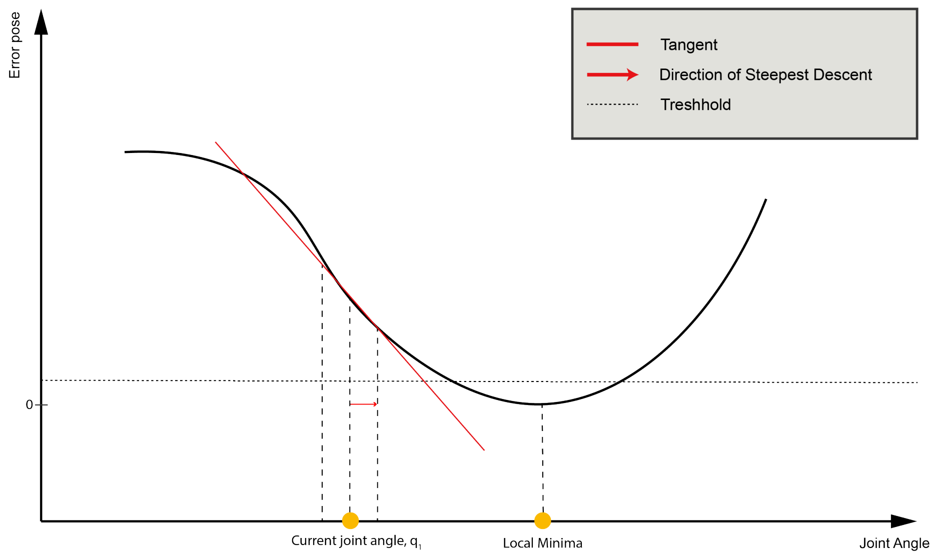


Figure 8: Illustration of gradient descent in a 2D case, i.e. with only one joint.

The pseudo-code for the Jacobian transpose method is provided in Algorithm 1.

Algorithm 1 Gradient Descent Method

```
goal_pose = y
q = current joint angles
step_size = desired step size
tolerance = set tolerance
e = goal_pose - current_pose
while norm(e) >= tolerance do
    J = Jacobian(q)                                   ▷ Compute Jacobian with method
    J_T = J.transpose()                               ▷ Compute Jacobian transpose
    gradient = alpha * J_T * e
    q += step_size * gradient
    q = check_joint_limits(q)                         ▷ Adjust to max/min value if above/under limit
    e = goal_pose - ForwardKinematics(q)              ▷ Compute pose with FK method
end while
```

6.2.1 Design Parameter Selection

In Equation (28) the learning rate (α) can be seen, The learning rate is a design parameter in the gradient descent optimization algorithm that determines the size of the step taken in the direction of the steepest descent during each iteration. If the learning rate is too high, the algorithm may overshoot the optimal joint angles and fail to converge, while if it is too low, the algorithm may take too long to converge to the optimal joint angles.

Common practice is to tune the learning rate of the algorithm through trial- and error, thus several tests were run to find out a constant learning rate that yields a well-behaved and fast solver. To find the best-suited learning rate for the algorithm tests were run for three different end-effector poses, ranging from small to large distances from the starting angle configuration in the Cartesian space (do note that the increasing Euclidean distances of the poses in Cartesian space do not imply that the magnitude of the distance will be in the same order within the joint-space). For each value of the learning rate, the test was executed five times to get the mean value of the calculation time. The results are presented in Table 2.

α	0.5	1	1.5	2	2.5	3	3.5	4	4.5
Test 1	504	291	216	169	137	108	101	50	N.C.
Test 2	568	318	244	199	160	143	103	N.C.	N.C.
Test 3	153	56	38	25	26	16	14	16	22

Table 2: Table over the computational time in milliseconds for different learning-rate values (N.C. stands for no convergence)

Of the tested learning-rate values it could be concluded that a rate of 3.5 will yield the best performance. In the executed tests it was shown that a learning rate which exceeds a value of four will lead (in some cases) to the algorithm not converging to a solution within a set amount of iterations (5000 in this case). By studying the behaviour of the algorithm further, by plotting the partial responses at each iteration, it can be seen that the non-convergence occurs due to oscillations in the algorithm when the learning rate is too high (which coincides with the learning-rate description above).

6.3 Gauss-Newton

The Gauss-Newton algorithm is often used in optimization for finding the optimal value of a function by finding the root of its derivative. As with the gradient descent method, Gauss-Newton starts off by defining an initial guess x_n. This is seen as the first guess of the optimal value, it then makes use of the formula presented in Equation (30) to find a better estimate of the optimal value, denoted x_{n+1}. Notice however that the formula presented is for the scalar case. The idea in the presented formula is to divide the gradient of x_n with the curvature at the same point. This means we are making use of the second-order derivative, which means this algorithm makes use of more information than the gradient descent.

$$x_{n+1} = x_n - \frac{f^{'}(x_n)}{f^{''}(x_n)} \tag{30}$$

This essentially means we are using a second-order Taylor expansion when linearizing the system. The Taylor expansion for this system is described here:

$$f(\theta_d) = f(\theta_i) + \left(\frac{\partial f(\theta)}{\partial \theta}\right)\bigg|_{\theta_i} (\theta_d - \theta_i) + \frac{1}{2!}\left(\frac{\partial^2 f(\theta)}{\partial \theta^2}\right)\bigg|_{\theta_i} (\theta_d - \theta_i)^2 + \mathcal{O}((\theta_d - \theta_i)^3) \tag{31}$$

The second-order derivative described in Equation (30) (or rather second-order partial derivative for this multi-dimensional case) is called the Hessian matrix. In the Gauss-Newton method, the Hessian matrix is used to approximate the curvature of the cost function at the current joint configuration. The gradient vector is used to determine the direction of the steepest descent. The Gauss-Newton method is closely related to the Newton-Raphson method, with the biggest difference in how the Hessian is computed. In the Newton-Raphson method, the Hessian is the exact second-order partial derivative, while in Gauss-Newton the Hessian is approximated using the Jacobian matrix. This approximation is done, as computing the exact Hessian matrix can be computationally expensive and often it is not even necessary to obtain the exact Hessian.

As stated, the Gauss-Newton method uses the Jacobian matrix to approximate the Hessian matrix. It does this by assuming that the function to be optimized can be expressed as a sum of squares of residuals. This assumption is considered reasonable in many optimization problems, including inverse kinematics.

Using this assumption, the Gauss-Newton approximation replaces the Hessian matrix in the Gauss-Newton method with the product of the Jacobian matrix and its transpose. The assumption that the residuals are small and that the Hessian is positive definite is made in this case. The left pseudo-inverse of the Jacobian matrix, seen in Equation (32), is then used instead of the original inverse to solve the optimization problem.

$$J^\dagger = (J^T J)^{-1} J^T \tag{32}$$

where J is the Jacobian matrix, J^T is the transpose of the Jacobian matrix, and $(J^T J)^{-1}$ is the inverse of the matrix product $J^T J$. This results in the equation below, where H is the Gauss-Newton approximation of the Hessian matrix.

$$\begin{aligned} x_{n+1} &= x_n - H^{-1} \nabla f(x_n) \\ &= x_n - J^\dagger r(x) \end{aligned} \tag{33}$$

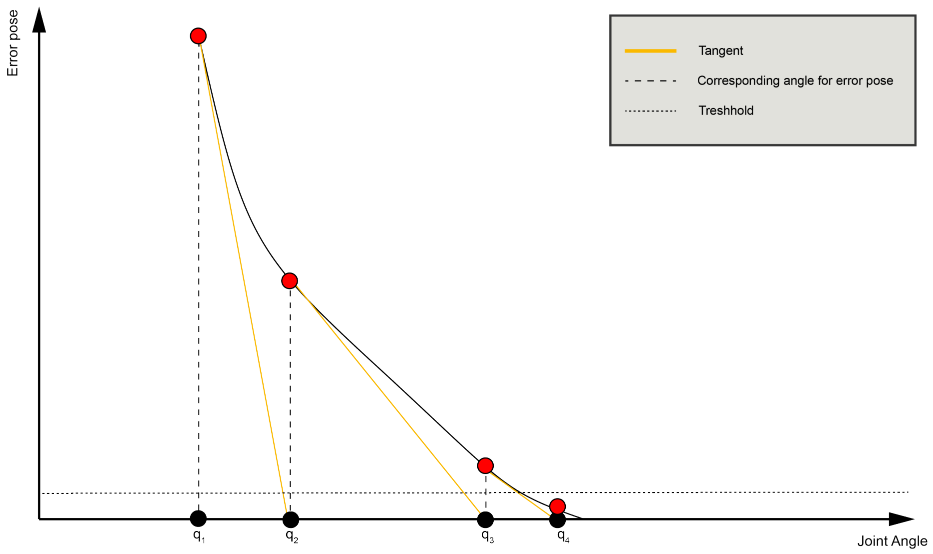


Figure 9: Illustration of how the Gauss-Newton method optimizes the problem with regard to one joint angle.

Algorithm 2 Gauss-Newton Method

```
goal_pose = y
q = current joint angles
step_size = desired step size
tolerance = set tolerance
e = goal_pose - current_pose
while norm(e) >= tolerance do
    J = Jacobian(q)                          ▷ Compute Jacobian with method
    J_T = J.transpose()                      ▷ Compute Jacobian transpose
    J_pinv = (J_T * J).inv() * J_T
    delta_q = J_pinv * e
    q += step_size * delta_q
    q = check_joint_limits(q)                ▷ Adjust to max/min value if above/under limit
    e = goal_pose - ForwardKinematics(q)     ▷ Compute pose with FK method
end while
```

6.4 Levenberg-Marquardt

The Levenberg-Marquardt method (also known as the damped least squared method or DLS). It has its origin in the least square curve fitting and has been used with great success amongst many manipulator configurations. It has been used in the field of robotic manipulators at least since 1986 by both Wampler [15] and Nakamura [16].

Like both the gradient descent and Gauss-Newton algorithms, Levenberg-Marquardt starts with an initial guess x_n and then iterates to the next guess x_{n+1}. The process of which the algorithm takes to update the estimate of the optimal value can be seen as a combination of Gauss-Newton and gradient descent using the following equation:

$$x_{n+1} = x_n + (J^T J + \lambda I)^{-1} J^T r(x_n) \tag{34}$$

where x_n is the estimate of x at the n-th iteration, J is the Jacobian matrix of the function $f(x)$ with respect to x, $r(x_n)$ is the residual vector at x_n, λ is a damping parameter and I is the identity matrix.

As stated, the Levenberg-Marquardt algorithm updates x using a combination of the Gauss-Newton algorithm and the gradient descent algorithm. For smaller λ, the algorithm behaves like the Gauss-Newton algorithm. For a well-behaved function, it means it will converge rather quickly. When λ is large, the algorithm will behave like the gradient descent algorithm, which generally is a bit slower but more robust.

The Levenberg-Marquardt algorithm uses the damping parameter λ to control the step size and prevent the algorithm from diverging in case of ill-conditioned problems. If the Jacobian matrix is ill-conditioned or singular, the Levenberg-Marquardt algorithm adds a damping term to the diagonal of the matrix, which reduces the step size and makes the algorithm more robust.

6.4.1 Design Parameter Selection

In Equation (34) the damping factor (λ) can be seen, as mentioned above the damping factor determines the behaviour of the algorithm. The damping factor is decided by means of trial- and error with the same test cases as the ones used for deciding the learning rate in Section 6.2.1.

λ	0.05	0.1	0.15	0.2	0.25
Test 1	63	68	116	141	179
Test 2	78	93	110	181	196
Test 3	88	133	157	225	N.C.

Table 3: Table over the computational time in milliseconds for different learning-rate values (N.C. stands for no convergence)

Algorithm 3 Pseudo Code for Levenberg-Marquardt Algorithm

```
goal_pose = y
q = current joint angles
step_size = desired step size
tolerance = set tolerance
e = goal_pose - current_pose
lambda = damping factor
while norm(e) >= tolerance do
    J = Jacobian(q)                                  ▷ Compute Jacobian with method
    J_T = Jacobian.transpose()                       ▷ Compute Jacobian transpose
    J_inv = (J_T * J + lambda * I).inv() * J_T
    delta_q = J_inv * e
    q += step_size * delta_q
    q = check_joint_limits(q)                        ▷ Adjust to max/min value if above/under limit
    e = goal_pose - ForwardKinematics(q)             ▷ Compute pose with FK method
end while
```

6.5 Other solutions

Although there has not been a lot of progress in the field of robotics in regard to inverse kinematics, there has been more research put into computer animation. This stems from the increased development of more and more complex computer games and movies which require real-life-like movements of (animated) humanoid robots, robotic manipulators and characters with various joints and limbs. The main difference between robotics and computer animation is that there often exist joint limitations and a need for collision avoidance as well as taking the end effector's pose into consideration instead of just the position for real-life manipulators. These requirements are of course relevant in some different areas within computer animation, but often not present all at once. Additionally, it is often more important in robotics to achieve higher accuracy, while computer animations put more emphasis on computational power in order to simulate multiple characters/manipulators simultaneously, making heuristic methods a viable option.

As for robotics, analytical and numerical solutions are viable options, but additional solutions have been developed for computer animations. Among those are *CCD* (Cyclic Coordinate Descent) and *FABRIK* (Forward and Backward Reaching Inverse Kinematics) which are heuristic methods [17].

These methods will however not be implemented and tested, but serves only the purpose of highlighting different techniques and their respective advantages and disadvantages.

FABRIK

FABRIK was introduced in the 21st century by Aristidou and Lasenby [18] and is a heuristic-based algorithm which, in theory, bypasses the use of rotational matrices, Jacobian, and frame transformations. The method has proven to be exceptionally fast in computing all the joint angles, and it has been proven to work well in computer animation. The idea is to iterate through each joint, starting with the last joint (*joint n*) and connecting it to the desired end position. The following joint is then connected to the previous (in a straight line) until the last joint (in this case *joint 1*) has been connected. The algorithm then reverses, connecting *joint 1* to its fixed starting position and iterates through to *joint n.* This iteration is continued until the last positional error is within a specified tolerance. The process is illustrated in Figure 10

In its novel form, the FABRIK method does not take either joint limits or the orientation of the end effector into consideration. Joint limits can however easily be implemented by always checking if the issued joint angle is within the limits and setting to adjust it to be within the limits if needed as explained by Santos et al [19]. There exist a few sources which also take the orientation of the end effector into consideration, which results in more computation. In addition to this, the method can be stuck in a local optimum, thereby not finding a global optimum for the problem

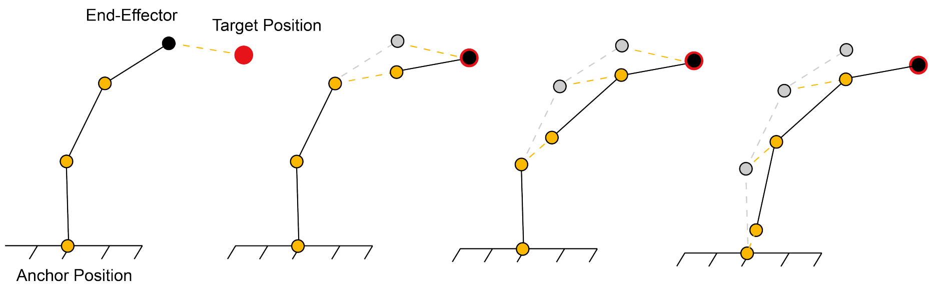


Figure 10: FABRIK illustration

CCD

The Cyclic Coordinate Descent (CCD) method, introduced in by Wang and Cheng [20] 1991, is another popular iterative heuristic algorithm for solving inverse kinematics problems in computer animation. It is a local method that operates by iteratively adjusting the joint angles of a robotic manipulator to minimize the difference between the end effector's current position and the desired target position.

The basic idea of CCD is to start with an initial guess of the joint angles and to iteratively adjust them, one joint at a time, to move the end effector closer to the target, illustrated in Figure 11 At each iteration, the method starts with the last joint and computes the vector from this joint to the end effector. It then rotates the joint by an amount proportional to the angle between this vector and the vector from the joint to the target.

This process is repeated for all the joints in a cyclic fashion, hence the name Cyclic Coordinate Descent. After one cycle of updating all the joints, the algorithm checks if the end effector is close enough to the target. If it is not, the process is repeated until the error is below a certain threshold.

One of the advantages of CCD is that it is a relatively fast method and is able to converge to a solution in many cases. However, it is important to note that the method can get stuck in local minima, especially in complex multi-joint manipulators. Additionally, CCD does not guarantee a globally optimal solution, and it can be sensitive to the order in which the joints are updated. CCD can also be slower than other IK methods, such as the gradient descent or Gauss-Newton, especially when dealing with complex robotic systems with multiple degrees of freedom.

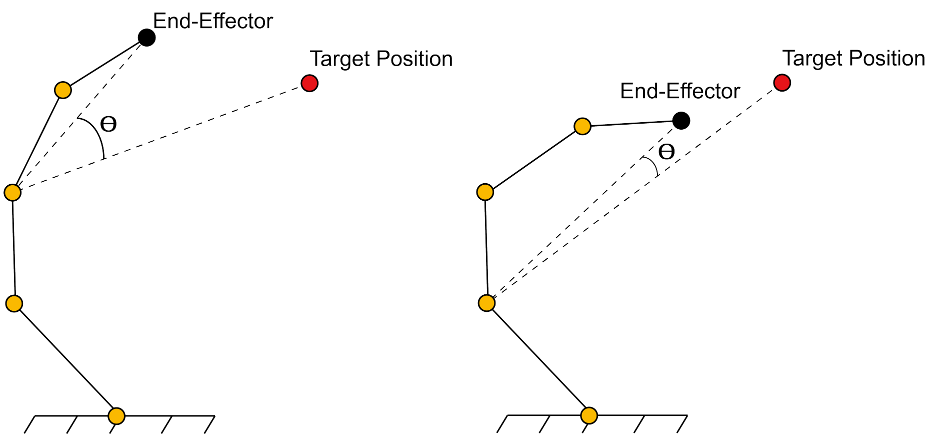


Figure 11: CCD illustration

7 Analytical Concept

The analytical solution to the inverse kinematic problem (or the closed-form expression as it's also commonly referred to) presents a way to find the joint-angle configuration which represents a certain pose in the Cartesian space in a direct fashion.

If an analytical solution can be derived there is no need for an iterative process that oftentimes affects the performance of the control system negatively (in the sense of draining the computation resources, resulting in a slow process which may impact the ability to run in real-time), hence why it is the most popular approach to the problem. However, for an arbitrary manipulator configuration, there is no guarantee that an analytical solution exists, and even so, if the solution exists it may be severely difficult to derive. Therefore, the first step to finding the analytical solution to this particular inverse kinematic problem is to explore the existence of the solution.

7.1 Existence of Analytical Solution & Piepers Theorem

As mentioned previously, there is no guarantee that the analytical solution exists for an arbitrary manipulator configuration. Because of this, it has been suggested in plenty of influential works (for example in [6] and [21]) within the field of robotic manipulators to have the design of a manipulator follow specific standards (such as the spherical wrist for 6-DOF manipulators) in order to ensure the existence of an analytical solution to the inverse kinematic problem.

The most common design standard when it comes to 6-DOF manipulators is that of the "spherical wrist"-design, which implies that the last three axes of the manipulator shared a common intersection of their individual rotation axes. The paper [22] by Alexander J. Elias and John T. Wen, states that with this manipulator design, the analytical solution to the inverse kinematic problem consists of dividing the problem into two sub-problems, where the first three joints control the position, whereas the last three joints control the manipulator pose.

This idea of the spherical wrist design was not the result of "trial-and-error" but rather grew forth from a research paper published by Donald Lee Pieper [1] during the late 1960s. This paper states that an analytical solution for the inverse kinematic problem exists if any three consecutive joint axes intersect at a common point (note that three parallel axes are also assumed to intersect in infinity) and to exemplify this the analytical solution to the "spherical-wrist" manipulator was derived.

The eM1-7's design does not comply with the "spherical wrist" design, hence the standard fashion of deriving the analytical solution is not applicable. The reason for this particular arm not sharing the standard of the "spherical-wrist" design is that it was originally based on other manipulators within the field of underwater surveillance, which historically has been predominantly driven by operators, thus making the need for inverse kinematics non-existent.

By studying the manipulator one can realize that there are in fact three concurrent axes which share an intersection, more precisely the parallel joints 2, 3 and 4, thus the manipulator satisfies the conditions suggested by Pieper [1].

7.2 Finding the Analytical Solution

Since the manipulator design is compliant with Pieper's condition it is presumed that an analytical solution exists. An important realization that was made was that the three parallel joints within the eM1-7's configuration closely resemble that of the "Three Revolute Parallel"-Manipulator for which the analytical solution is previously derived (for example in the Advanced Robotic course given at UCLA [23]). By dividing the kinematics problems into sub-tasks where one of the tasks is that of the "Three Revolute Parallel"-Manipulator, the overall problem becomes much more manageable and can be solved by studying each sub-task by itself and then compiling the final result. Due note that there is still a coupling between the different subtasks, therefore it is required that the tasks are executed in the order presented below.

The methodology of dividing a kinematic chain into sub-tasks when deriving a closed-form inverse kinematics solution has been used in a multitude of previous works. Some examples can be found in the papers written by Li Jiang [24] and Mathias Brandstötter [25], where different methodologies are used to solve the inverse kinematics for widely different manipulator configurations.

7.3 Sub-Problem Solutions

To properly describe the solution to each of the different sub-problems, the first part of the derivation is to denote the homogeneous transformation matrix of the wished end effector pose:

$$T_{end} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{35}$$

Each of the variable's contributions to the overall homogeneous transformation matrix is explained in Section 4.1.3.

Furthermore, (as covered in Section 5.2) the inverse kinematics solution does not actually find the centre point of the end-effector, or as it is more commonly known the tool-frame origin, but rather it finds the position and orientation of the (virtual) joint six in relation to the base coordinate system. To increase the readability of the following parts within this section this point will be referred to as the end-point.

7.3.1 SP1: Global *z*-Axis Rotation

The first sub-problem within the analytical inverse kinematics solution consists of finding the x- and y-position of the endpoint within the global coordinate frame. Because of how the coordinate frames are set up and how the end-point is defined it can be determined that only the first joint will have an effect on these positions. The desired positions are denoted as p_x and p_y respectively in the end-effector matrix (see Equation (35)). A visual representation of the first sub-problem can be seen in Figure 12.

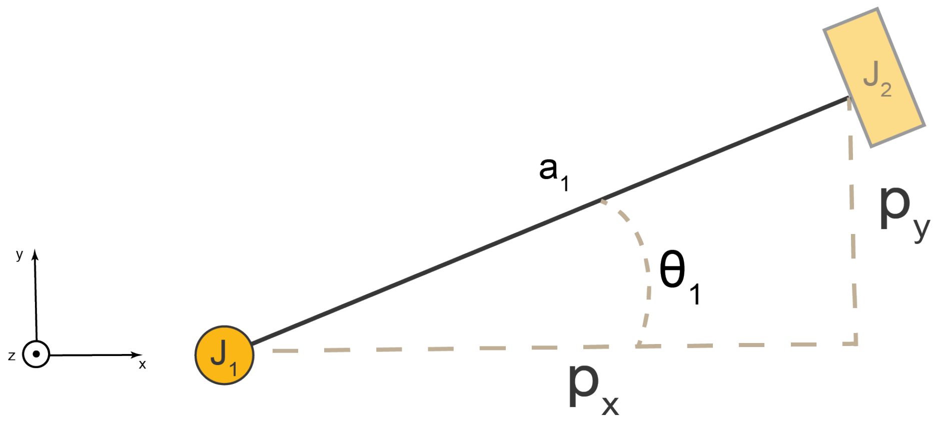


Figure 12: A top-side view of the first sub-problem.

Based on the structure of the first sub-problem, finding the solution is only a matter of solving a simple trigonometrical problem. One key factor to note is that to ensure that all possible solutions to the inverse kinematic problem are found we need to take into account that arctangent is only defined for angles between [-90, 90] whilst the joint, in this case, spans over a larger interval.

To counteract this issue, two different equations are presented for joint-angle one which differs by one period (180 deg).

$$\begin{aligned} \theta_1 &= \arctan(p_y, p_x) \\ &\vee \\ \theta_1 &= \arctan(-p_y, -p_x) \end{aligned} \tag{36}$$

7.3.2 SP2: 3R-Parallel Manipulator

To find the solution to the second sub-problem one firstly needs to recognize the structure of the "Three Revolute Planar" manipulator problem (which now will be referred to as 3RP), and why it is that the joints 2, 3 and 4 can be assumed to follow these principles.

The 3RP problem states that to find the inverse kinematic solution, the following criteria need to be full-filled:

- The orientation of the end-effector is known (within the corresponding 2D plane)
- The position of the end-effector is known (within the corresponding 2D plane)
- The first joint angle lies in the origin of the global coordinate system

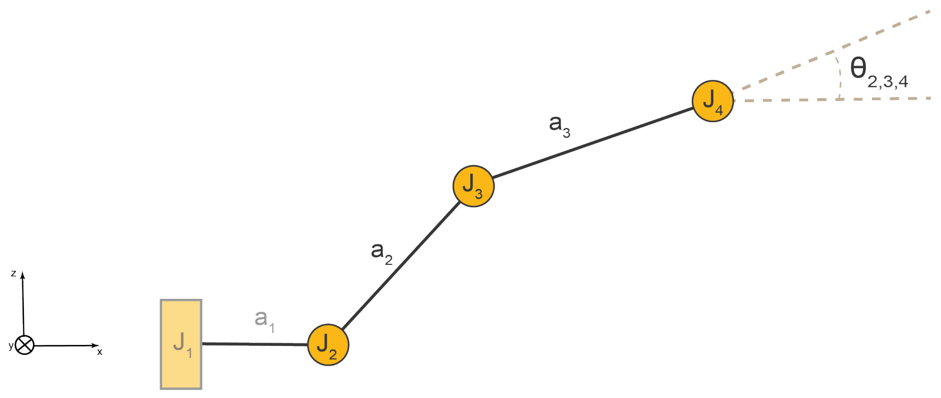


Figure 13: A side view of the second sub-problem.

To satisfy these conditions and in turn, be able to solve the 3RP IK problem three major steps needs to be taken, firstly the 2D plane where the planar manipulator acts need to be defined (moving forward this plane will be referred to as the 3RP-Plane), secondly the end-effector pose within the 3RP plane needs to be derived since it is by definition defined in the three-dimensional space (see Equation (35)) hence it needs to be projected onto the two-dimensional 3RP-plane and lastly, the 3RP problem should be solved to find joint angles 2,3 and 4.

The 3RP plane will be oriented according to the first joint angle, and this means that when calculating distances etc. for the 3RP-manipulator problem the first joint angle needs to be taken into consideration.

The first action of finding the endpoint's pose within the 3RP plane is to determine the angle of attack. To find this angle the following equation is used, and do note that in the same manner as for the first angle, two different equations for the joint angle are defined which differ by one period (180 deg).

$$\begin{gathered}\theta_{234} = \arctan(-a_z, a_x \cos(\theta_1) + a_y \sin(\theta_1)) \\ \vee \\ \theta_{234} = \arctan(a_z, -a_x \cos(\theta_1) - a_y \sin(\theta_1))\end{gathered} \tag{37}$$

Equation (37) consists of two parts which can be interpreted geometrically, firstly $a_x \cos(\theta_1) + a_y \sin(\theta_1)$ represents the length of the resulting vector from projecting the z-axis of the end-effector frame onto the xy-plane. Secondly, by first deriving the projection of the z-axis onto the xy-plane both the opposite and adjacent sides of the triangle which relate to the desired angle are known. Hence why the angle then is calculated via the use of arctangent.

To find the position of the "end-effector" in the 3RP problem one needs to calculate the x- and y-coordinates in the 3RP-manipulator plane. This can be accomplished with the following equations:

$$x = p_x \cos(\theta_1) + p_y \sin(\theta_1) - a_1 - a_4 \cos(\theta_{234}) \tag{38}$$

$$y = -p_z - a_4 \sin(\theta_{234}) \tag{39}$$

Equation (38) and Equation (39) consist of projecting the "true" end-effectors x- and y-coordinates onto the 3RP plane, and then removing the link lengths which are not present within the 3RP-problem (which are link 1 between the first and second joint as well as the projection of link a_4 between joint 3 and 4).

Now all the variables required to solve the 3RP Inverse kinematic problem are known, a visualization of the problem can be seen in Figure 14.

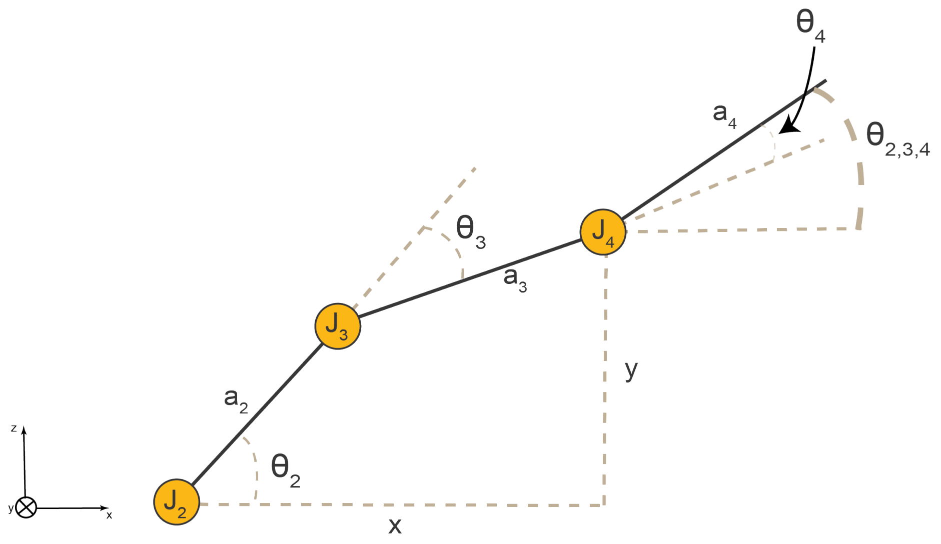


Figure 14: The 3RP planar manipulator problem

The procedure of solving the 3RP-manipulator problem uses the fact that the previously calculated orientation of the end-effector in the 3RP plane gives the following mathematical conjunction:

$$\theta_{234} = \theta_2 + \theta_3 + \theta_4 \tag{40}$$

Therefore, it is only needed to determine joint angles 2,3 to solve the problem. The first angle to find is θ_3 and this can be easily derived using the law of cosines. The law of cosines states that if all sides of a triangle are known (which with the help of regular rules of trigonometry reins true in this case) one can calculate the angle of a triangle. This in turn gives the following expression for the third joint angle (which of course returns two values for the angle):

$$\theta_3 = \arccos\left(\frac{x^2 + y^2 - (a_2^2 + a_3^2)}{2\, a_2\, a_3}\right) \tag{41}$$

The solution for the second joint angle is not as straightforward, to find a geometrical relation which solves for θ_2 one needs to study the geometries of the first two links in further detail.

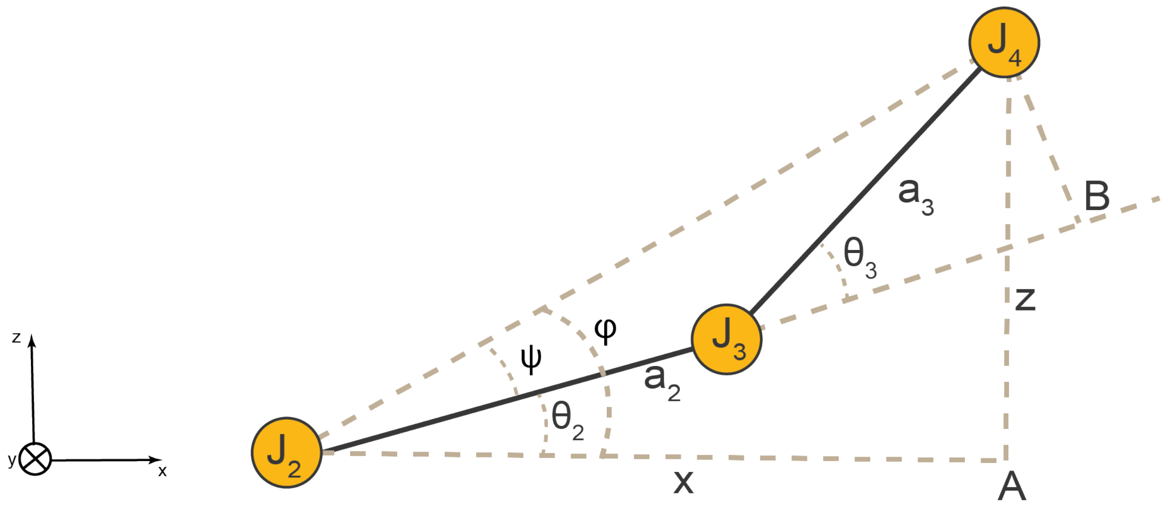


Figure 15: Geometrical overview of the first two links within the 3RP problem.

Two geometrical conjunctions can be found within Figure 15 and these are from triangle J_2J_3B:

$$\tan(\Psi) = \frac{a_3\ \sin(\theta_3)}{a_2 + a_3\ \cos(\theta_3)} \tag{42}$$

From triangle J_2J_3A:

$$\tan(\varphi) = \frac{z}{x} \tag{43}$$

Furthermore, in Figure 15 a relation between angles θ_2, β and γ is found, hence by using Equation (42) and Equation (43) the second joint angle can be derived according to:

$$\theta_2 = \arctan\left(\frac{y}{x}\right) - \arctan\left(\frac{a_3\ \sin(\theta_3)}{a_2 + a_3\ \cos(\theta_3)}\right) \tag{44}$$

All that remains of the 3RP-problem is now to use the known relation in Equation (40) to find the fourth joint angle:

$$\theta_4 = \theta_{234} - \theta_2 - \theta_3 \tag{45}$$

Thus concluding the solution to the second sub-problem which has achieved the second, third and fourth joint angles.

7.3.3 SP3: Tool-frame orientation

The last sub-problem purely affects the orientation of the end-effector, since both origins of the coordinate frames for joints five and six interlay with the end-point. A visual representation of the sub-problem can be seen in Figure 16.

The fifth joint angle can be determined by the following expression:

$$\theta_5 = -\arccos(a_y\ \cos(\theta_1) - a_x\ \sin(\theta_1)) \tag{46}$$

Equation (46) represents projecting the orientation of the end effector's z-component into the *xy*-plane, thereafter the x- and y-components are scaled in regard to the first joint angle. Then by taking the cosine inverse on the resulting projection, the fifth joint angle is found.

The last remaining joint angle is found via the expression:

$$\theta_6 = -\arccos\left(-(o_x \sin(\theta_{234}) \cos(\theta_1) + o_y \sin(\theta_{234}) \sin(\theta_1) + o_z \cos(\theta_{234}))\right) \tag{47}$$

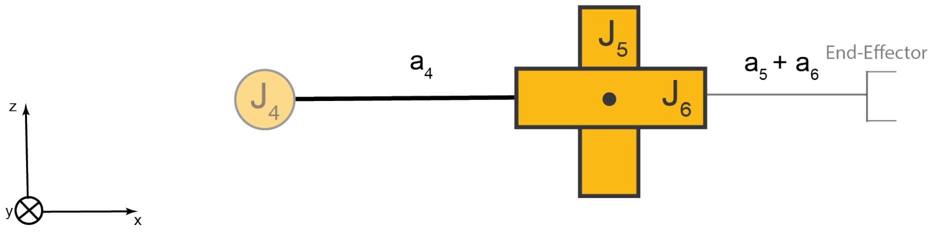


Figure 16: A side view of the third sub-problem.

7.4 Choosing Solution

When deriving the analytical solution for inverse kinematics, trigonometrical equations are frequently used. Due to the nature of these functions, most of the times there are often multiple solutions, for instance, $\arccos(\theta) = \arccos(-\theta)$. Hence why there can exist multiple solutions (32 solutions) to choose from, this is an issue since the algorithm should return one angle configuration, and it should do so without any assistance from an operator. Thus, the algorithm needs a way to determine the most suitable candidate for the available solutions.

Firstly, all the solutions need to be validated, since some "solutions" will not be compliant with the final end-effector pose for which the algorithm was searching. The algorithm can validate this by using forward kinematics on the found angle configuration and comparing the pose with the desired end-effector pose. Another constraint is to check that the given angle configuration is within the boundaries of the angle constraints. If no angle configuration satisfies the constraints, angle clipping can be used to return an estimate of how close the manipulator can go within its limitations. Clipping is a method where one sets an angle which exceeds its boundary conditions to either the maximum or minimum value, depending on which boundary was exceeded.

The remaining angle configurations, after confirming the validity of the suggested solutions, can all be deemed appropriate candidates for the resulting angle configuration. Depending on what the aim of the manipulator control is, the method of choosing an angle configuration can differ. The most direct and easy approach is to choose the angle which results in the least amount of movement in joint space, and this should in practice yield the least amount of energy consumption whilst also resulting in the shortest time duration for the manoeuvre. Of course, in some specific applications it could be crucial that the manipulator is following a trajectory with an *elbow-up* configuration, in cases like these the optimal way to choose the solution is subject to change.

8 Singularity Free Path Generation

The inverse kinematics of a manipulator realizes the possibility to describe a certain position of the end-effector in Cartesian space with a set of angles in the joint space. However, to ensure the feasible motion of the manipulator it is often not enough to simply state the final joint configuration, and instead a more detailed description of the path between the initial- and final state is needed.

Recalling the problem description in Section 1.2 one goal of the master thesis is the implementation of the said path with respect to singular configurations. Essentially the problem boils down to creating a path which can, in an appropriate amount of time, find its way from an initial- to a final state whilst avoiding singular regions which could yield unwanted behaviour from the manipulator. In the case of this master thesis trajectory generation is not covered, and this would be a further development of the path-planning where one would introduce velocity and time constraints.

One of the most common practices when working with path-creation for manipulators, which is brought up in John J. Craig's textbook [6] for example, is that of via-point creation. By having intermediate via-points between the initial- and goal state one can, by requiring that the manipulator traverses through these via-points, control the path taken by the manipulator with a discrete approach.

Furthermore, path-planning for the manipulator can be separated into two different types depending on in which space the path will be described, joint- or cartesian-space paths. Join-space paths are described as intermediate angle configurations from an initial state to a goal state while cartesian-space paths are instead described as intermediate poses (SE3).

8.1 Joint-Space

When planning a path within the joint space the inverse kinematics only needs to be derived for our end- and starting states, i.e. it is required to know the joint angle configurations at the beginning of the path and in the end (can also be extended to intermediate goals which are required to be visited). This in turn makes the joint-angle path planning much less computationally heavy since inverse kinematics is rarely needed.

To find the intermediate points on the path when working in joint-space it is beneficial to use joint-angle interpolation between the end- and starting state. This way a smooth path can be created where each joint makes appropriately large steps to end up in the goal state thus completing the path.

8.2 Cartesian-Space

In contrast to joint-space path planning cartesian-space planning opens up a lot more design choices, for instance, the path can take numerous amount of shapes and the most common is a straight line motion of the end-effector which is what has been implemented in this master thesis.

Straight-line motion can be created in different ways where the truest form would include parameterizing the motion of the end effector to ensure that every time step would apply a straight motion (this is effectively a trajectory implementation). A more simple approach is to have an approximately straight cartesian path by ensuring that each via-point is created in proximity to one another which makes it possible to interpolate between the angle configuration of each via-point whilst having the end result still resembling a straight cartesian path.

8.3 Implementation

In this master thesis, the goal with the path planner was to have a simple implementation that would realize smooth movement in the Cartesian space, which essentially is finding a path between an end- and staring-state which successfully can avoid singularity regions.

The first part of creating the path is finding the intermediate point via the means of interpolation. It is ill-advised to directly interpolate between different SE3-poses hence why the interpolation contains separately interpolating the end-position and the orientation. The position is linearly interpolated between the start and end position whilst the orientation is converted to quaternions and thereafter interpolated in the same manner.

To ensure that the motion of the end-effector mimics that of a straight line in Cartesian space the intermediate poses need to be in close proximity to one another, as mentioned above. Thus, the number of interpolated points is decided by looking at the Euclidean distance between the end and starting state.

The next step is adapting the cartesian path to reference values which the actuators (joints) can follow. To realize this any of the inverse kinematics solutions that have been developed can be used. Applying the inverse kinematics will yield a number of joint-angle configurations equal to the number of points on the path. The following step is to ensure that no singular configurations are in play when following the path. To realize this the Jacobian (which describes the relations between cartesian and joint velocities in the system) is used.

In Section 5.3 it is stated that when the manipulator is in a singular configuration the determinant of the Jacobian will yield zero as a result. In addition to being able to identify singular configurations, the determinant can also be studied to find out whether the manipulator is approaching a singular configuration, i.e. when the manipulator is entering a singular region. This manner of searching the path for singularities introduces a design parameter which is the threshold for which the determinant should not cross for a configuration to be deemed "valid" (if the goal is to have a well-behaved smooth motion). This design parameter is a trade-off between how well the manipulator follows the specified Cartesian path and how smooth the joint velocities will be.

To overcome the unwanted behaviour of the manipulator when trying to traverse the singular regions one makes use of the fact that the path can be planned in both joint- and cartesian-space. Since singularities are a problem that occurs when trying to define joint angle configurations from a cartesian position the path planner switches over to joint-angle interpolation within the singular regions. This results in a path which the manipulator can follow which closely resembles straight-line motion but ensures that no singularities of the manipulators occur (where the quality of the result depends on the chosen threshold which was defined earlier).

A pseudocode over the path planning can be seen in Section 8.3, furthermore a visual representation of the resulting path is included in Figure 17 where the threshold has been exaggerated to more clearly show the inner workings of the path-planner.

Algorithm 4 Pseudo Code for Singularity Free Path Planning

```
SE3Poses = linspace(Tstart, Tend, n)
angles = InverseKinematics(SE3Poses)
for idx in angles do
    if idx in SingularConfigs then
        skip
    end if
    detJ = Determinant(Jacobian)
    if detJ < threshold then
        SingularityStart = idx
        j = idx
        while Determinant(Jacobian[j]) < threshold do
            SingularConfigs.append(j)
            j += 1
        end while
        SingularityEnd = j
        InterpolateAngles = linspace(angles[SinuglarityStart:SingularityEnd], [idx:j])
        angles[SinuglarityStart:SingularityEnd] = InterpolateAngles
    end if
end for
```

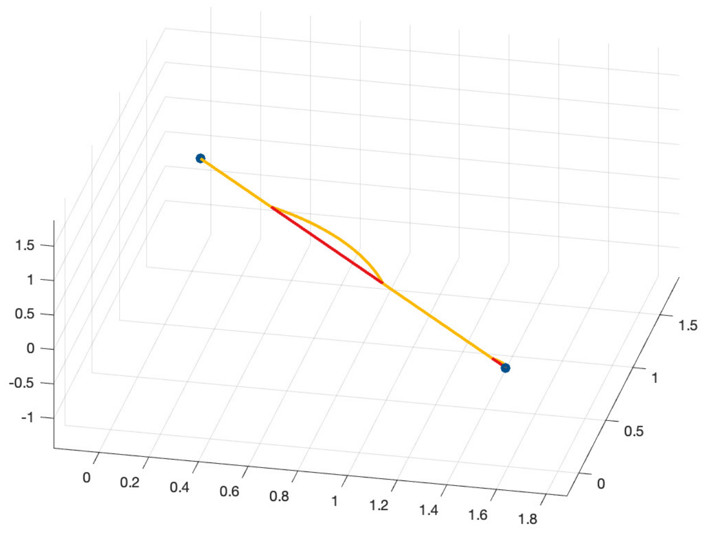


Figure 17: Visual representation of the generated path. The red line represents the straight cartesian path whilst the yellow line represents the singularity-free path.

9 Results

This chapter aims to present the results of the work conducted. It has been separated into two different parts, one for the three numerical inverse kinematic solutions and a separate section for the analytical solution. The validation is separated since the different approaches to the problem are fundamentally different, hence why it would not make sense to compare them in detail to each other. In addition, when looking to validate the different approaches to the inverse kinematic solution different tests are of importance whether the solution is numerical or analytical. However, the average computational times will be compared together since it clearly shows why the analytical solution to the inverse kinematic is superior.

9.1 Numerical Inverse Kinematics

For the validation of numerical inverse kinematics, the interesting factors are essentially how well it can converge to a solution and if convergence happens within a reasonable time frame.

So to test the solutions it is relevant to have different test cases where the desired end pose differs since the initial guess within the optimization algorithms can have a large impact on the end result. Therefore, multiple test cases are created where each represents different distances between the end- and start-state, where the start-state remains constant in the zero-configuration (the starting point of the manipulator) and the end-states chosen at different distances away from the start in cartesian space. For the first test that was conducted, the aim is to understand the behaviour of each algorithm, hence there is no time aspect included, to begin with. Later on in the results chapter, each algorithm will be tested through a multitude of different test-case to properly time the algorithms.

In the results tested below it should be noted that the iteration limit was set at 5000 iterations, although for the purpose of more clearly showing the behaviour of each algorithm the diagrams will only show the first 650 iterations of each test.

9.1.1 Tests on different end-effector poses

The first test case represents an end-effector position which is located in a non-singular region as well as having a joint-angle solution close to the initial guess (zero-config).

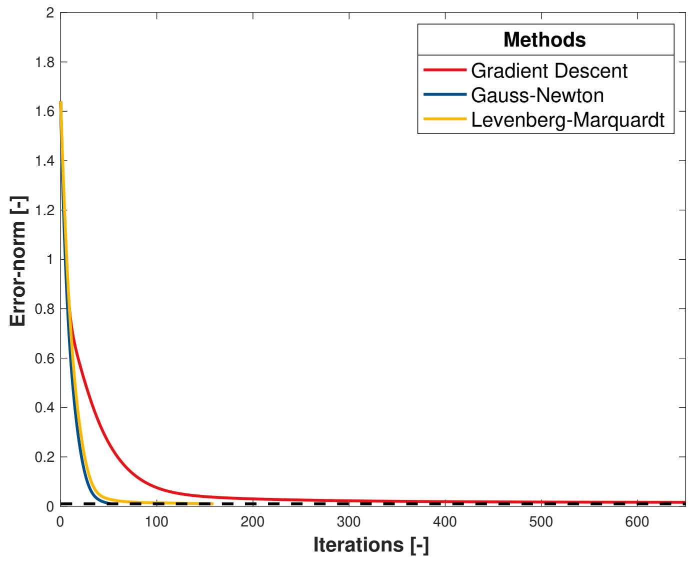


Figure 18: In numerical solutions to the IK-problem for the first test case, all methods converge within a set interval of 5000 iterations.

The second test case represents an end-effector position which is located within a singular region (i.e. in close proximity to a singular configuration), more precisely the end-effector position is located close to the zero-configuration in Cartesian space.

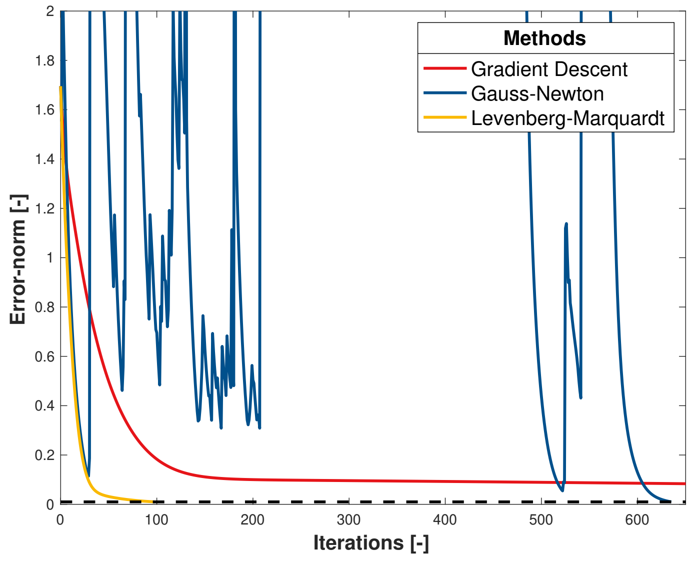


Figure 19: Numerical solutions to the IK-problem for the second test case, all methods converge within a set interval of 5000 iterations. The Gauss-Newton algorithm appear to diverge initially, but eventually converge. It should be noted that the figure is zoomed in in order to see the initial behaviour, thus not showing when the gradient descent method converges.

The third test case represents an end-effector position which is located in a non-singular region as well as having a joint-angle solution which is located far away from the initial guess.

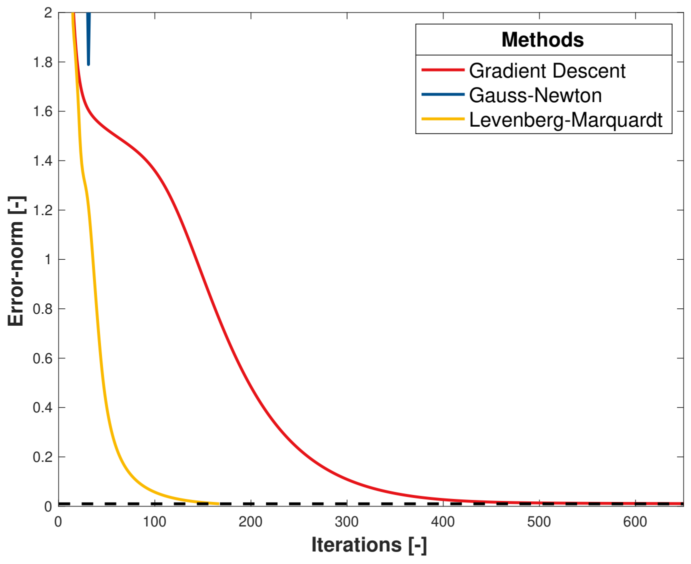


Figure 20: Numerical solutions to the IK-problem for the third test case, all methods except the Gauss-Newton method converge within a set interval of 5000 iterations. One can see that Gauss-Newton diverges quickly and does not converge any closer.

9.2 Analytical Inverse Kinematics

Due to the nature of the analytical solution, there is not any computational speed or accuracy to worry about. The analytical solution is created in a way that gives a direct response to a position request, in contrast to the numerical solutions where the error will decrease with each iteration, hence the major part of the results in this section will be validating that the algorithm manages to follow trajectories appropriately, i.e. the right solution is chosen from a set that is non-zero (choosing solutions is covered in Section 7.4).

By once more using the singularity-free path generator together with the inverse kinematic solution the result will present a way to validate whether the analytical inverse kinematics solution yields appropriate results when running on a Cartesian trajectory. A satisfactory result would be that the trajectory is followed whilst no drastic changes are made in the joint space (this is what is deemed as a smooth movement, for example, the algorithm should not change midway during a trajectory execution from elbow-up to elbow-down configuration).

Hence the presented results will consist of a MATLAB visualisation of the manipulator trail after execution of the singularity-free path (same path as visualized in Figure 17) as well as each joint-angle during the execution of the trajectory.

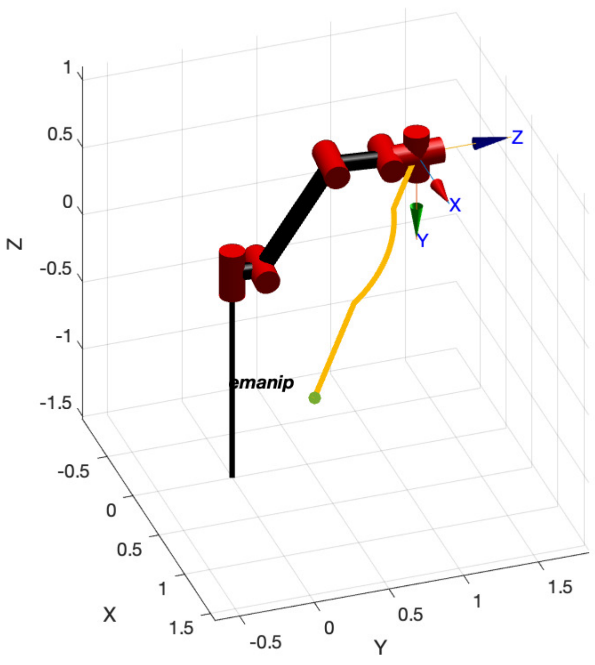


Figure 21: Manipulator trail after executed trajectory

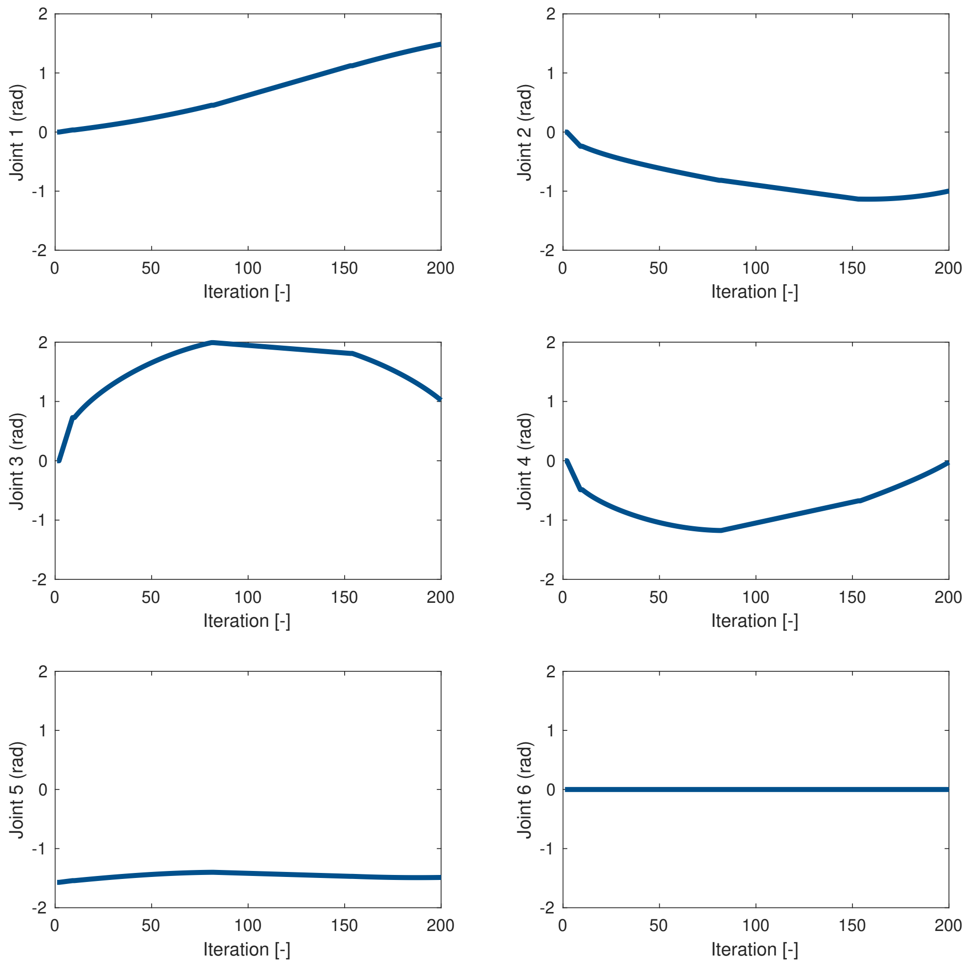


Figure 22: Joint-angles during the path execution with the analytical inverse kinematics

9.3 Further Validations

This section will present some tests executed for numerical and analytical solutions. These tests aim to further validate the behaviour of the solutions in regard to the speed at which a solution can be calculated. This is of the highest importance in real-life applications since the manipulator should essentially be able to decide the joint angles from the inverse kinematics solution in real-time (real-time can be assumed a response frequency of above approximately 20Hz).

9.3.1 Calculation Times

As previously mentioned calculation times are compared between each of the numerical algorithms as well as the analytical. To ensure that the results of these tests are reliable every test case was run a set number of times and the mean computational times were derived, this is to take into consideration that the optimization algorithms have a tendency to differ slightly in time to convergence. The specifications of the test benchmark environment are presented in Table 4.

Computer	MacBook Pro 2017
Processor	2.3 GHz Dual-Core Intel Core i5
Memory	8GB
Compiler	Apple Clang 14.0.0
Version	C++11
Compiler option	O2
External libraries	Eigen3

Table 4: Presentation of the computer and compiler used for compiling and executing the tests

A maximum number of iterations were set to 10000 with a threshold of 0.01 (meaning that the norm of the error pose should be lower than 0.01 for convergence.). The step size was set to 0.1 for all algorithms.

Test #	Gradient Descent [μs]	Levenberg-Marquardt [μs]	Gauss-Newton [μs]	Analytical Solution [μs]
1	84	7757	304	45
2	243	7186	272	35
3	450	24300	229	32
4	278	7818	206	42
5	352	4778	218	33
6	468	5131	856	32
7	499	4602	**F**	31
8	383	2577	207	33
9	122	1188	240	33
10	503	4863	983	33
11	662	9553	268	32
12	342	3242	5042	32
13	1569	9781	242	38
14	657	6237	**F**	32
15	3051	26372	364	31
16	289	2399	**F**	32
17	287	3855	**F**	31
18	559	5899	**F**	32
19	809	9749	130	34
20	1083	8296	391	34
21	1412	**F**	378	31
22	298	13130	**F**	32
23	1950	6771	405	35
24	577	1285	416	33

Table 5: Time for solving the inverse kinematics for multiple different poses, in case of no convergence, it is denoted as **F**.

All test poses can be seen in Section A.

9.3.2 MoveIT

By implementing the inverse kinematics solutions into the ROS package MoveIt one can use visualization tools such as RViz to validate the behaviour of the inverse kinematics. RViz gives the user the ability to grab onto the end-effector of the manipulator and move it around within the workspace, at each iteration the underlying inverse kinematics needs to provide a solution to accurately represent the visualization of the manipulator in the RViZ GUI. This lets the user tests a lot of different positions in the workspace efficiently, and this is what has been done for the solutions presented in this report.

To try to visualize the results Figure 23 and Figure 24 were captured when working with the GUI to show off how this type of testing was executed. It could be concluded that for all inverse kinematic solutions, the inverse kinematics could be calculated accurately and fast enough to enable the manipulator to be dragged around in the workspace in "real-time", for well-behaved poses (i.e. in singular configurations or heavily singular regions the quality behaviour dropped).

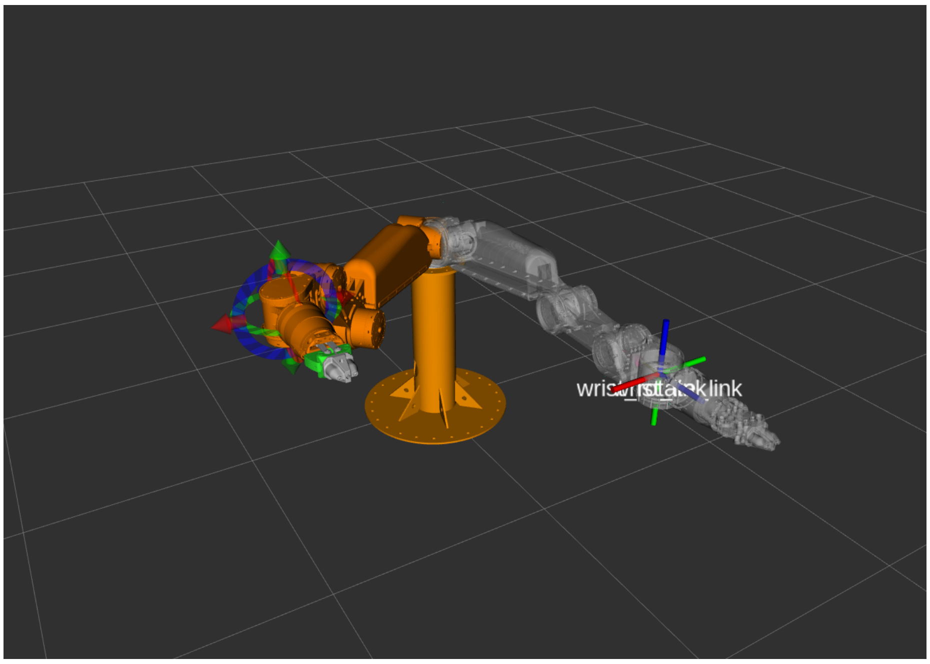
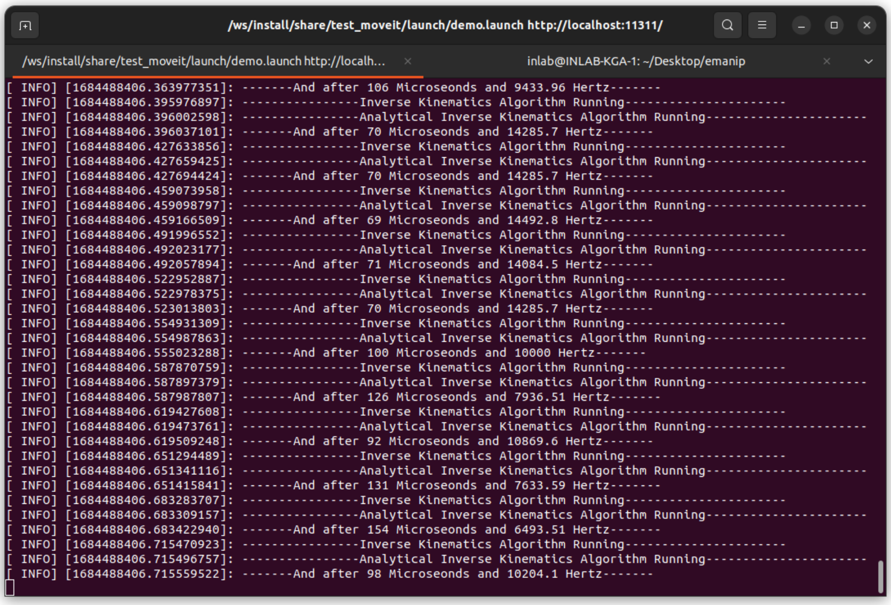


Figure 23: A snapshot of how the RViz GUI looks, notice that the end-effector is highlighted meaning that it is being moved by the user.



```
/ws/install/share/test_moveit/launch/demo.launch http://localhost:11311/
/ws/install/share/test_moveit/launch/demo.launch http://localh...    inlab@INLAB-KGA-1: ~/Desktop/emanip
[ INFO] [1684488406.363977351]: -------And after 106 Microseonds and 9433.96 Hertz-------
[ INFO] [1684488406.395976897]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.396002598]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.396037101]: -------And after 70 Microseonds and 14285.7 Hertz-------
[ INFO] [1684488406.427633856]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.427659425]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.427694424]: -------And after 70 Microseonds and 14285.7 Hertz-------
[ INFO] [1684488406.459073958]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.459098797]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.459166509]: -------And after 69 Microseonds and 14492.8 Hertz-------
[ INFO] [1684488406.491996552]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.492023177]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.492057894]: -------And after 71 Microseonds and 14084.5 Hertz-------
[ INFO] [1684488406.522952887]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.522978375]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.523013803]: -------And after 70 Microseonds and 14285.7 Hertz-------
[ INFO] [1684488406.554931309]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.554987863]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.555023288]: -------And after 100 Microseonds and 10000 Hertz-------
[ INFO] [1684488406.587870759]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.587897379]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.587987807]: -------And after 126 Microseonds and 7936.51 Hertz-------
[ INFO] [1684488406.619427608]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.619473761]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.619509248]: -------And after 92 Microseonds and 10869.6 Hertz-------
[ INFO] [1684488406.651294489]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.651341116]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.651415841]: -------And after 131 Microseonds and 7633.59 Hertz-------
[ INFO] [1684488406.683283707]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.683309157]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.683422940]: -------And after 154 Microseonds and 6493.51 Hertz-------
[ INFO] [1684488406.715470923]: ----------------Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.715496757]: ----------------Analytical Inverse Kinematics Algorithm Running----------------------
[ INFO] [1684488406.715559522]: -------And after 98 Microseonds and 10204.1 Hertz-------
```

Figure 24: A snapshot of terminal output whilst moving the end-effector.

10 Analysis

This section will provide an analysis of the earlier presented results. First and foremost the different methods to solve the inverse kinematic problem have been evaluated and compared to one another to be able to place a verdict on whether or not the method is sufficient to use in a real application. Furthermore, a brief analysis of the MoveIt-tool (which was used for implementation in this project) will be included, which will include what we felt were the advantages and disadvantages, as well as if the software package could have been used differently in order to achieve better results.

10.1 Inverse Kinematic Solutions

To be able to place a verdict on the developed solutions it makes the most sense to analyse each individual part, both the inverse kinematic solvers as well as the singularity path-planning (since this is essentially an extension of the inverse kinematics in this case).

10.1.1 Gradient Descent Method

The developed solution which used the Gradient Descent method could from the obtained results be deemed as a valid numerical option for solving the IK problem, seeing as it yielded a valid angle configuration for all well-behaved poses.

The interesting part with this specific method beforehand was the approximation that the transpose of the Jacobian essentially could work in the same manner as the inverse (with appropriate scaling). This approximation was made to ensure numerical stability in the presence of singular regions or singularities. The result shows that this was in fact the case, the method did not have trouble converging to a correct joint angle configuration when working within singular regions (as could be seen from Figure 19, which essentially is just moving within a singular region).

In addition to exhibiting robustness in regards to singularities the method also proved to be well-behaved when dealing with poor initial state approximations (as seen in Figure 20), but convergence was slow (in relation to having a good initial guess) which would negatively impact its use in real-time applications. Although it is worth mentioning that in real applications the gap between the initial- and final state is rarely as poor as in some of the testing since the complete solution in both joint-space and Cartesian space works with via-point creation, similar to that of the path generation in Section 8. By using the algorithm on the generated path it can be ensured that the initial guess is not too far from the actual answer.

Furthermore, when the method was implemented into the MoveIt package it was deemed that the algorithm could in fact run well in "real-time" meaning that convergence to a solution of the algorithm happened sufficiently fast enough so that a update frequency of over 20hz could be upheld (the testing was done according to Section 9.3.2). This would make the drawbacks of the algorithm essentially obsolete since the major trade-off to the stability of using the Jacobian Transpose is that the convergence time is affected. The convergence time increase relates to the fact that the Transpose of the Jacobian is a very rough approximation of the actual inverse.

One interesting aspect that was not explored within the limits of this work is that of a dynamic learning rate, the method we used was a constant learning rate decided by means of trial- and error. When choosing the learning rate it was obvious that a compromise needed to be taken since no specific rate was the best option for all different test cases this of course could be solved if one let the learning rate be dynamic instead of constant. Dynamic implementations of the learning rate have been mentioned in works regarding the Jacobian Transpose method and are said to provide an improvement to the convergence times of the algorithm [26]. Therefore this could be of use if one would like to further improve the computational speed, which in the case of this work was deemed good enough.

10.1.2 Gauss-Newton Method

In contrast to the Gradient Descent method the Gauss-Newton method showed to be a lot less numerically stable, it is especially sensitive to poor initial guesses.

The Gauss-Newton method for inverse kinematics uses the pseudo-inverse as an approximation of the regular Jacobian inverse. In contrast to the Jacobian transpose, the pseudo-inverse is a lot closer to the actual inverse of the Jacobian, which turned out to have both advantages and disadvantages.

The regular Jacobian inverse is heavily unstable in singular regions (meaning that it may yield extremely high values) and at the exact singular configuration the Jacobian is non-invertible, the pseudo inverse is meant to solve this problem by making the matrix invertible in the presence of singularities. What was found was that by using the pseudo-inverse (and also introducing a damping factor to the Jacobian inverse) the algorithm sometimes manages to find solutions in the proximity of singularities, however, the behaviour of the algorithm becomes very oscillatory and thus for some singular configurations convergence was not possible within the set amount of iterations (the oscillative behaviour is very clear in Figure 19.

This behaviour is further proven when using the inverse kinematic algorithm as a plugin for the MoveIt package. Within non-singular areas of the workspace, the algorithm achieves valid solutions within an appropriate time frame, making it so that the manipulator essentially moves in real time. However, in close proximity to singularities, the solver still has problems finding correct solutions within the specified time frame.

From the results, it could also be concluded that the initial guess plays a crucial part in the numerical instability of the algorithm. This however has less to do with the pseudo-inverse of the Jacobian and more to do with the fact that the Gauss-Newton optimization method is known for having convergence issues if the initial guess is at a large distance away from the optimum (as mentioned in Section 6.3). From the results in Figure 20, where the initial guess was located far off the joint-angle solution of the algorithm, the unstable behaviour is presented and as a consequence of this the method does not converge.

There are however some upsides to this method of solving the inverse kinematics, mainly when the task is located in the non-singular areas of the workspace and when used with the path generator for example which would ensure that the initial guess is at an appropriate distance in the joint-space. For these cases, the convergence times are generally faster than for the other numerical methods, as can be seen in Table 5.

Finally, the Gauss-Newton method was also implemented into the ROS package MoveIt is tested according to Section 9.3.2. Within MoveIt it could be concluded that the algorithm could work in "real-time", as long as the task which was to be executed was not located within singular-regions. The issue of bad initial guesses is essentially redundant in MoveIt since it works in a similar fashion to path generation where the inverse kinematic algorithms are at intermediate points with only a short distance between them.

10.1.3 Levenberg-Marquardt Method

The Levenberg-Marquardt method is supposed to offer the best of both worlds of the previous algorithms. Mainly that the convergence times should be faster than that of the Gradient Descent method whilst the convergence rate should be higher compared to the Gauss-Newton method.
By only studying the three graphs in Section 9.1.1 the anticipated behaviour could be confirmed since it is clear that the Levenberg-Marquardt method of solving the inverse kinematics is faster than the Gradient-Descent methods whilst still yielding convergence in all three cases (in contrast to the Gauss-Newton method).

However, when further tests were made (For the computational times in Table 5) the implementation of the algorithms was made in C++ (in relation to the previously implemented Python algorithms which were used to analyze the behaviour of each algorithm) to ensure that the tests yielded appropriate speeds which could be compared to having the algorithms run on the actual hardware. The results showed that once implemented in C++ the calculations times relative to that of the gradient descent algorithm were in-fact larger. By researching the reasoning behind this the conclusion was drawn that when the translation was not written in an optimized way making the additional matrix multiplications present in the Levenberg-Marquardt method decreased the computational speed so that the Gradient descent method in most cases yielded better performance. Seeing as the computational times still were appropriately fast to achieve the goal of running the algorithm at a minimum frequency of 50hz no further time was spent on optimizing the algorithm.

Furthermore if one were to improve this method the first thing is to look into writing a more optimized algorithm in C++ another big factor that has made a negative impact on the algorithm is that the damping parameter, which is described in Section 6.4.1, was made constant instead of dynamic. The reasoning behind this was to decrease the computational efforts of the algorithm since this was a concern early on in the project. Since the result showed that the algorithms perform extremely well against the computational time constraint one could instead look into having a dynamic learning rate. This could improve the overall performance since in singular regions the algorithm would have varied the damping parameter to more closely mirror the Gradient Descent behaviour which could have increased the possibility of convergence as well as in non-singular regions achieving faster calculation times. There are also a lot more ways the damping parameter can be introduced instead of the trial- and error method used in this thesis, many papers have researched the topic and in the paper "On damping parameters of Levenberg-Marquardt algorithm for nonlinear least square problems" [27] a multitude of examples for choosing the damping parameter is covered.

10.1.4 Analytical Method

Initially, it was clear that a well-working analytical approach to inverse kinematics would be superior in almost all instances. The issue at hand was that far from all manipulator configurations have existing analytical/closed-form solutions to the inverse kinematic problem. Fortunately, the analytical solution for this specific manipulator configuration was possible to derive.

Ensuring the validity of an analytical solution is actually a rather simple process, it's essentially just to compare the initial homogeneous transformation matrix (input/goal) to the forward kinematics of the returned joint angle configuration. For the solution presented in this report, this step is included in the solution selection since one of the few issues with the analytical approach is that it will (in a majority of cases) yield multiple solutions.

Our goal with the solution selection was to ensure that the manipulator achieves smooth movement and no abrupt changes in joint-angle configuration happened during task execution. This behaviour is essentially what is proven in Figure 21 and Figure 22. In Figure 21 it is shown that the manipulator has successfully followed the specified path, one may notice that the path is not a straight line which is a result of the singularity-free path generation to ensure that the manipulator does not traverse within singular regions. Furthermore, in Figure 22 the results show that each joint exhibits continuous movement in the joint space and no drastic joint changes are made (which could have been the case if the solution selection was faulty).

In addition to the above-mentioned results, it was concluded in Table 5 that for all the constructed test cases the analytical inverse kinematics found an appropriate solution. This result is important since a faulty analytical model of the inverse kinematics could in theory work very well in some areas of the workspace whilst missing solutions in other areas (for example by recalling to Section 7.3.1 it is mentioned that multiple equations exist for determining a single joint angle, solely dependent on which area the manipulator is working within).

It was also shown when implemented into MoveIt that this method yielded the best computational speed as well as proved to give stable results for almost all cases. The same result is also present when benchmarking for different poses. In other words, it proves to be much quicker than the numerical methods as well as more precise.

10.1.5 Path Generation

The path generation created in this master thesis aims to provide a solution to the last question within the problem description (Section 1.2). Even though the approach taken here is a simple one, it does satisfy the conditions of a singularity-free path generation in cartesian space (without generating a large computational load) which can be seen both in Figure 17 as well as in Figure 21 where the path was executed using the analytical approach to the IK-problem.

This approach to path generation will ensure that the movement of the manipulator is limited to that of non-singular regions, which is of great benefit to the overall system. Traversing singular regions with the manipulator can have a negative impact on the motion of the manipulator and even in some cases damage the hardware or its surroundings due to the abrupt increases in joint velocities (as described in Section 4.2).

Furthermore, since the method of generating the path is based on a user-defined threshold level, which determines where a singular region begins, it gives the user the option to weigh singularity-free movement with the path following accuracy.

Of course, with this simple implementation, there is no planning element to the path which would be beneficial, or in some cases necessary, if for example, obstacles are present within the manipulator workspace. But in its current state, it is deemed to be a good starting point for further development of a more complex path-planning approach.

10.2 MoveIt

MoveIt is generally an excellent tool for controlling robotic manipulators and provides an intricate interface for a user without knowledge of inverse kinematics and path planning. The interface provides a rather quick setup of the package and the GUI makes it easy to command the manipulator to execute certain tasks and move into different poses. The implementation part of an inverse kinematics plugin is also a rather straightforward process. The problems faced when using MoveIt are first and foremost the problem of not knowing much about the underlying code. It is absolutely possible to research much of the source code for Moveit to gather a deeper understanding, but knowing how everything connects is not a straightforward task. This becomes a problem when working with critical systems where each part of the code must be proven to work in a foreseeable manner.

Another problem encountered during the project is the fact that MoveIt defines coordinate frames and frame transformations according to how the robot URDF[4] is specified. This is in contrast to how we defined the coordinate frames for each link which is done according to the DH convention. The major difference this results in is that MoveIt defines all coordinate frames with the z-axis in the direction of the global z-axis while the DH convention specifies the z-axis in the direction of the

[4]URDF stands for Unified Robot Description Format which is a format for defining a robot structure

11 Conclusions

To draw conclusions from the presented work, it is beneficial to recall the questions that this master thesis aimed to solve (which are presented in Section 1.2).

The first posed question, and probably the most important as well, was in what ways the inverse kinematic problem could be solved for the eM1-7's manipulator configuration. The approach that was taken from the beginning was to study both numerical and analytical approaches to the problem, with the main reason being that there was no certainty that an analytical solution would exist.

The conducted work resulted in well-working numerical -as well as analytical solutions presented, where the analytical solution is superior due to its nature of being non-iterative as well as finding all possible solutions. It is also mentioned in the problem description that a valid solution to the problem should ensure numeric stability. Based on the validation principles that were run as well as how the solutions were formed it could be concluded that the final result would ensure numerical stability within the non-singular areas of the workspace. Even though the heuristic methods were not implemented, they presented a lot of advantages, such as avoiding problems with ill-defined Jacobians. It would in the future be interesting to compare the performance of these to the evaluated numerical ones.

Even though the analytical solution is far superior to its numerical counterparts having a multitude of solutions to choose from led to a more interesting result where the benefits of the analytical solution could be more clearly presented. This ties into the second posed question in the problem formulation regarding the performance of the algorithms. Regarding performance the key factor is the computational speed, ensuring that the algorithm can yield solutions at a fast enough rate to enable the manipulator to be controlled in essentially real-time speed. It can be concluded from the result that both the numerical and analytical methods provide excellent performance when it comes to computational speed.

Another important performance measure is the precision (and convergence) of the inverse kinematics solutions. Both of the presented methods (analytical and numerical) provide high accuracy, although they differ in how this accuracy is achieved (numerical methods have a set threshold whilst the analytical method is accurate in its nature).

Furthermore, the issue of the inverse kinematics yielding multiple solutions to a single task was also mentioned as something that should be answered within this report. For the numerical algorithms, there is an issue with the dependence on the initial guess and subsequently, if the correct local minimum will be found. To counteract this problem the introduction of a path generation where the steps between intermediate positions will in most cases yield the correct solution for the numerical algorithms but in extreme cases may not converge correctly. In contrast, for the analytical solution this is fully controlled by the algorithm and the solution is chosen based on principles set up in the algorithm which eliminates the problem of finding the appropriate solution.

The last thing that is highlighted in the problem description is the problem of generating paths with respect to singular regions. We believe that the provided solution and subsequently the presented results of the algorithm give an answer to the posed question and, even if the solution is a simple one, should effectively eliminate singularity issues when traversing within the workspace.

To summarize we believe that this report has presented a well-working methodology to solve the problem of the inverse kinematics for the eM1-7 manipulator, where using the analytical solution together with the singularity-free path generation could be applied to the manipulator with good effect. Additionally, it was proven that even though analytical methods are superior, it is fully reasonable to take a numerical approach to solve the inverse kinematic problem for this specific manipulator configuration.

References

[1] Donald Lee Pieper. *The Kinematics of Manipulators Under Computer Control.* Computer Science Department. Stanford Artificial Intelligence Laboratory - Stanford University: Stanford University, 1968.

[2] NetModule. *CAN.* `https://netmodule-linux.readthedocs.io/en/latest/howto/can.html`. Accessed: 2023-04-25.

[3] *MoveIt - Moving robots into the future.* `https://moveit.ros.org/1`. Accessed: 2023-04-25.

[4] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* New York: John Wiley and Sons, 1989.

[5] Kris Hauser. *Robotic Systems - Chapter 4: 3D Rotations.* Mar. 2018.

[6] J.J. Craig. *Introduction to Robotics: Mechanics and Control.* Third. Addison-Wesley series in electrical and computer engineering: control engineering. Pearson/Prentice Hall, 2005. ISBN: 9780201543612. URL: `https://books.google.se/books?id=MqMeAQAAIAAJ`.

[7] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control.* Cambridge University Press, 2017.

[8] Ruibo He et al. "Kinematic-Parameter Identification for Serial-Robot Calibration Based on POE Formula". In: *IEEE Transactions on Robotics* 26.3 (2010), pp. 411–423. DOI: `10.1109/TRO.2010.2047529`.

[9] Samad A. Hayati. "Robot arm geometric link parameter estimation". In: *The 22nd IEEE Conference on Decision and Control.* 1983, pp. 1477–1483. DOI: `10.1109/CDC.1983.269783`.

[10] W. Veitschegger and Chi-Haur Wu. "Robot accuracy analysis based on kinematics". In: *IEEE Journal on Robotics and Automation* 2.3 (1986), pp. 171–179. DOI: `10.1109/JRA.1986.1087054`.

[11] H. Zhuang, Z.S. Roth, and F. Hamano. "A complete and parametrically continuous kinematic model for robot manipulators". In: *Proceedings., IEEE International Conference on Robotics and Automation.* 1990, 92–97 vol.1. DOI: `10.1109/ROBOT.1990.125952`.

[12] Saeed B.Niku. *Introduction to Robotics - Analysis, Control, Applications Third Edition.* Hoboken, NJ : John Wiley & Sons, Inc., 2020.

[13] A. Balestrino, G. De Maria, and L. Sciavicco. "Robust Control of Robotic Manipulators". In: *IFAC Proceedings Volumes* 17.2 (1984). 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984, pp. 2435–2440. ISSN: 1474-6670. DOI: `https://doi.org/10.1016/S1474-6670(17)61347-8`. URL: `https://www.sciencedirect.com/science/article/pii/S1474667017613478`.

[14] W. A. Wolovich and H. Elliott. *A computational technique for inverse kinematics.* 1984. DOI: `10.1109/CDC.1984.272258`.

[15] Charles W. Wampler. "Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods". In: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1 (1986), pp. 93–101. DOI: `10.1109/TSMC.1986.289285`.

[16] Yoshihiko Nakamura and Hideo Hanafusa. "Inverse kinematic solutions with singularity robustness for robot manipulator control". In: *Journal of Dynamic Systems Measurement and Control-transactions of The Asme* 108 (1986), pp. 163–171.

[17] Matilda Richardsson. *Most efficient Inverse Kinematics algorithm for Quadruped models.* Stockholm, Sweden, 2022.

[18] Andreas Aristidou and Joan Lasenby. “FABRIK: A fast, iterative solver for the Inverse Kinematics problem”. In: *Graphical Models* 73.5 (Sept. 2011), pp. 243–260. ISSN: 1524-0703. DOI: `https://doi.org/10.1016/j.gmod.2011.05.003`.

[19] Phillipe Santos et al. “M-FABRIK: A New Inverse Kinematics Approach to Mobile Manipulator Robots Based on FABRIK”. In: *IEEE Access* 8 (Jan. 2020), pp. 208836–208849. DOI: `10.1109/ACCESS.2020.3038424`.

[20] L.-C.T. Wang and C.C. Chen. “A combined optimization method for solving the inverse kinematics problems of mechanical manipulators”. In: *IEEE Transactions on Robotics and Automation* 7.4 (1991), pp. 489–499. DOI: `10.1109/70.86079`.

[21] Bruno Siciliano and Oussama Khatib. *Robotics: Modelling, Planning and Control.* Springer, 2016.

[22] Alexander J. Elias and John T. Wen. “Canonical Subproblems for Robot Inverse Kinematics”. In: *IEEE Access* (Oct. 2022).

[23] Jacob Rosen. *Advanced Robotic - MAE 263B.* Jan. 2019.

[24] Li Jiang et al. “An integrated inverse kinematic approach for the 7-DOF humanoid arm with offset wrist”. In: *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (2013), pp. 2737–2742.

[25] Mathias Brandstötter, Arthur Angerer, and Michael W. Hofbaur. *An Analytical Solution of the Inverse Kinematics Problem of Industrial Serial Manipulators with an Ortho-parallel Basis and a Spherical Wrist.* 2014.

[26] Samuel R. Buss. *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods.* 2009.

[27] A O Umar et al. “On damping parameters of Levenberg-Marquardt algorithm for nonlinear least square problems”. In: *Journal of Physics: Conference Series* 1734.1 (Jan. 2021), p. 012018. DOI: `10.1088/1742-6596/1734/1/012018`. URL: `https://dx.doi.org/10.1088/1742-6596/1734/1/012018`.

Appendices

A Test Poses for Benchmarking

Test #	Position			Quaternions			
	x [m]	y [m]	z [m]	x [-]	y [-]	z [-]	w [-]
1	1.5230	-0.1267	0.0815	-0.5015	0.4979	-0.4991	0.5015
2	1.5041	-0.0843	0.1244	-0.4999	0.4965	-0.4999	0.5037
3	1.4462	-0.1011	-0.2105	-0.6974	-0.1681	-0.2430	0.6529
4	1.4610	-0.0400	0.2920	-0.5020	0.4980	-0.4980	0.5020
5	1.1980	-0.5950	0.3110	-0.5050	0.4940	-0.4960	0.5030
6	1.2360	-0.3160	0.5650	-0.5070	0.4910	-0.4950	0.5060
7	0.9780	-0.0940	1.0070	-0.5090	0.4890	-0.4940	0.5070
8	0.9770	-0.0940	1.0050	-0.7030	0.0060	-0.0710	0.7050
9	0.8180	-0.4960	1.0760	-0.3000	0.0290	-0.0910	0.9490
10	0.0440	1.3560	0.3640	-0.3030	0.0260	-0.0890	0.9480
11	-0.3110	1.3760	0.6160	-0.3070	0.0200	-0.0860	0.9480
12	1.2510	-0.0220	0.4460	0.5650	-0.5080	0.4580	-0.4620
13	1.4580	-0.4510	0.3640	-0.5040	0.4960	-0.4970	0.5030
14	1.3990	0.2100	0.4340	-0.2650	0.4600	-0.5250	0.6650
15	1.2730	-0.3690	0.4960	-0.7340	-0.1380	-0.2790	0.6030
16	1.2050	-0.6340	0.6640	-0.7120	-0.0720	-0.3130	0.6240
17	1.2370	-0.3640	0.4300	0.5870	-0.5550	0.4300	-0.4030
18	1.2360	-0.3640	0.4300	-0.3720	0.3380	-0.6160	0.6070
19	1.5321	0.0035	0.1586	-0.5014	0.4986	-0.4982	0.5016
20	1.5555	0.1073	0.2355	-0.5033	0.4965	-0.4968	0.5034
21	1.5477	0.1023	0.3089	-0.5047	0.4951	-0.4957	0.5045
22	1.5324	0.0968	0.3046	-0.3599	0.3529	-0.6026	0.6187
23	1.5350	-0.1861	0.3000	-0.5295	0.1484	-0.4628	0.6953
24	1.3528	-0.1917	0.6136	-0.5308	0.1445	-0.4623	0.6955

Table 6: Positions used in each of the test cases for the calculation time tests