BACHELOR

Trajectory generation for collaborative robotic manipulators

Jansen, Tim G.A.J.

Award date:
2022

Department of Mechanical Engineering,
Control Systems Technology Section

Trajectory generation for collaborative robotic manipulators

Tim Jansen
1328026

Project Supervisors:
Dr. Ir. E. Torta
Msc. B. Sen

6th July 2022, Eindhoven

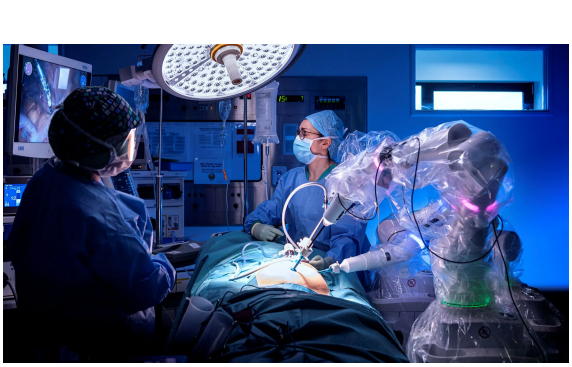
Contents

List of symbols

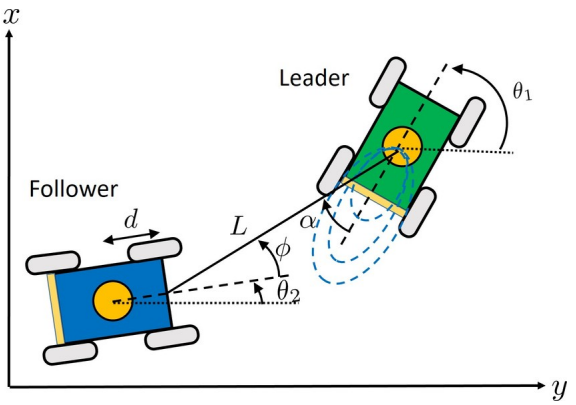
Symbol	Meaning	Unit	Abbreviation
a	Acceleration	Meter per second squared	[m/s^2]
d	Distance	Meter	[m]
e	Error	Meter	[m]
F	Force	Newton	[N]
f	Frequency	Hertz	[Hz]
J	Jerk	Meter per second cubed	[m/s^3]
T	Torque	Newton meter	[Nm]
t	Time	Second	[s]
v‘	Velocity	Meter per second	[m/s]
ω	Rotational velocity	radians per second	[rad/s]
ω_n	Natural frequency	radians per second	[rad/s]

1 Introduction

The world we are living in nowadays relies on technology more and more. A particular sector in which technology will have an increasingly larger role is the medical sector. In current times of COVID-19 it has become even more apparent how complicated the deployment of medical staff can be. Technological advancements which would enable robots to take over certain medical procedures are therefore highly demanded. Use cases of these kinds of robotizations in the medical sector are endless [1]. Examples range from basic scanning procedures to complete robot-performed surgeries. Many medical interventions require multiple actions to happen at once. A possible way to realize a multitude of actions would be to use multiple robots at the same time. In such a configuration it is increasingly more complex to operate correctly as all actions rely on each other to work perfectly. An example of such a task can be seen in Figure 1.1a. A way of controlling such a multi-robot system is through the principle known as Leader-Follower robot design. Leader-follower robots have already been extensively implemented for mobile robots in many different sectors, [2], [3], such as the warehousing sector, see Figure 1.1b. In the medical sector, however, the more applicable robot type is the manipulator robot. Therefore, in this report, the focus will be on the control of leader-follower manipulator robots.



(a)



(b)

Figure 1.1: (a): Two manipulator robots in action during a surgical intervention [4]. (b): Mobile robots in a leader-follower based design [5].

As said, this report will focus on the control algorithm design of a leader-follower configuration of two manipulator robots for the medical field. Accuracy and reliability are important in all sectors but for medical use these are of utmost importance. In order for the robots to function accurately and predictable, a good control algorithm is desired.
In the first part of this report such a control algorithm will be designed and tested on two Franka Emika Panda robots, commonly referred to as Franka robots. These Franka robots have seven Degrees of Freedom, which enable them to move through their workspace with relative ease and precision [6]. This makes these robots a very good option for this particular use case.

2 Problem definition

In order to systematically solve the proposed use case of medical leader-follower control, it is important to break the objective down into concrete research questions.

2.1 Main research question

The main objective of controlling robots is to control them with high accuracy. In this case the robots have to be controlled in a leader-follower principle, which requires an extra level of accuracy. Not only do the robots have to be accurate separately, but also the transformation between the two robots has to be controlled with great accuracy. A leader-follower robot design allows for many different operations to be carried out, such as x-ray scans. If robots were to perform such scans the process could be broken down into three main movements. First, moving towards the to-x-ray location; next, moving around the to-x-ray position to make the scan; and lastly, moving away from the patient again. Taking all these different tasks into account, it is clear why it is of importance that many different types of reference trajectories can be tracked by the robots. In order to evaluate its capabilities for these different movement requirements the robots will be tested on the following few trajectories.
A trajectory where both robots move towards- or away from a shared goal; a trajectory where both robots move around a shared goal.
In order to define accuracy in these trajectories, a position error goal of 1 [mm] will be set. Apart from the position of the robots' end effectors, its orientations are just as important. For the x-ray scan use case, it is most important that the end effectors point towards each other to perform a proper scan. Therefore this orientational requirement is also set for the proposed trajectories. Performance and accuracy are not the only important factors to be considered. Safety of operation is of equal importance. Especially for robots working in the proximity of humans.
Taking these requirements into account, the main research goal of this report can be established.

How can you accurately control two robots performing trajectories based on the leader–follower principle? The trajectories will be moving towards-, away- and around a shared goal with the end effectors pointing towards each other. These trajectories should be performed with a maximum position error of 1 [mm]. On top of that, the trajectory control algorithm should be able to ensure safety.

To accomplish accurate trajectory tracking a few major important features are desired. A good position control algorithm is required. This algorithm should allow for stable and high-performance position tracking [7]. A good position algorithm should have a good balance between accuracy and the time-spent getting to proposed target (i.e. Trajectory time-scaling) [8], [9], [10]. In order to assist position control into accurately controlling a trajectory, velocity control should also be implemented [11]. Velocity control will greatly help track the trajectory. On top of that, an algorithm should be put into place which helps to transform the desired position goals into smooth trajectories which take into account velocities along the way. This type of trajectory will help both position- and velocity control.
In order to ensure safety, the trajectory generation should have a technique of avoiding collisions [12], [9], [13].

With the help of the current control and leader-follower literature, a hypothesis on this main research question can be defined.

In order to properly track the proposed trajectories it is important to transform the target waypoints to smooth point-to-point trajectories which will enable a control strategy based both on position and velocity control of the robots. The trajectory should be able to avoid collisions and trade-off accuracy and velocity.

2.2 Secondary research questions

In order to systematically solve the main research question it is important to break down the main aspects of the question into smaller subquestions.

2.2.1 Trajectory profile

First of all, the trajectory generation itself. It is proposed that a smooth trajectory should be used which enables both position and velocity reference signals so that they can be used for position- and velocity control. This forms the following research question.

What trajectory profile is most optimal to allow the robots to be controlled in position and velocity?

Looking at the current literature for trajectory profiles, the most popular trajectories seem to be the trapezoidal- and polynomial profiles. In the motion profile theory itself, there are also different options which can be carefully tuned and optimized to fit the desired outcome [14], [10].
The defined goal of such a trajectory profile, is to generate a smooth movement which is able to define both position and velocity. In order to verify smoothness of end effector movement, research has shown that, the displacement, velocity, acceleration and jerk of the end effector should all be continuous curves [15], [16].
Comparing the theory of these trajectory profiles with the evaluation of smoothness method, a hypothesis can be made.

In order to both accomplish position and velocity control a trajectory method where these parameters are not predetermined is required. This is most applicable in a polynomial profile. The order of polynomial should be chosen to satisfy continuous displacement, velocity, acceleration and jerk.

2.2.2 Leader-follower principle

On top of the trajectory tracking control of the robots themselves, the leader-follower algorithm is just as important. Research has shown that there is not just one way of defining the leader-follower principle. In contrary, there are many different options for a leader-follower connection [17]. This makes it an important research topic and results in the following research question.

What sort of leader–follower principle is most applicable in this medical use case?

The proposed options for leader-follower options can be expressed as follows.
The option of determining the real-time reference signal based on both robots' errors. This is important for cases where the transformation between the robots is the main point of interest. This theory could be compared to the well-researched equal hierarchy type of collaboration between robots [18], [19].
Another option is the option of determining the robots velocities independently. If one of the robots is having an 'easier' time tracking the trajectory it can speed up itself, after which the other robot will try to catch up once it has come to a less difficult part itself. This is useful in cases where both robots do need to accurately follow their own trajectories but not necessarily at the same time.
In other cases it would also be possible to only take one of the robot's position and velocity into account [20]. Such an option is particularly useful in cases where the transformation between both robots is not as important anymore.
With these known options a hypothesis for the research questions is established.

The leader-follower principle will take on different strategies dependent on the current movement. Moving away- and towards a goal can be done at different speeds for the robots as this will not require the robots to have an exactly defined transformation between each other. During the scanning phase, this transformation is, however, very important. Therefore, for the moving around a goal trajectory, it will require the controller to take into account both robots and their transformation.

2.2.3 Non-identical robots

Another interesting research challenge is to look at leader-follower systems with non-identical robots. As proposed in the introduction, this research will focus on using two identical Franka robots. But to broaden the scope of this research, it is also very interesting to look at non-identical robots (i.e. heterogeneous robots) in a leader-follower configuration [21].

What happens to the controllability and accuracy if the follower robot is not the same as the leader robot?

Similarly, what, even though the robots are identical, would happen if one of the robots were to carry an additional load. Would the original control algorithm still function properly, or will it require the same control strategy as two non-identical robots? Researching questions like this would to help expand the use case from only doing x-ray scans to actual other tasks involving the positioning of other objects.

What happens to the controllability and accuracy if one of the robots has to position another object, causing the robots' end effector masses to be different?

Looking at the current literature regarding leader-follower robots it becomes clear that it is not very common to use two different types of robots. However, research has shown that robots operating in a leader-follower fashion can properly be controlled even though they operate in different environments, i.e. different loads or tasks [22]. According to this literature they can be accurately controlled if the separate robot controllers are robot specific. This means that the controller for the leader and follower should not necessarily be the same but must be tailored towards the current environment the robot is operating in. With this knowledge the following hypothesis is made.

If a correct control architecture is designed, the accuracy should not be influenced by the type of robot, or its environment, as long as the desired waypoints are in the robot's workspace and the independent joint accuracy is of the same order. A 'correct' control architecture would in this case consist of a controller which takes into account the difference in dynamics, the position and the velocity of both robots, when making its trajectory choices.

2.2.4 Operating close to robot singularities

Another difference which can be observed between the leader and follower robot is when one of the robots, in particular the follower robot, is operating close to its workspace border. When a robot is operating close to its workspace border, it might behave differently as it will be less accurately manoeuvrable as it will near its singularity. Singularity regions will cause the end effector velocity to be very high for very low joint velocities, which in turn will make controlling the robots more difficult [23]. The consequences of this phenomenon would be interesting to evaluate as it is very likely to happen in real life situations if no careful thought is put into it. Therefore, a research question is dedicated to this subject.

What happens to the controllability and/or accuracy when the follower robot is tasked to operate close to its workspace boundary (i.e. singularity region).

Looking at the literature regarding the control of robots near their singularities it becomes clear that the main solution is to completely avoid getting close to singularities [24], [25]. This would be most desirable for any trajectory. However, in the case of leader-follower control, it would be interesting to see what would happen to the leader-follower relation whenever such a singularity was not avoided by the follower. The hypothesis for the question is the following.

The follower robot will have difficulties controlling itself precisely when it is near its singularity. This will cause its tracking error to converge to zero slower. This in turn will cause the robot to perform its trajectory slower. Which, if designed for, will limit the speed of the leader-follower trajectory as well.

2.2.5 Autonomous navigation

A very interesting type of trajectory control is the so-called autonomous navigation control. In this use case, this would translate to only telling the robot its final goal, like a body part of a patient. With that goal, it will determine by itself how it will move towards it. This is very interesting as many control algorithms, especially leader-follower control, will use the knowledge of future steps or waypoints in the determination of the robots trajectory. Therefore, seeing if it is possible to completely let the robot take over the navigation side of the control, would be very interesting. The research question regarding autonomous navigation is a follows.

To what extent are leader-follower systems able to implement autonomous navigation. Also, what sort of differences in terms of accuracy and performance can be observed between a pre-determined trajectory or a autonomously generated trajectory?.

The most important principle of autonomous navigation in this use case, is to avoid collisions with other robots or objects in its environment. In order to accomplish this, a way of understanding its environment is required. This can either be solved using real-time sensors [26], [27], or a predetermined mapping of its environment. Based on this mapped environment, a collision-free trajectory should be chosen to get to the desired end effector pose [12], [9], [13]. In the case of medical robots, there are multiple options of tasks the robots will have to fulfill. Depending of the type of operation the robots' environment might be constant or might be changing over time. For a constant environment, a predetermined mapping of the environment would be sufficient as this allows for lower computational costs. For a dynamically changing environment, however, real-time sensors will have to be used to map the environment.

As this mapping of the environment and collision avoidance will increase the complexity of the control system, it can be assumed that the performance of the robots will decrease whenever they are tasked to autonomously navigate towards the desired pose, as opposed to simply following a predetermined trajectory.

3 The control and simulation architecture

To solve the proposed problem it is important to define a control and simulation architecture. This architecture will function as the foundation upon which the control theories can be designed and tested.
This chapter will focus on explaining all the parts which the control and simulation architecture consists of.
At the end of this chapter, this architecture is also related to the repository where all the files can be found.

3.1 The programs

To build the proposed control and simulation environment, certain coding or simulation programs are required. For this particular research, Matlab and Simulink are used. The choice to use these is based on the fact that it is a well-known, user friendly, environment. This ease-of-use is particularly important, as it enables the user to easily test multiple different control theories.

3.2 The overview

The control architecture is based on a standard closed loop system [28],[29], but it is modified to function in leader-follower control.
A clear overview of the control and simulation structure can be seen below. In the next subchapters, the separate parts of this structure will be explained.

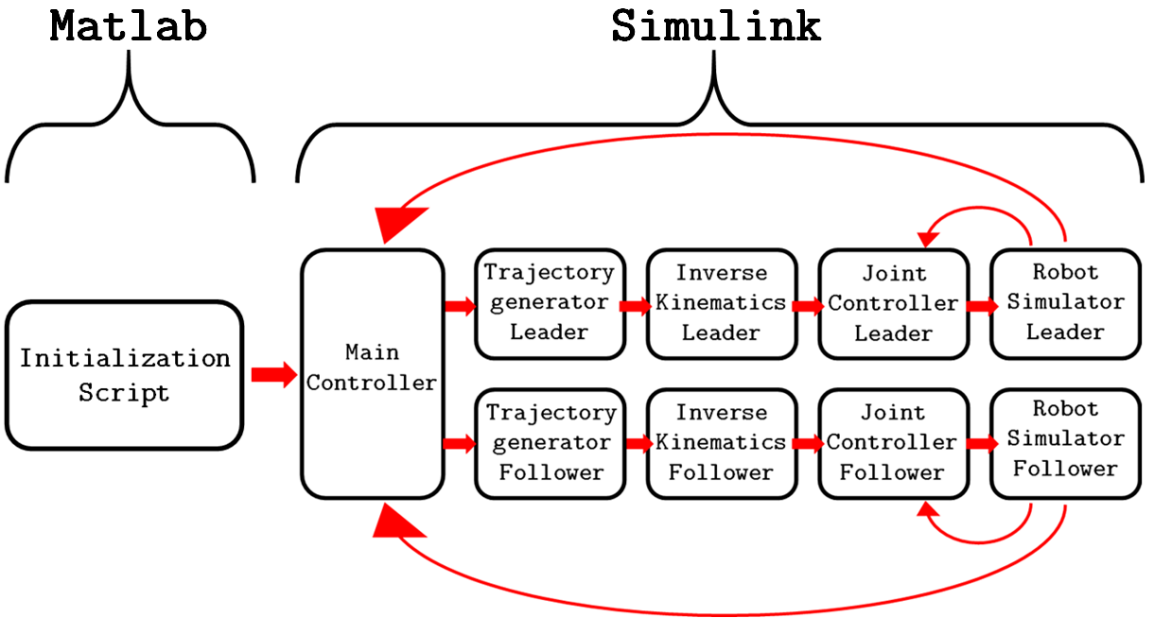


Figure 3.1: A visual representation of the proposed control and simulation architecture

3.3 Initialization script

The initialization script is the origin of the complete simulation. In this script, the main goal of the simulation is defined.
As seen in Figure 3.1, this part of the architecture is made in Matlab. The goal of this part of the system is to initialize a few variables into the Matlab workspace. These variables are essential as the rest of the simulation is build upon them.
The variables which it generates are the following.

- The robot specifications, based on its URDF [30].
  This includes things like dimensions, joint limits, gravity vectors, and much more.
- Waypoints along the desired trajectory for the leader robot.
  The desired trajectory is user defined as well as the amount of samples along this trajectory.
- The position and orientation of the follower base, with respect to the leader base.
  This is also a completely user defined variable, see Figure A.1.
- The position of the leader base, with respect to the 'world'.
  Again, this is up to user preference.
- The desired distance and orientation between the leader- and follower end effectors.
  User defined as well.
- The initial robot configuration for leader and follower robot. Auto-computed by the script. Based on the first leader waypoint and the above introduced positional relations between leader and follower robot.

As explained in the list above, many of the initialized variables are made from user defined choices. This is useful as it allows the control researcher to test and debug multiple kinds of situations. But it is especially important as it will allow the end user to easily configure their desired setup for the particular operation.

3.4 Main controller

The main controller is the brain of this complete architecture. This controller will use the previously initialized variables, together with additional user defined options to determine the current robot goal.
These 'extra' user defined options relate to the different scenarios explained in the problem definition chapter. Examples of such options could be the particular leader-follower algorithm, or the maximum velocity, or the choice to time-scale the trajectory or not. These kinds of options will be taken into account by the controller to determine the current robot goal. These additional user options will be elaborated in the next chapters.
As mentioned, the goal of this controller is to determine the current robot goal. This current robot goal specifically means, the current position and velocity goal for the leader and follower end effectors.

3.5 Waypoint-to-waypoint trajectory generator

As explained, the output of the main controller consists of a signal in which the current position and velocity goal is determined. However, as shown in the literature, in order for a robot end effector to follow a smooth trajectory, its displacement, velocity, acceleration and jerk should all be continuous curves [15], [16]. This would not be the case if it were to immediately send the new position and velocity goals, as defined by the main controller, to the robot interface.
That is where the importance of this waypoint-to-waypoint trajectory generator comes in. This generator should be able to transform these position and velocity goals into a smooth trajectory. Which would ideally be continuous in position, velocity, acceleration and jerk.

3.6 Inverse Kinematics

After a smooth, continuous, trajectory has been generated, a desired end effector position is outputted. In order for the robot to actually be able to move towards that end effector position, it has to know what joint configuration enables this position. This cannot only be accomplished on positional level, but also on velocity level. That task is solved in this part of the architecture.

3.7 Joint controllers

Now that the desired joint positions and velocities are known it is important to convert this into joint torques. This is called a joint controller. Ideally, this joint controller should be tuned specifically to the behaviour of the robot. This controller should also consist of a feedforward and feedback part for optimal results.

3.8 Robot simulator

At last, the robot simulator. This part of the structure is tasked with simulating the behaviour of the robot as how it would function in real life. Ideally, this simulator should mimic the real robot and environment as close as possible.
The resulting motion and behaviour of the robot is then fed back into the main controller and joint controller, as seen in Figure 3.1, to finish the closed-loop control architecture.

3.9 Link to repository

As described, these simulations work with a Matlab and a Simulink file combination.
In my repository I have proposed four simulations. These four simulations are the following.

- Simulate a single Franka robot
- Simulate a dual Franka robot setup
- Simulate a Franka robot and KinovaGen3 robot combination
- Simulate a Franka robot and Universal UR10 robot combination

Simulate a single Franka robot:

- Matlab file: panda_trajectory.m
- Simulink file: single_robot.slx

Simulate a dual Franka robot setup:

- Matlab file: panda_trajectory_patient.m
- Simulink file: panda_trajectory_sim_patient.slx

Simulate a Franka robot and KinovaGen3 robot combination:

- Matlab file:panda_trajectory_Heterogeneous_Kinova.m
- Simulink file: panda_trajectory_Heterogeneous_Sim_Kinova.slx

Simulate a Franka robot and Universal UR10 robot combination:

- Matlab file: panda_trajectory_Heterogeneous_Universal.m
- Simulink file: panda_trajectory_Heterogeneous_Sim_Universal.slx

Where to find these files specifically, and how to use them is all explained in the README file, which is also in the repository.

4 The process

To properly solve the proposed research questions it is important solve the problem in a systematical fashion. This means that the problem should be solved by testing different literature based theories, on many of the aspects of the project. After implementing a single approach, its result should be carefully evaluated. Based upon this result, a choice on the implementation of that specific approach shall be made.
This chapter will focus on explaining this systematical process for every part of the previously proposed architecture.

4.1 Initialization script

As explained in the previous chapter the goal of this part of the structure is to initialize some important values for the simulation.

4.1.1 The robot specifications, based on its URDF

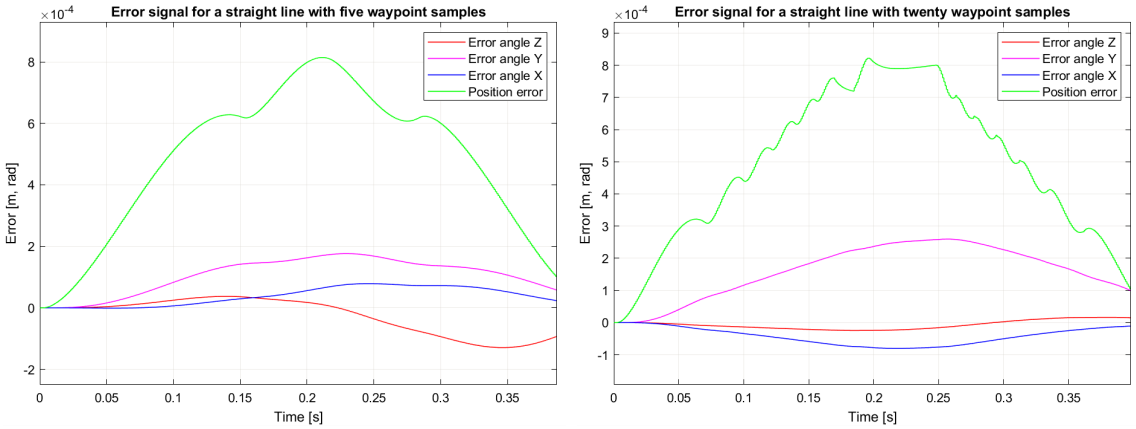
As the title suggests, the main robot specifications are retrieved from an external file, the so-called URDF. The URDF allows the user to easily import all of the important robot specifications. This subject required no systematical choice process.

4.1.2 Waypoints along the desired trajectory for the leader robot

This is a very important variable as it definitively determines the goal of the simulation. This variable can be influenced by the desired trajectory itself and by the sampling rate at which waypoints are placed along the trajectory.
In the design of this initialization script, it is important to evaluate what sort of waypoint sampling rate is appropriate and what sort of influence it has on the simulation.

First of all, it is easy to understand that a low number of waypoints poorly represents a complex path, see Figure A.2 for a exaggerated example of this. However, there also exist less complex paths such as straight lines, which in theory would require only two waypoints. Therefore, in terms of accurately representing a continuous path, it is important to take into account the complexity of the path when deciding upon the waypoint sample rate.

On top of representing the continuous path, the amount of waypoints might also influence the control behaviour. To investigate this influence, a test was done where a simple straight line path was simulated using either five waypoints or twenty waypoints. A straight line path is important here as you can factor out the, previously introduced, misrepresentation factor.
The results of these simulation can be evaluated by looking at both the accuracy and the performance. Accuracy will be quantified using the tracking error, whilst performance will be quantified using the total time the robot took to finish the trajectory.



(a) (b)

Figure 4.1: (a): Error signal with only five waypoints on a straight line trajectory.
(b): Error signal with twenty waypoints on a straight line trajectory.

From the error comparison seen in Figure 4.1, it becomes clear that the controller is influenced by the number of waypoints. As can be seen, the error signal with more waypoints has a more rugged profile as the controller keeps adjusting itself at every waypoint. The reason for this will become more clear in subsection 4.2. Apart from the more rugged profile, both simulations exhibit the same amount of accuracy as both simulation have a similar error quantity.
In terms of performance, the five waypoints trajectory was completed 1.76% faster compared to the twenty waypoints trajectory. This seems to be a negligible difference but could be taken into account if performance wants to be optimized.
Apart from accuracy and performance, no major difference can be observed. However, it is important to understand that because of the controller's behaviour, a higher amount of waypoints might lead to better controllability. This has to do with the fact that the controller mainly evaluates the robot performance at the waypoints. So more waypoints would in turn mean more instances where the controller can adjust or evaluate. Reasons for this will become more clear further on into this report once the controller has been explained.

All in all, from these kinds of tests it had become clear that in terms of accuracy or performance no major difference are observed between the number of waypoints. In terms of controllability and trajectory representation, however, it is noted that a relatively higher amount of waypoints is preferred.
Once this was discovered in this process, a choice was made to have distance between every waypoint of around 0.03 [m]. This allowed for minimal representation errors and good controllability, whilst keeping the amount of waypoints reasonable.

4.1.3 Positioning of both robots in space and their end effector relations

As introduced in the previous chapter, a lot of the user input in the initialization script had to do with the positioning of the robots in space, with respect to each other, and their end effector relations. The intention behind this is to have a user-friendly simulation program in which almost every imaginable situation can be reconstructed in the simulator.
However, this comes with its challenges. If almost every variable is adjustable, the relation between leader waypoints and follower waypoints also changes. So a robust way of relating this leader-to-follower waypoint conversion to these user-inputs had to be constructed.
Concretely, the goal is to setup of way to convert, the leader end effector pose w.r.t. the leader base, to the follower end effector pose w.r.t. the follower base.

With all of these user-inputs, both position and orientation can be chosen. A useful way of expressing position and orientation together, is with the use of transformation matrices.

The relevant transformation matrices for the desired waypoint conversion can be seen in Figure 4.2.

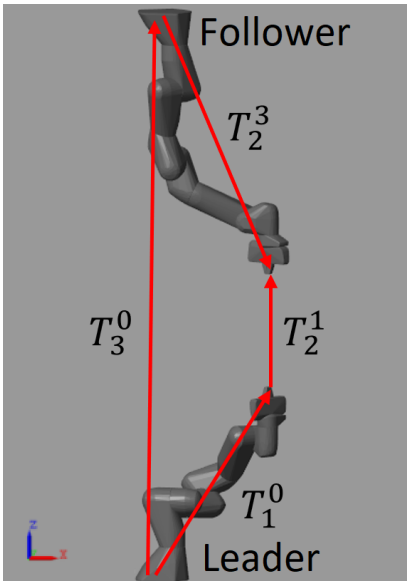


Figure 4.2: A visual representation of the relevant transformation matrices required for the waypoint conversion process.

Where:

- T_1^0 is the transformation matrix representing the end effector pose of the leader robot (target) w.r.t. the leader base (reference).
  This is retrieved from sampling the, user-inputted, trajectory.
- T_2^1 is the transformation matrix representing the relation between leader- (reference) and follower (target) end effector
  This is retrieved from desired distance and orientation between end effectors, inputted by the user.
- T_3^0 is the transformation matrix representing the position and orientation difference between leader- (reference) and follower (target) base.
  This is retrieved user input as well.
- T_2^3 is the transformation matrix representing the end effector pose of the follower robot (target) w.r.t. the follower base (reference).
  This is the to-be-computed transformation matrix.

Then with the use of transformation matrix computations [31], Equation 4.2 can be constructed

$$T_2^0 = T_1^0 T_2^1 = T_3^0 T_2^3 \tag{4.1}$$

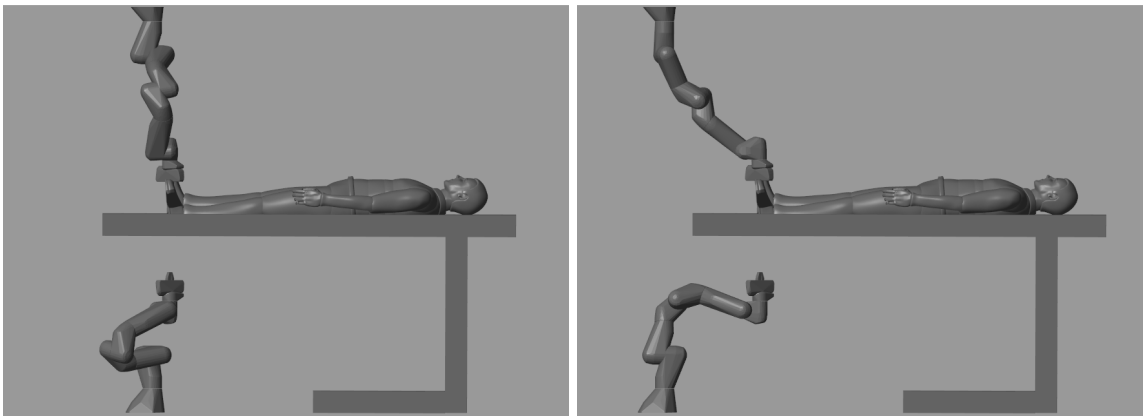
Then, since T_1^0, T_2^1, T_3^0 are known, one can easily solve for T_2^3.

$$T_2^3 = (T_3^0)^{-1} T_1^0 T_2^1 \tag{4.2}$$

Once this transformation matrix is computed, the end effector position and orientation can be retrieved for the follower
This method of leader-to-follower waypoint conversion has been tested for many different situations and has proven to work every time.

Once the relations between leader and follower position and orientation have been established, it is finally time to place them both in the environment. For a medical use case, a possible environment

could be a surgery room. For trajectory demonstration purposes, a room with a patient on a surgery table was chosen.
The robots are placed within the environment based upon the, user inputted, trajectory coordinates. However, there is still room for deviations if only the end effector position, inputted by the user, is taken into account. Looking at Figure 4.3, it can be seen that two totally different robot configurations are still able to have the same global end effector position, by altering the positioning of the robot bases.



(a) (b)

Figure 4.3: (a): Robot base positioned close from end effector position. (b): Robot base positioned further from end effector position.

In order to optimize the positioning of the robots together with the desired global end effector goal, a manipulability optimization is very useful.
Manipulability refers to how well a robot is able to manipulate its end effector position and velocity at a certain robot configuration. A possible way of quantifying this is with the use of the robots geometric jacobian ($\mathbf{J}$) [32], [33]. A common way of computing this manipulability is by taking the determinant of the jacobian matrix ($\mathbf{J}$) at a certain robot configuration. However, as the Franka robot is a redundant, 7DOF robot, the jacobian matrix is non-square, which disallows the determinant to be computed. A solution to this is to calculate the determinant of $\mathbf{J}\mathbf{J}^T$ [32].
Such a manipulability analysis can be done by altering the end effector position w.r.t. the robot base and evaluating its manipulability over all waypoints in the trajectory.
This optimization process was done as an example for a straight, upwards, trajectory where the to-be-optimized coordinate was the x-direction. By altering the end effector x-position w.r.t. the robot base, its manipulability over the trajectory could be analyzed.
In Figure A.3, the results of the manipulability analysis can be found. By doing this analysis, one could find the most optimal x-coordinate to perform the trajectory.
In this example case, that would be x-coordinate of 0.35[m].
With the knowledge of the desired, user-inputted, global, trajectory coordinates and the knowledge of the most manipulable end effector coordinates w.r.t. the leader base, an optimal placement of the robots within the environment could be computed.

The reason one would want to optimize for manipulability is for a number of reasons of which two reasons seem to be most important for this research [34]. First, as mentioned earlier in subsection 2.2, there are multiple medical use cases imaginable which include autonomous navigation. More specifically, collision avoidance. If for example, halfway through the trajectory an obstacle is detected and must be avoided, it is very important that the robot is able to reposition itself easily. Therefore ensuring an optimal amount of manipulability along the trajectory is very important for robots to easily change their trajectory as it can be assumed that it is not close to its singularities, whenever its manipulability is sufficiently high.
In this case, the use of this manipulability analysis could be compared to a reachability analyses,

which is described in literature in connection with autonomous navigation [35].
Secondly, as also mentioned in the problem statement, other interesting research topics include heterogeneous robot combinations. In such a combination it is important to be able to evaluate manipulability as it could give the researcher an understanding of this other robot's workspace or singularities. Therefore, if one was to replace one of the Franka robots with a different, possibly less maneuverable, robot, they could analyze and optimize its manipulability in order for this new robot to function similarly to the Franka robot.

4.1.4 Initial robot configuration

The last important variable to initialize is the initial robot configuration.
This is needed in the simulation in order for the robot to start at the beginning of the trajectory instead of at its home configuration.
This initial robot configuration is calculated for both leader and follower using an inverse kinematics function.
If the robot were not to start at the beginning of the trajectory, but at is home configuration for example, its initial error to the desired trajectory would be very large. Due to this large error, the joint controllers will want to send very high joint torques to the robot, however, these torques are also physically limited by the actuator capacity.
The torque limits of a Franka robot can be seen in Table B.1.
Very high initial errors will be hard to compensate for. This behaviour was clearly discovered by analyzing the influence of either starting at the home configuration or starting at the start of the trajectory. The results can be seen in Figure 4.4 and Figure 4.5.

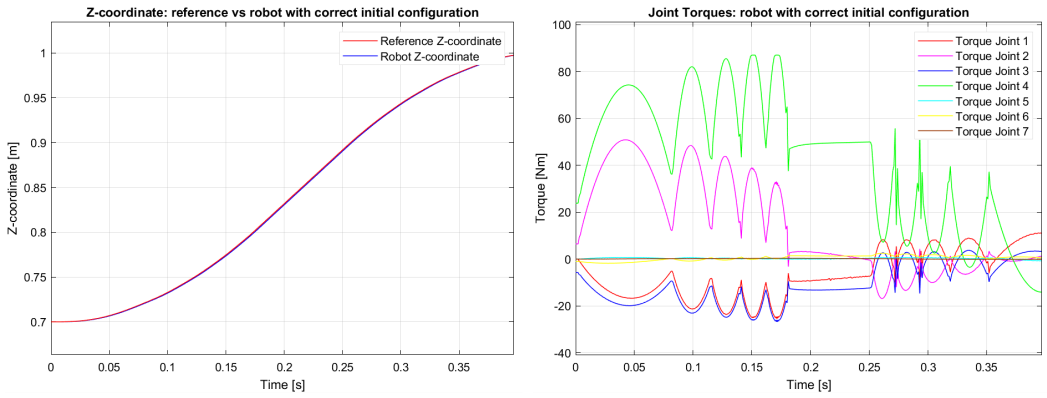


Figure 4.4: Robot starting at desired initial configuration

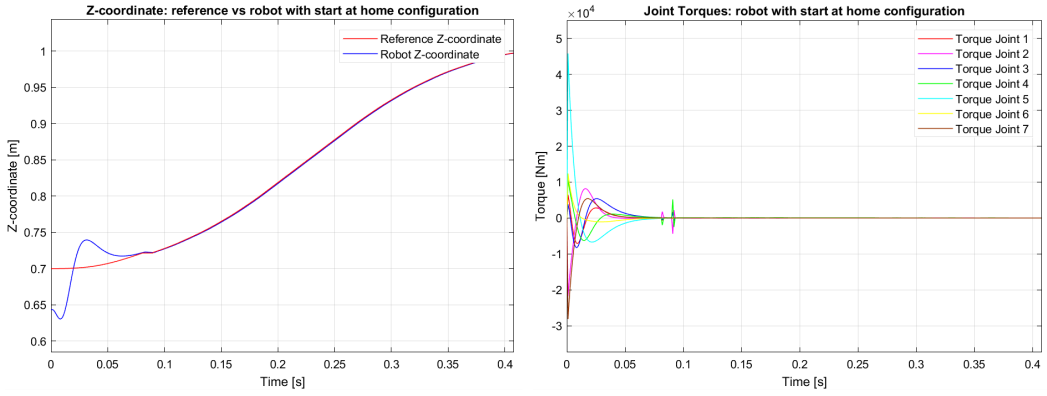


Figure 4.5: Robot starting at its home configuration

As you can see, if one was to start the trajectory whilst the robot was still in its home configuration, very high joint torques would be computed by the joint controllers. As such amounts of joint torques are not physically possible, it is important to carefully initialize the initial robot configuration, as is done in this initialization script.

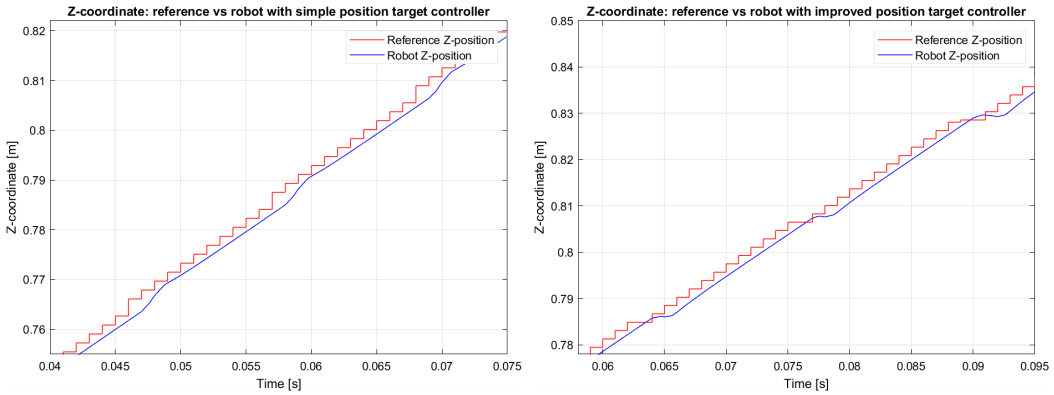
4.2 Main controller

The next section of the architecture, which underwent a systematical analysis, is the main controller. The main controller has many different functions, which will be separately analyzed in this subchapter.

4.2.1 Position target

One of the functions of the main controller is to determine the current robot position and orientation target. As previously explained, the desired trajectory is sampled into waypoints. These waypoints are the basis upon which the controller determines its position targets.
The goal of the main controller is to send a current position and a desired position to the waypoint-to-waypoint trajectory generator. So that this generator can generate a continuous, smooth, trajectory in between that.
The main controller determines the current and desired waypoint by constantly analyzing whether or not the robot is within 1 [mm] of its target waypoint. Once that is the case, the main controller will change its current and desired waypoint to the next and so on, until the trajectory has been completed.
This sounds quite straightforward. Yet, by analyzing the reference signal, it is clear this approach is not ideal. See Figure 4.6a

As is clear, whenever the main controller senses the robot to be in 'close-enough' proximity of the desired waypoint the next waypoint sequence is started. However, when this was sensed, the robot was not perfectly on the desired waypoint as the robot allows for an error of 1 [mm],
As a solution, the main controller was now designed to use the position of the reference signal at the moment of 'hitting' the desired waypoint as the new current waypoint. The result can be seen in Figure 4.6b.



(a) (b)

Figure 4.6: (a): Original simple position controller reference signal, clear bump upwards. (b): Improved position controller reference signal, no bump upwards, but a flat line is introduced.

As you can see, the reference signal does not spike upwards anymore. However, since the reference signal value was used, for the new current position, there was a time step where the reference signal would be flat. This was because, once the waypoint was 'hit', the new current position was told to be the exact position it was one time step earlier.

Now the spike upwards was solved, however, a sudden slowness in the reference signal was introduced. To solve this, once a waypoint was 'hit', the current position was again told to be that of the reference signal at the moment of 'hitting' the desired waypoint. However, this time, a small distance was added to it to overcome the sudden flat line.
In order for this extra distance to be correct, it should be based on multiple factors such as the sample time, trajectory and end effector velocity.
First, sample time should be taken into account, as when you would pick a larger sample time, the increase in reference signal which would be observed every time step would also increase. Next, the trajectory should also be taken because the amount of change in coordinates should be taken into account. For example a trajectory where the X-velocity is higher then the Y-velocity should also give the X-reference signal a higher bump.
And lastly, the end effector velocity, as this determines the overall speed at which the reference signal changes its position.
The complete equation which calculates the amount of bump a reference coordinate receives is seen in Equation 4.3

$$\begin{bmatrix} X^{bump} \\ Y^{bump} \\ Z^{bump} \end{bmatrix} = \frac{V_{EE}}{f} \cdot \left(\begin{bmatrix} X^{des} \\ Y^{des} \\ Z^{des} \end{bmatrix} - \begin{bmatrix} X^{curr} \\ Y^{curr} \\ Z^{curr} \end{bmatrix} \right) \div \left\| \begin{bmatrix} X^{des} \\ Y^{des} \\ Z^{des} \end{bmatrix} - \begin{bmatrix} X^{curr} \\ Y^{curr} \\ Z^{curr} \end{bmatrix} \right\| \tag{4.3}$$

Where,

- V_{EE} is the end effector velocity, which can be determined by the user.
- f is the frequency (sample rate).
- X, Y, Z^{bump} are the to-be-added values.
- X, Y, Z^{des} are the coordinates of the waypoint which is the desired goal.
- $X, Y, Z^{current}$ are the coordinates of the waypoint which is the current position.

Once this bump value was calculated, the previous flat line problem could be solved. The result of this bump can be seen in Figure 4.7.

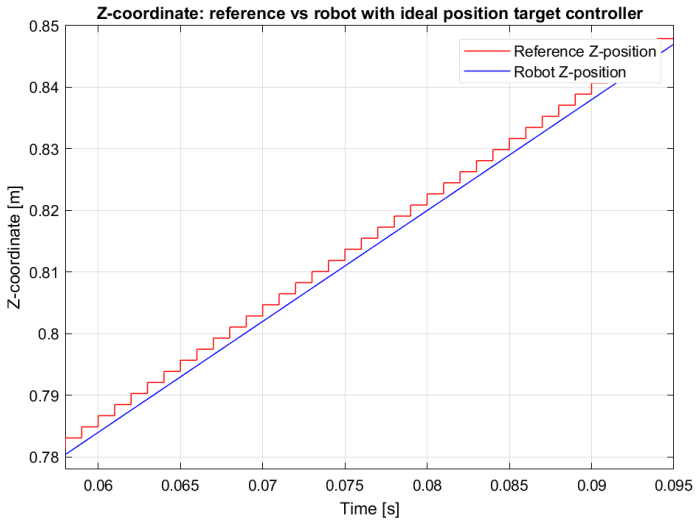


Figure 4.7: Ideal position controller reference signal where no effect around waypoints is present

As you can see, there is no effect around the hitting a waypoint anymore. From testing many different options and configurations, this position determination approach seems to be working flawlessly in every time.

4.2.2 Velocity target

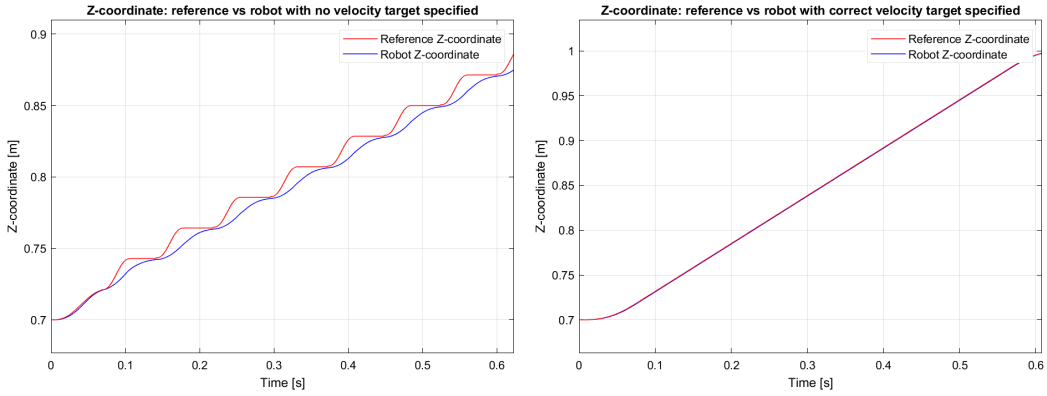
The next task of the main controller is to tell the waypoint-to-waypoint trajectory generator what the end effector velocity at the current waypoint and the desired waypoint will be. This end effector velocity has to be divided into its coordinate frame components, X, Y and Z. This division can be calculated using Equation 4.4.

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = V_{EE} \cdot \left(\begin{bmatrix} X^{des} \\ Y^{des} \\ Z^{des} \end{bmatrix} - \begin{bmatrix} X^{curr} \\ Y^{curr} \\ Z^{curr} \end{bmatrix} \right) \div \left\| \begin{bmatrix} X^{des} \\ Y^{des} \\ Z^{des} \end{bmatrix} - \begin{bmatrix} X^{curr} \\ Y^{curr} \\ Z^{curr} \end{bmatrix} \right\| \tag{4.4}$$

This solves the X,Y,Z, velocity components.
For the angular velocity components, the following formula is used, see Equation 4.5.

$$\begin{bmatrix} \omega_\alpha \\ \omega_\beta \\ \omega_\gamma \end{bmatrix} = \omega_{EE} \cdot \left(\begin{bmatrix} \theta_\alpha^{des} \\ \theta_\beta^{des} \\ \theta_\gamma^{des} \end{bmatrix} - \begin{bmatrix} \theta_\alpha^{curr} \\ \theta_\beta^{curr} \\ \theta_\gamma^{curr} \end{bmatrix} \right) \div \left\| \begin{bmatrix} \theta_\alpha^{des} \\ \theta_\beta^{des} \\ \theta_\gamma^{des} \end{bmatrix} - \begin{bmatrix} \theta_\alpha^{curr} \\ \theta_\beta^{curr} \\ \theta_\gamma^{curr} \end{bmatrix} \right\| \tag{4.5}$$

Using both these formulas, the velocities can be calculated once the desired- and current end effector position and orientation are known.
If this controller were to not take into account velocities, the reference signal, generated by the waypoint-to-waypoint trajectory generator, would function significantly worse, as can be seen in Figure 4.8.



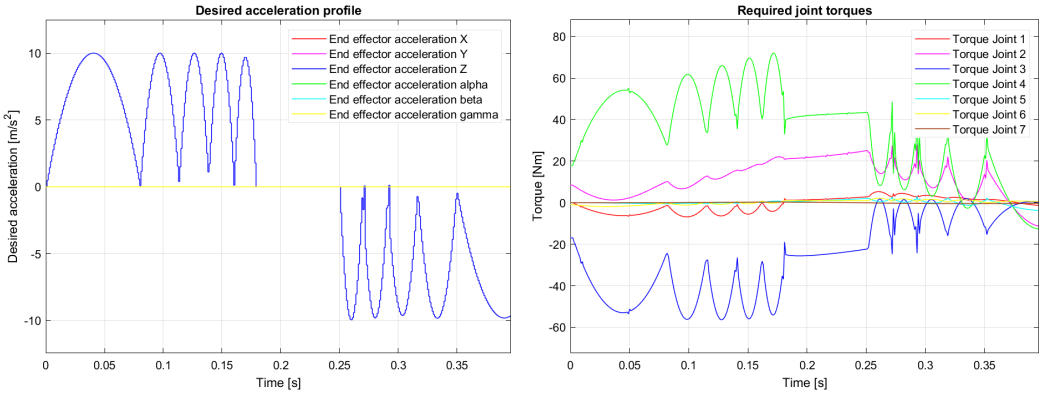
(a) (b)

Figure 4.8: (a): Generated reference signal with zero velocity at each waypoint.
(b): Generated reference signal with correct velocity at each waypoint.

Looking at these results it is clear that the main controller should be capable of computing the joint velocities based on the chosen end effector velocity. In order to define this end effector velocity, a user-input was created. This way the user can directly choose what end effector velocity should be.

4.2.3 Acceleration target

Apart from the position and velocity target, another important factor contributing to the performance of a robot is its acceleration. Acceleration is directly linked to forces or torques through Newton's second law. Therefore, being able to control or bound the trajectory to a certain acceleration maximum would be very helpful in avoiding joint torque limits [11].
The contribution of end effector acceleration to required joint torques can be clearly seen in Figure 4.9

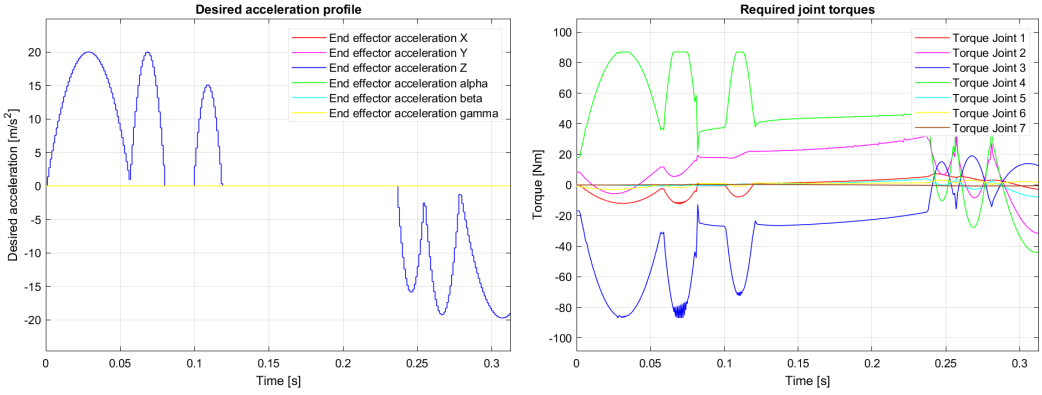


(a) (b)

Figure 4.9: (a): The acceleration profile of the desired trajectory.
(b): The required joint torques to fulfill this desired trajectory.

From this figure, the importance of bounding the acceleration of a trajectory becomes very clear. The method of limiting acceleration was chosen to be based on the waypoint evaluation method as well. As was already briefly noted previously, the main controller functions on a waypoint evaluation method. So every time a waypoint is met, it evaluates itself. Therefore a choice was made to have an acceleration of zero at every waypoint. As this allows the controller to rapidly change its decision making if required.

Bounding this acceleration has been proven to be very important to ensure good cooperation with joint torque limits. As you can see in Figure 4.9, the maximum acceleration is bounded to be an absolute value of $10m/s^2$.
From testing, this turned out to be about the maximum safe acceleration which does not exceed joint torque limits. Looking at Figure 4.10, a simulation was performed where the user chose the maximum acceleration to be $20m/s^2$.



(a) (b)

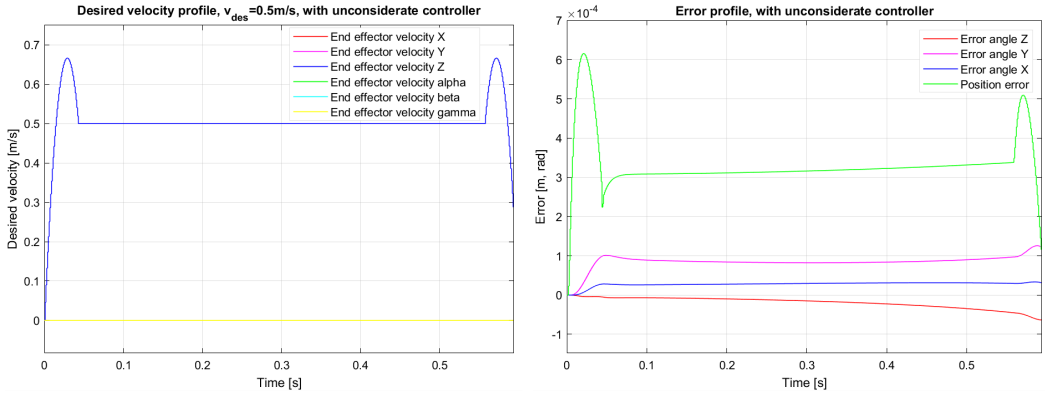
Figure 4.10: (a): The acceleration profile of the desired trajectory, with too high user-inputted acceleration limit.
(b): The required joint torques to fulfill this desired trajectory, clearly bounded by torque limits.

Clearly the torques on joint four are limited by the actuator's limits.
This is far from ideal in terms of accuracy as the robot cannot physically keep up with the reference signal. On top of that, the repeatability of such a trajectory is also greatly reduced. This is because the predictability of a robot, which operates at its torque limits, is not great.

From this acceleration to joint torque analysis, it has become clear that a strong relation between end effector acceleration and joint torques exists. Therefore, letting the user be able to influence this acceleration, can directly ensure that the desired trajectory complies with the physical limits of a robot. For this reason an acceleration boundary user-input was integrated into the main controller.

4.2.4 Start-to-end trajectory

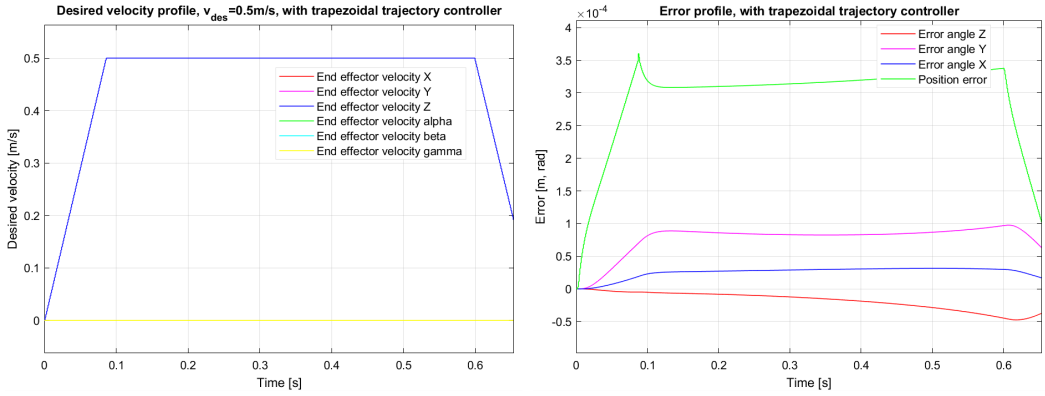
In the subchapter about velocity targets, we have seen that having correct velocities at each waypoint is essential for a smooth trajectory. However, it is also easy to understand that this velocity should be zero at the beginning and end of the trajectory. So the complete trajectory should start at zero, then it should increase to some value, after which it finally decreases to zero again.
Before taking into account this fact into the controller, the velocity profile did not fully comply with the chosen set end effector velocity target. Which in itself is a problem since the user should be able to control this by choice. However, this also resulted in increased tracking errors.
See Figure 4.11, where a desired end effector velocity of 0.5m/s was chosen.



(a) (b)

Figure 4.11: (a): The velocity profile of the desired trajectory, which clearly overshoots the desired velocity.
(b): The error signal, which displays clear peaks when the velocity peaks as well.

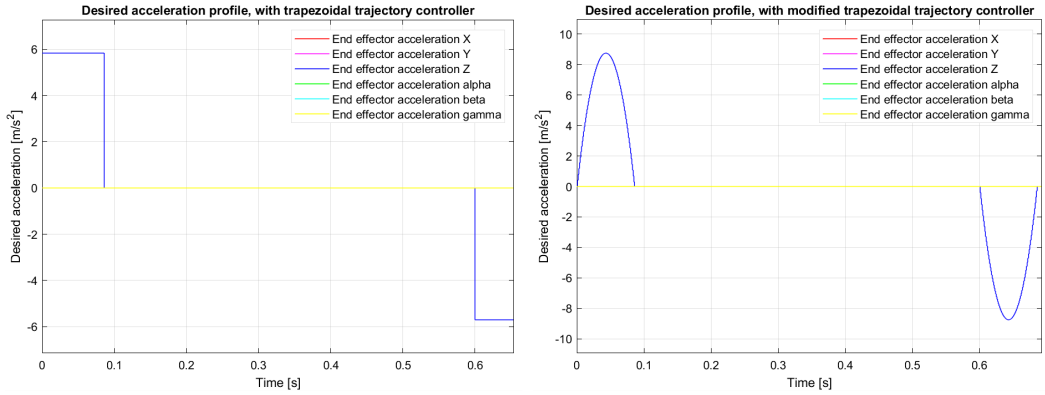
As you can see, because no start-to-end trajectory profile was designed in the main controller yet, the velocity has to shortly increase above its desired value. This is because the main controller does not yet take into account the fact that the first and last waypoints require zero velocity. Because of this, the trajectory generator has to over-compensate on velocity.
A very common trajectory profile which meets the start-to-end velocity requirements (of zero-constant-zero velocity), is the trapezoidal profile [14].
Incorporating this profile into the start-to-end trajectory gave the results seen in Figure 4.12.



(a) (b)

Figure 4.12: (a): The velocity profile of the desired trajectory, which exactly follows a trapezoidal profile .
(b): The error signal, which displays smaller peaks, but still peaks.

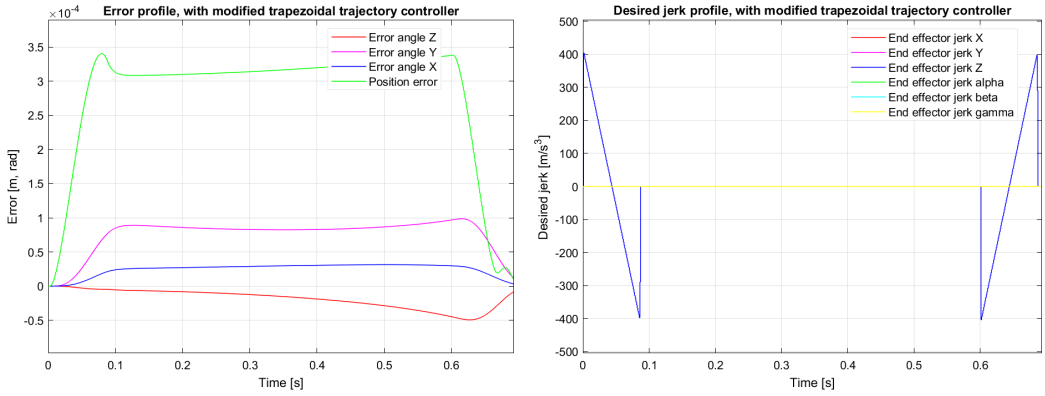
As you can see, the trapezoidal profile had been incorporated. Which had made the error peak smaller. However, this error peak still persisted. This is due to the fact that in a trapezoidal profile, the acceleration and jerk are discontinuous signals. As we know from the literature [15], [16], to ensure smooth trajectories, position, velocity, acceleration and jerk should all be continuous signals. So the next step would be to make acceleration and jerk continuous. By modifying the original trapezoidal profile, the acceleration could be made continuous as well.
Figure 4.13, shows the acceleration profile difference which was created by modifying the trapezoidal profile.



(a) (b)

Figure 4.13: (a): The acceleration profile associated with the standard trapezoidal trajectory.
(b): The acceleration profile associated with the modified trapezoidal trajectory.

This continuous acceleration profile had a clear positive effect on the tracking error. The peak in error which was previously observed in Figure 4.12, had now been decreased by about 50%. This final error result can be seen in Figure 4.14.



(a) (b)

Figure 4.14: (a): The error signal associated with the modified trapezoidal trajectory, which clearly has a smaller peak.
(b): The jerk profile associated with the modified trapezoidal trajectory, which is discontinuous.

As you can see, the peak in error has been greatly reduced. However, it is still present to a small extent. This is because the jerk is still discontinuous as is also visible in Figure 4.14. Unfortunately, with the built-in trajectory generators used in Simulink, jerk is not controllable. For this reason, jerk is still discontinuous, which is the most likely cause of this small peak in error.

4.2.5 Error calculation

In this report, the error, or tracking error, has been mentioned plenty of times. However, it is important to define what sort of error calculation is used. This definition is particularly important for this particular research, since a leader-follower robot combination is used.
In most cases the position error is defined as the difference between the desired position and the actual robot position.
However, the definition of the problem statement had stated that orientation should also be taken into account.
And what about the fact that two robots are used?
In order to take all of these factors into account it is important to clarify what sort of operation is ongoing. If the transformation between leader- and follower end effector must be controlled very precisely a very detailed error calculation should be used. But if both robots are just moving independently to some goal, their errors could also be evaluated independently.
So first of all, if the movements are independent, the error which the controller calculates is just the position error of the robot's end effector itself. This is the simple, independent case.
If, however, the robots are moving collaboratively, the error calculation has to take into account both robots' position errors and their orientation errors.
See Figure 4.15, for a visual representation of these errors.

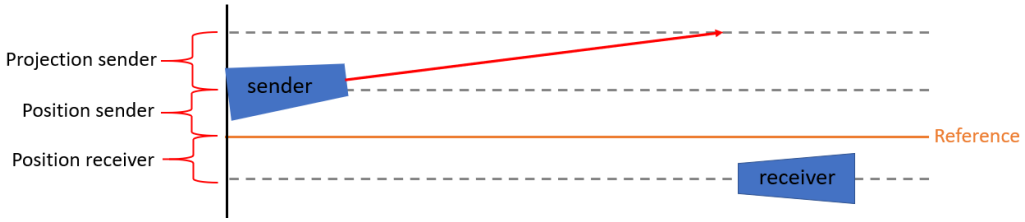


Figure 4.15: The visual representation of all errors combined.

The total error which the controller should use in this situation is defined as follows.

$$e_{total} = e_{sender} + e_{receiver} \tag{4.6}$$

Where

$$e_{sender} = e_{position} + e_{projection} \tag{4.7}$$

And

$$e_{receiver} = e_{position} \tag{4.8}$$

In these equations it is assumed that some type of signal is to be send from one end effector to the other, a use case scenario of this could be an x-ray scan. Therefore, the total error consists of the error associated with the sender and the error associated with the receiver.
The receiver error is quite straight forward as it only contains its position error.
The sender error, however, consists of its position error, but also of its projection error. The projection error denotes the error which results from its orientation error which is amplified over a certain distance. In this case, this distance is equal to the distance between the end effector of the leader and follower robot.
This projection error is defined using the following equation.

$$e_{projection} = d_x(|e_\alpha| + |e_\beta|) + d_y(|e_\alpha| + |e_\gamma|) + d_z(|e_\beta| + |e_\gamma|) \tag{4.9}$$

Where

- d_x,d_y,d_z, is the distance between end effectors in x,y and z coordinates, respectively. Measured using the end effector coordinate frame.
- e_α,e_β,e_γ, is the angle error around axis z,y,x, respectively. Measured using the end effector coordinate frame.

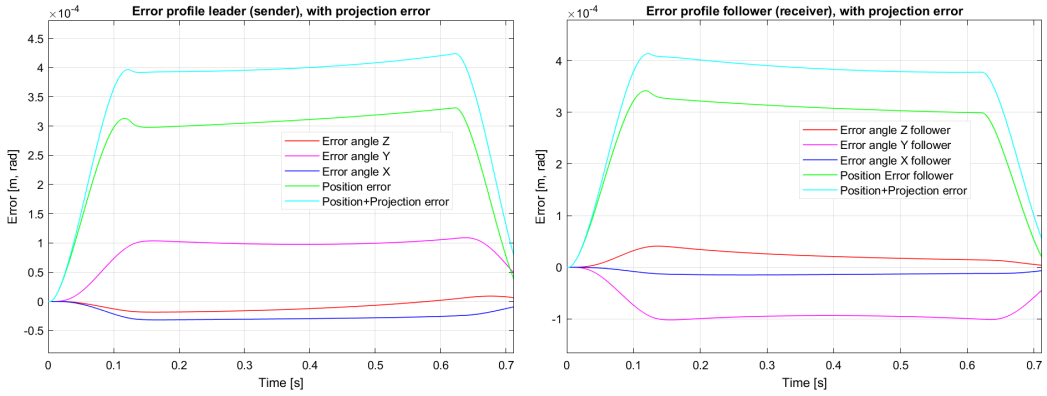
With the use of these equations, the controller is able to take into account both position and orientation errors of the sender plus the position error of receiver to come to a complete error calculation which is able to exactly relate with how much error a signal could be transferred between the two robots.
The independent values of orientation error and position error, are calculated by comparing the orientation and position of the robot to the orientation and position of the reference signal. For example:

$$e_\alpha = ref_\alpha - robot_\alpha \tag{4.10}$$

Where:

- e_α is the orientation error around axis Z of the end effector's coordinate frame.
- ref_α is the reference signal angle around axis Z of the end effector's coordinate frame.
- rob_α is the robot orientation angle around axis Z of the end effector's coordinate frame.

A simulation with these new error calculations was done. This clearly shows the difference between the plain, position error, and the more detailed position+projection error. See Figure 4.17



(a) (b)

Figure 4.16: (a): The error graph of the leader robot, which in this case was chosen to be the signal sender.
(b): The error graph of the follower robot, which in this case was chosen to be the signal receiver.

As you can see, for both robots the position+projection error is calculated. However, the error which the controller uses to evaluate its performance must only be based on the position+projection error of the sender plus the plain position error of the receiver.
To allow for both the leader or the follower to be a sender or receiver, a user-input was created which lets the user decide which of the robots is the sender.
In the simulated example, illustrated in Figure 4.17, the leader was chosen to be sender. With this knowledge, the controller is able to calculate the complete scanning error which it uses to evaluate its performance. This complete scanning error for this case can be seen in Figure 4.17.

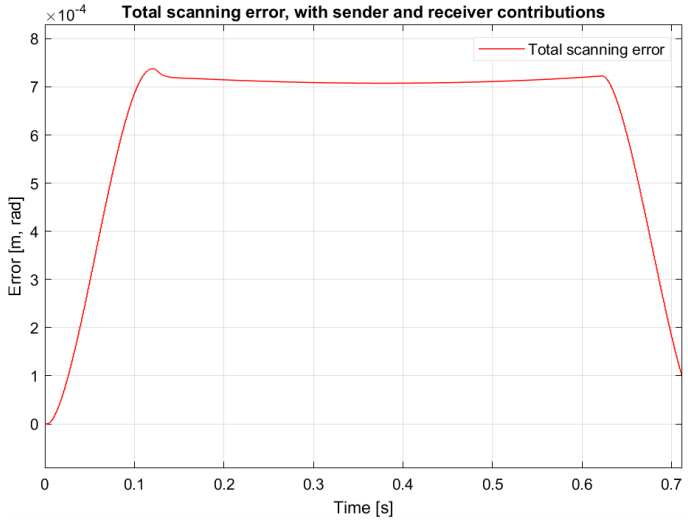
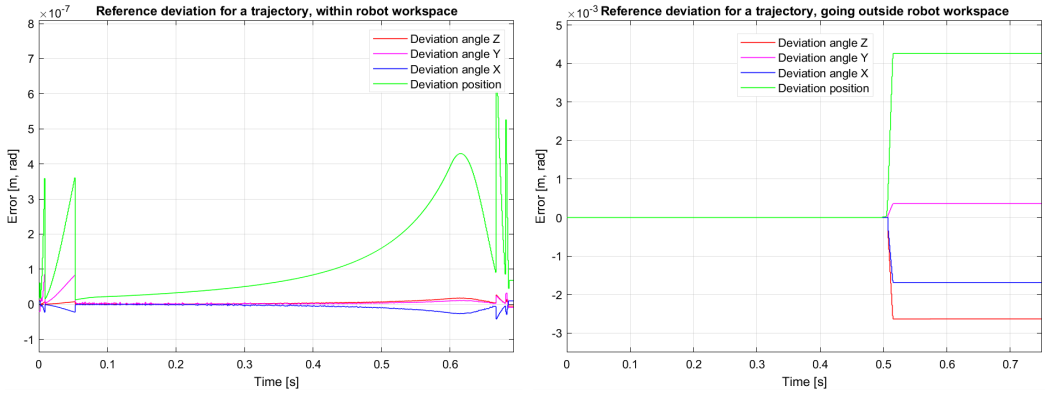


Figure 4.17: Complete scanning error, where the leader is sending the signal and the follower is receiving it.

4.2.6 Reference deviation

A physical limit placed on every robot trajectory is the robot's workspace. As explained in the architecture chapter, one of the important parts of the control loop is the inverse kinematics. If one was to choose a trajectory which the robot is physically not able to reach, the solution to the inverse kinematics could never be fully correct. This incorrect inverse kinematics solution would then result in a incorrect end effector position. In some cases, it would be desirable for the robot to try to get as close as possible, but in other cases it would not be considered a good idea if the robot went off-trajectory.
An algorithm should be added to the controller which detects such an end effector deviation. Once this would be detected, it would either allow it and move on, or disallow it and stop the robot.
To detect such a deviation from the reference signal, the desired end effector pose (which is the inverse kinematics input) should be directly compared to the end effector pose which would result from the joint angles computed by the inverse kinematics.
Doing this for an arbitrary trajectory which fully operates within the robot workspace the deviation from the reference signal looks like Figure 4.18a.
As you can see, the deviation varies a bit but always stays very small ($< 1\mu m$).
However, if we task the robot to follow a trajectory which is outside of its workspace, seen in Figure 4.18b, this reference deviation increases rapidly. From this signal, it is detected by the controller that the current reference signal cannot be solved by the inverse kinematics solver.



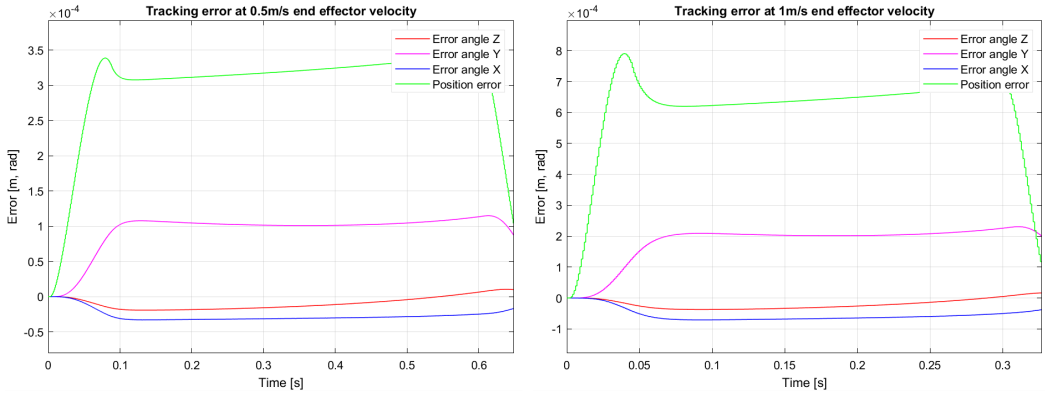
(a) (b)

Figure 4.18: (a): The deviation from the reference signal calculated for a trajectory within the robot workspace.
(b): The deviation from the reference signal calculated for a trajectory going outside the robot workspace.

With this inverse kinematics error calculation, not only can deviations be detected, but the control engineer is also able to ensure that relatively little reference deviation exists when working within the robot workspace. Since, as we saw in Figure 4.18a, the error associated with the inverse kinematics solution, is at least a thousand times smaller then the 1 [mm] error goal. This knowledge gives the control engineer more insight into the quantity of some of the error sources in a system, which is very important.

4.2.7 Time scaling

In the subchapter about the velocity target, it was mentioned that the user is now able to directly choose the end effector velocity. However, in some cases, the user either does not want a constant velocity or does not know a realistic value.
In such a case it would be very beneficial if the controller would be able to choose an optimal velocity himself [8], [9], [10]. For that reason, the trajectory time scaling algorithm was designed.
In essence, the time scaling algorithm is able to maximize performance for every trajectory whilst ensuring a desired accuracy.
To investigate this relation between performance and accuracy, trajectories were simulated at different velocities. Then by comparing their error signals a relation could be found.
In Figure 4.19, such a comparison between 0.8m/s and 0.4m/s is shown.



(a) (b)

Figure 4.19: (a): Tracking error for an end effector velocity of 0.5m/s.
(b): Tracking error for an end effector velocity of 1m/s.

From such tests, it became clear there was a direct relation between velocity and error. On top of that, this relation actually seemed to be linear as well. With the knowledge of this (linear) relationship a time scaling algorithm could be constructed.
This algorithm was based on Equation 4.11.

$$v_{des} = v_{curr} \frac{e_{des}}{e_{curr}} \tag{4.11}$$

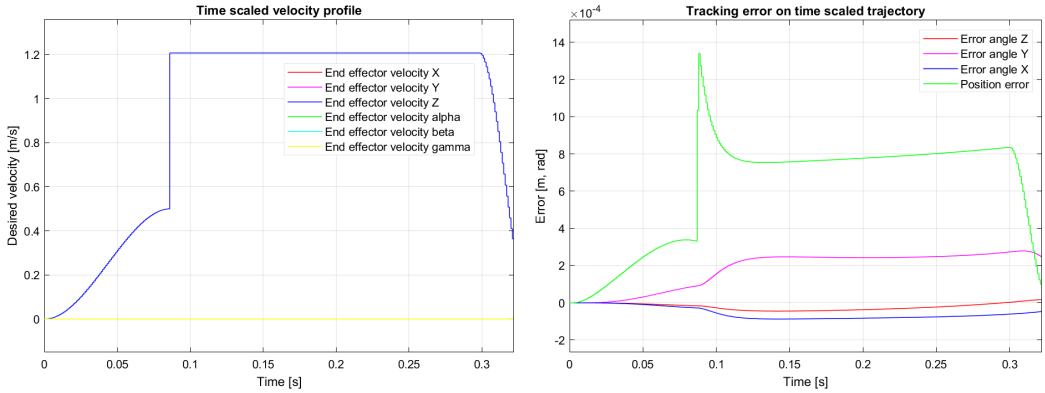
In theory, this formula, if integrated correctly into the controller, should ensure that the velocity is changed so that the actual error matches the desired error.
This desired error could be easily chosen by the user. It is adviced to choose some value slightly below 1[mm], like 0.8[mm]. This small difference is to ensure that the error always stays below 1[mm].

It is important to clarify that in previous chapter a lot of controller choices have been based on satisfying the joint torque limits. In the initial stages of these time scaling algorithms this was not yet taken into account, but this will be solved later on in this subchapter.

This initial method of time scaling was integrated into the controller. The controller could scale the velocity to match the desired error on a few conditions.

- It had not changed end effector velocity in the last 0.05 secs.
- The trajectory is not accelerating or decelerating for the first and last waypoint.
- The error was at least 10% larger or smaller then desired.

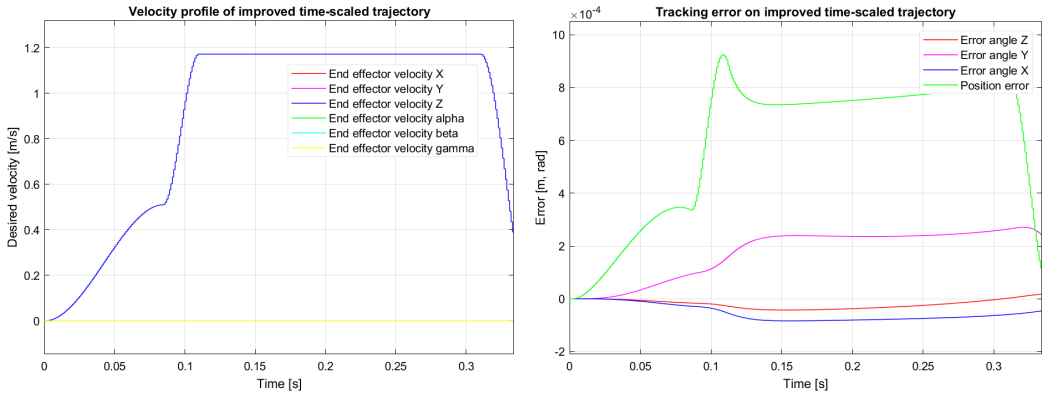
With the implementation of this initial time scaling algorithm an error and velocity profile of a system which used this time scaling method looked as follows. See Figure 4.20



(a) (b)

Figure 4.20: (a): Velocity profile of a time-scaled trajectory.
(b): Tracking error of a time-scaled trajectory.

In this simulation an initial velocity of 0.5m/s was chosen. As you can see, as soon as trajectory is done accelerating to 0.5m/s, the controller can start optimizing velocity to match the desired error. This means, it immediately increases the velocity to 1.2m/s. This increases the error rapidly as well, after which it settles down to around 0.8[mm]. 0.8[mm] is the desired error so it exactly did what it was tasked to do. However, the one problem with this solution is the immediate increase in velocity. This would require infinite acceleration which is impossible, so the robot would never be able to fully keep up. This problem can also be seen in the error graph as there is a very large peak associated with this sudden velocity increase.
A solution to this would be to give the trajectory some time to increase towards the new time-scaled velocity.
Looking at Figure 4.21, this more reasonable time-scaling method can be seen.

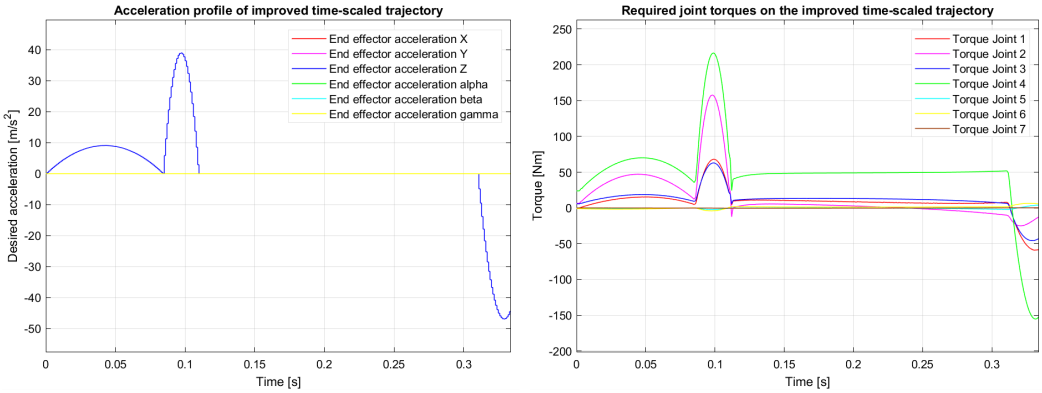


(a) (b)

Figure 4.21: (a): Velocity profile of a time-scaled trajectory, which does not require infinite acceleration.
(b): Tracking error of a time-scaled trajectory, with a much smaller error peak.

With this new method, the error had been greatly reduced. However, as was previously mentioned. Joint torque limits had not been taken into account yet. However, when one is to take them into account, this current method does not completely work either.

This is because even though the required acceleration is not infinite anymore it is still relatively large. See Figure 4.22.

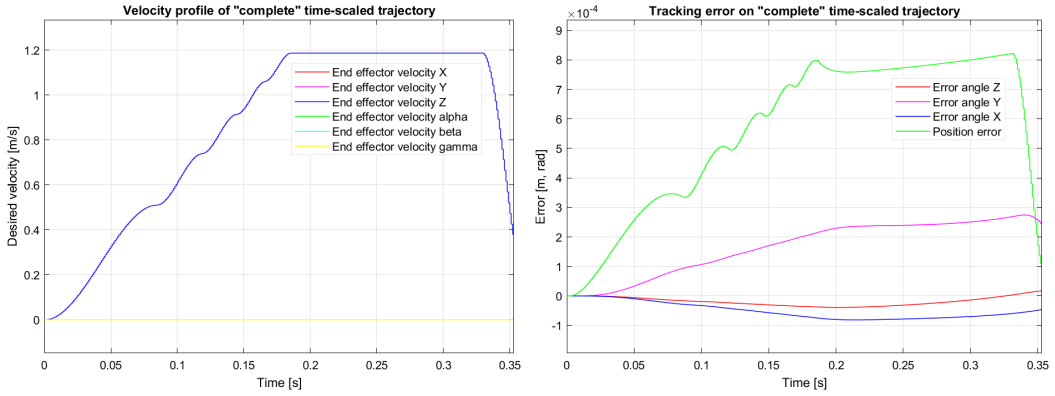


(a) (b)

Figure 4.22: (a): acceleration profile of a time-scaled trajectory, with a required acceleration of around $40m/s^2$.
(b): Required joint torques, which greatly exceed the physical joint torque limits.

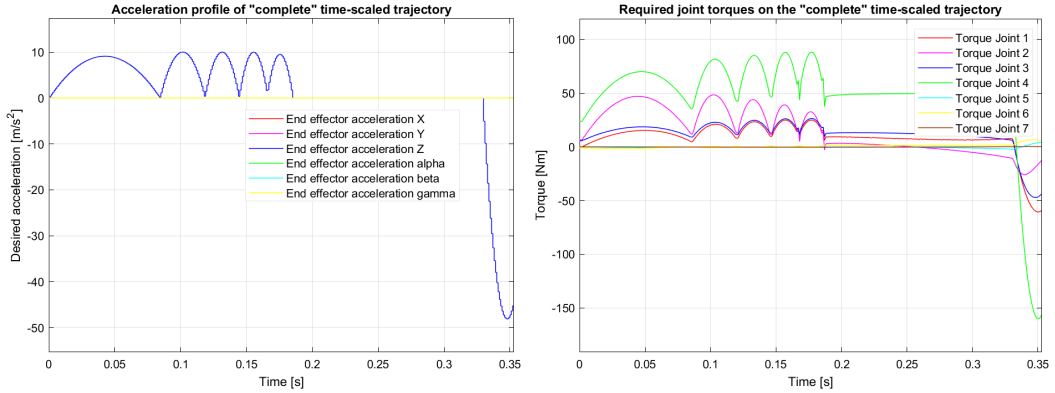
As you can see, the acceleration profile requires the robot to accelerate at around $40m/s^2$, which, by looking at the torque limits, is known to be too fast.
In the acceleration target chapter we had already proposed the user-input of a maximum acceleration. So it would be very important to also have the time-scaling algorithm satisfy this acceleration limit [11].
The method of satisfying this is by keeping the controller be able to increase the velocity. However, this velocity increase should always be limited by the maximum user-inputted acceleration.

This new method makes the algorithm increase to the desired velocity in a more step-wise manner as in most cases it is not able to completely increase to the desired velocity. This makes the velocity optimization process a little less optimal as it reaches this velocity a bit slower. However, it is able to satisfy the imposed acceleration bounds.
The result of this method can be seen in Figure 4.23. Where the maximum acceleration was chosen to be $10m/s^2$.
In Figure 4.24, it can be seen that the acceleration required for this time-scaling method does not exceed this user-inputted boundary, which helps the joint torques stays within its physical limits.



(a) (b)

Figure 4.23: (a): Velocity profile of a time-scaled trajectory, which complies with user-imposed acceleration boundaries.
(b): Tracking error of a time-scaled trajectory, which complies with user-imposed acceleration boundaries.



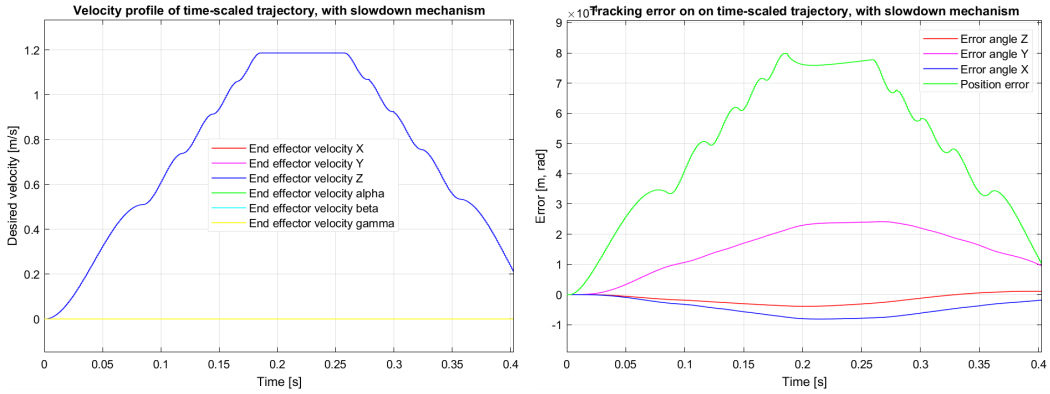
(a) (b)

Figure 4.24: (a): Acceleration profile of a time-scaled trajectory, which complies with user-imposed acceleration boundaries of $10m/s^2$.
(b): Required joint torques, which do not exceed its joint torque limits (apart from the final deceleration, this will be discussed next).

4.2.8 Slow down algorithm

In the time-scaling chapter, the acceleration and deceleration towards the optimal velocity had been solved. It had also been solved whilst satisfying the acceleration boundaries. However, now that the controller was able to speed the robot up to such high velocities, a new problem had arised. This problem was that with such a high velocity, the last waypoint-to-waypoint sequence was not enough to completely slow down the robot anymore. This is also clearly visible in Figure 4.24. As you can see, the at the end of the trajectory the deceleration required is around $-50m/s^2$. Therefore, some forward looking mechanism should be installed which looks ahead to see whenever it is time to start decelerating towards an ending velocity of 0m/s.
From the results of the time-scaling algorithm, it was discovered that this step wise method of changing velocity was very good at complying with acceleration constraints.
For this reason a similar step wise system was put into place which starts decelerating the robot as soon as it is required.

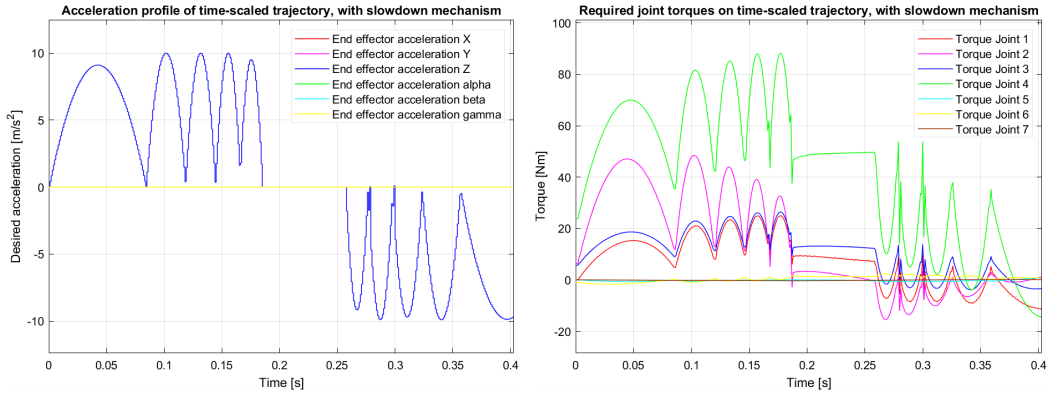
The result of this step wise, slowdown method, on the velocity and error can be seen in Figure 4.25.



(a) (b)

Figure 4.25: (a): Velocity profile of a time-scaled trajectory, with the step wise slowdown method incorporated.
(b): Tracking error of a time-scaled trajectory, with the step wise slowdown method incorporated.

As you can see, the time-scaled optimal velocity is still reached. However, it is used a little shorter as the trajectory is told to decelerate a bit earlier in order to satisfy the acceleration constraint of $10m/s^2$ (which also translates to a maximum deceleration of $-10m/s^2$).
Looking at Figure 4.26, it is also clear that the acceleration profile exactly stays within the desired $|10m/s^2|$.
This in turn helps the joint torques stay within their limits [11].



(a) (b)

Figure 4.26: (a): Acceleration profile of a time-scaled trajectory, with the step wise slowdown method incorporated.
(b): Required joint torques, which do not exceed its joint torque limits.

4.2.9 Leader-follower algorithm

The last aspect of the main controller is the leader-follower algorithm.
In the problem analysis it had been mentioned that for this specific healthcare case, multiple different use cases for the robots would be imaginable. Different use cases might also require different 'constraints' between the leader and follower movement.
Almost all use cases imaginable require a leader-follower algorithm which falls into either one of the following categories.

- Conventional leader-follower algorithm
- Leaderless algorithm
- Perfect transformation algorithm

Conventional leader-follower algorithm.

This algorithm consists of a concrete leader robot and a concrete follower robot. The leader robot will always be either at the same progression through its trajectory compared to the follower [20], [36]. Or it will be ahead in progression through its trajectory compared to the follower.
In principle such an algorithm would be used if the positioning of both robots is independently important, but their transformation between them is not. Such an algorithm is especially interesting in a autonomous navigation use case, where the leader first has to 'discover' its trajectory. This makes it that the follower could never advance as it is dependent on the trajectory of the leader, which it still has to compute [37].

To realize this conventional leader-follower algorithm, the main controller sets the following constraints to its control strategy:

- Follower progression $\leq$ leader progression.
- Follower end effector velocity $\leq$ leader end effector velocity.
- Follower end effector acceleration $\leq$ leader end effector acceleration.
- Error leader = position error leader.
- Error follower = position error follower.

Leaderless algorithm.

This leader-follower algorithm is based on the fact that either one of the robots can be a leader or a follower. This means that it does not matter, which robot is advancing through its trajectory faster. Both robots can just move as fast as they independently want to.
Such an algorithm is again useful if only the independent positioning is important but not necessarily their transformation between each other.
Such a trajectory could also be an autonomous navigation system, but it would require both robots to be able to autonomously discover its trajectory separately. Another, scenario could be when beforehand the complete trajectory would be known. In this case the trajectory of both robots could be computed beforehand and both robots could follow its own trajectory without worrying about each other.

To realize this, the main controller has to put no constraints on the progression, velocity or acceleration as both robots operate independently.

The perfect transformation algorithm.

In this algorithm, not only is the independent robot position important, but also the transformation between end effector is important [18], [19].
This means that they should always be at the exact same progression through their trajectory.
A use case for such an algorithm could be an x-ray scan for example.
To realize this perfect transformation algorithm, the main controller sets the following constraints to its control strategy:

- Follower progression = Leader progression (and vice versa)
- Follower end effector velocity = leader end effector velocity (and vice versa).
- Follower end effector acceleration = leader end effector acceleration (and vice versa).
- Error leader (scan sender) = position+projection error leader.
- Error follower (scan receiver) = position error follower.
- Total error = error leader (scan sender) + error follower (scan receiver).

With these three algorithms almost every use case can be simulated with an appropriate leader-follower relationship.
The imposed constraints realize a smooth trajectory for both robots for all algorithms.

4.3 Waypoint-to-waypoint trajectory generator

The next part of the control loop proposed in Figure 3.1, is the waypoint-to-waypoint trajectory generator. As explained, the goal of this part is to generate a smooth trajectory based on the desired conditions calculated by the main controller.
As this control architecture is structured within Matlab and Simulink, there are a two different profile theories which can be used. Either a trapezoidal profile or a polynomial profile.
The polynomial profile can in turn then be divided into a B-spline, Cubic- or Quintic polynomial.

4.3.1 Trapezoidal

First of all, the trapezoidal trajectory.
In the start-to-end trajectory part of this report, the trapezoidal was already introduced. It was explained that this trapezoidal trajectory was used (and modified) because it matched the desired goal.
In the start-to-end trajectory, the starting and end velocity were zero, which is also the case for a trapezoidal profile [14].
However, this zero velocity is not the case for the waypoint-to-waypoint sequence, as the intermediate waypoints in a trajectory must have a velocity.
The effect of having zero velocity at each waypoint has already been shown in Figure 4.8.
In this figure it is clear that a trajectory profile such as the trapezoidal profile is far from optimal for waypoint-to-waypoint trajectories.
For this reason, the trapezoidal trajectory can immediately be disregarded.

4.3.2 B-spline

In a B-spline trajectory a trajectory is generated which vaguely follows some of the intermediate waypoints, but is not constrained to reach them, see Figure A.4. The trajectory is only constrained to follow the initial and last waypoint at a specified time.
This is for a few reasons not ideal for a waypoint-to-waypoint trajectory because as mentioned, the controller wants to satisfy a maximum error of 1[mm], across the complete trajectory. So if the waypoint-to-waypoint trajectory is able to generate a trajectory which is not constraint to follow all waypoints it cannot be ensured that the generated trajectory is the desired trajectory.
A solution to this could be to make more waypoints. In that case the intermediate waypoints, which are not constrained to be hit, are extra waypoints which are not too important. However, then the problem still remains that the 'original', important, waypoints have become the initial and final waypoint for that sequence. This is problematic as the B-spline function does not allow the controller to specify initial and final boundary conditions, such as velocity or acceleration.
Another method this issue could be solved is by customizing the B-spline trajectory, which is a common method chosen in literature [38]. However, seeing this is not a standard option in Simulink this was avoided.

In summary, Simulink's B-spline option generates an unwanted simplification of the trajectory and does not allow to influence velocity or acceleration directly. This makes influencing the generated trajectory very difficult.

4.3.3 Cubic polynomial

A cubic polynomial, or third order polynomial is a trajectory which in fact does pass through the desired waypoints. On top of that, one is also able to define velocity conditions for these waypoints. A cubic polynomial seems to offer the possibilities which were lacking in the previous two trajectory theories.
However, this trajectory profile does not constrain acceleration or jerk to be continuous [39]. Having the acceleration and jerk not be continuous will have negative effects on the smoothness of the end effector movement [15], [16]. This lack of smoothness will in turn cause greater tracking errors.

The influence of this discontinuous acceleration behaviour on the tracking error was already visualized in Figure 4.12.
For these reasons, the cubic polynomial is a good option as it allows for continuous position and velocity profiles, which can be directly influenced by the controller. However, the missing continuous acceleration and jerk are not optimal.

4.3.4 Quintic polynomial

The last standard Simulink trajectory is the Quintic polynomial, or fifth order polynomial.
This trajectory profile does allow direct influence on the acceleration. This also solves the discontinuity problem of the acceleration. Unfortunately, the jerk is still not constrained to be continuous for this trajectory.
This discontinuity of jerk has a negative effect on the smoothness of the end effector movement. The effect of the discontinuity was already illustrated in Figure 4.14.

From the standard trajectories provided by Matlab/Simulink, this quintic polynomial functioned the greatest as it allowed for the most amount of parameters to be influenced and constrained all but jerk to be continuous.
Finding a trajectory which is also constrains jerk to be continuous might be solvable using an even higher polynomial function [40]. However, this becomes a very complex mathematical problem as one would have to write their own custom script for this.

It was decided to use the quintic polynomial as it allowed for greatest amount of end effector trajectory smoothness, whilst still being relatively easy to use as it can be integrated using a standard Simulink block.
Using a polynomial trajectory in between waypoints is also the expected result when looking at the current literature [41], [40], [39].

4.4 Inverse Kinematics

With the newly generated trajectory, these end effector positions must somehow be translated into joint positions and velocities. This is the task of the inverse kinematics.
This inverse kinematics will be evaluated on a position and velocity level.

4.4.1 Inverse kinematics position level

The inverse kinematics calculation of a manipulator robot is quite complicated. However, Simulink supplies the user with a set inverse kinematics block.
The generated inverse kinematics solution is based on the desired end effector pose, the weights, and the initial guess.
The end effector pose is quite straightforward as this is the desired end effector pose.
The weights refer to the relative amount of precision required for a certain coordinate or angle. In this case, all coordinates and angles are equally important so they all receive the same weight.
Lastly, the initial guess. The inverse kinematics solver uses the initial guess to start looking for solution from this initial guess. In the system the input into this initial guess is the previous inverse kinematics solution. This is ideal as the robot can then start looking for inverse kinematics solution which are very similar on the position the robot is already at. This helps the end effector movement stay smooth as no sudden jumps in joint position are required.

To illustrate the importance of using the previous inverse kinematics solution as the current initial guess it is important to understand that many different inverse kinematics solutions are possible.
In Figure A.5, multiple different inverse kinematics solution are given for the same end effector pose. This illustration shows how much a inverse kinematics solution can deviate depending on the initial guess.
For this reason, the proposed initial guess method is crucial to satisfy smooth end effector movement.

4.4.2 Inverse kinematics velocity level

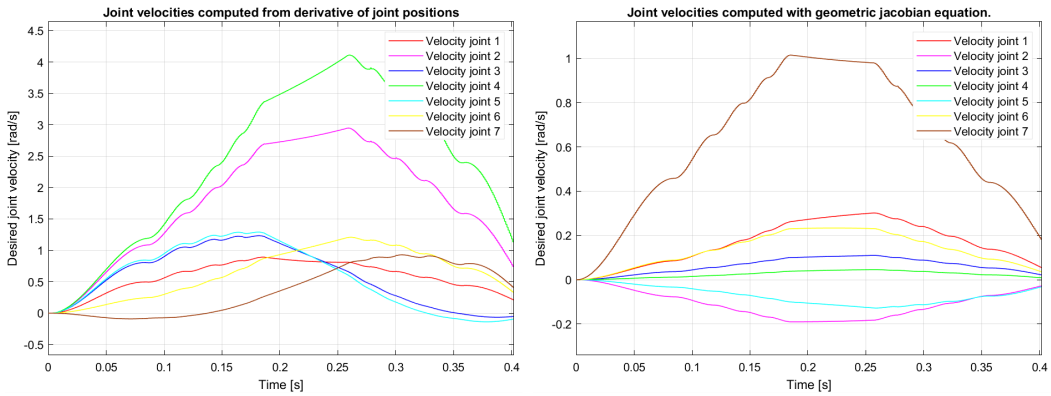
A common way of converting end effector velocities into joint velocities is with the use of a geometric jacobian (**J**) [42], [11]. By retrieving this jacobian for the current robot configuration it can be used in Equation 4.12. Using this equation, the joint velocities which satisfy the desired end effector can be computed.[31]

$$\dot{q} = J^{+}\dot{X} \tag{4.12}$$

Where:

- $\dot{q}$ is the vector of joint velocities.
- J^{+} is the pseudo-inverse of the geometric jacobian.
- $\dot{X}$ is the desired end effector velocity.

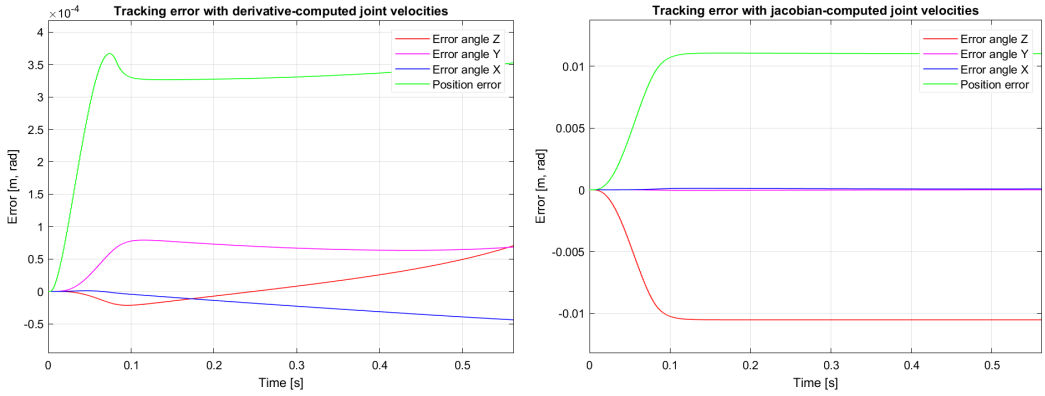
The jacobian of the current configuration can easily be computed by a standard Simulink block. For that reason this approach is quite straightforward.
However, a model which had implemented this joint velocity computation as an input into the joint controllers did not function well. The computation of the desired joint velocities did not seem to match up with the desired joint angles.
To verify this discrepancy, a test was performed. In this test the derivative of the inverse kinematics computed joint positions was compared to the joint velocities computed using Equation 4.12. The results can be seen in Figure 4.27



(a) (b)

Figure 4.27: (a): The joint velocities computed by taking the derivative of the joint positions. (b): The joint velocities computed with the jacobian in Equation 4.12.

By comparing these results it is very apparent that either solution computes very different joint velocities.
To verify which solution is correct a test was performed, where the 1[mm] error threshold constraint was relaxed and the joint torque limits were removed. These were removed to be able to properly simulate an incorrect system.
In this test, either the derivative method or the jacobian method would be used to compute the joint velocities. These computed joint velocities would be used in the joint controller.
In Figure 4.28, the error results of both options are displayed.



(a) (b)

Figure 4.28: (a): The tracking error if the joint velocities were computed by taking the derivative of the joint positions.
(b): The tracking error if the joint velocities were computed with the jacobian in Equation 4.12.

As you can see, when using the jacobian-computed joint velocities the error is approximately 35 times larger. This is a clear indication that using the jacobian to control velocity is not a good option in this case.

It is quite unexpected that the jacobian method functioned so poorly as it is a well established method of velocity control in the current literature [42], [11].
A reason for this could be a poor implementation into the control architecture.
As we will see in the next chapter, the joint controllers are based on a PD computed torque controller. This type of controller takes the joint position, joint velocity and joint acceleration into account. It could be possible that such a controller is not capable of using a joint velocity which does not match the derivative of the joint position.
However, this seems unlikely to be the source of the issue as it does not take into account the actual discrepancy which exists between the derivative-computed velocities and the jacobian-computed velocities. Intuitively it would be logical that these computation methods should grant the same solution. Which is not the case.
Therefore, it is likely that something in the computation of the jacobian-computed velocity is going wrong.
This could either be the jacobian computation itself. It could be the pseudo-inverse computation of the jacobian. Or it could be the joint velocity computation as seen in Equation 4.12.
The jacobian computation is done using a standard Simulink block. The input is the robot configuration and the output is the Jacobian matrix corresponding to that configuration. The configuration which is inputted is the same configuration which is calculated by the inverse kinematics block and which is used as an input to the joint controller. Therefore, there seems to be no possible human error here.
The pseudo-inverse computation and joint velocity computation are done in another block. This computation can be seen in Figure A.6. Again, this is a very straightforward computation where there seems to be no room for human error.
All of these error sources have been checked and tested and no solution could be found, apart from assuming Simulink computation errors.

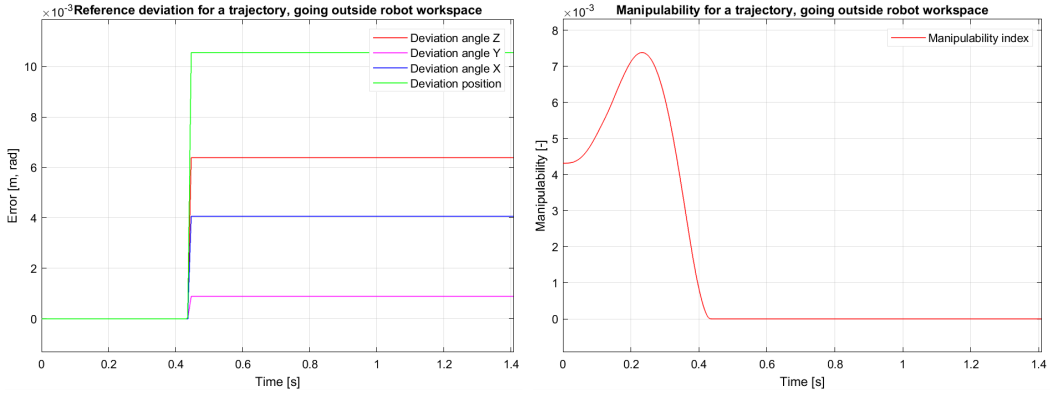
For that reason, a choice was made to use the derivative method to compute joint velocities as an input to the joint controller. As that option did function well.

4.4.3 Singularity induced reset

Robot singularities play a big role in the topics of multiple subquestions of this research. For that reason, a more thorough analysis was done on these singularities.

As proposed in subsubsection 4.2.6, an algorithm was put into place which detects trajectory deviations. To investigate the source of these trajectory deviations it was analyzed whether or not these also represented actual singularity configurations.
Just like in previous parts of this report, the robot's manipulability will be analyzed using the determinant of $\mathbf{JJ}^T$. Where $\mathbf{J}$ is its geometric jacobian.
If a the robot is in a singularity configuration, this value becomes zero [32].

A possible trajectory which has the inverse kinematics solution deviate from the trajectory is having a straight line going up to high, where the desired trajectory goes beyond the robot's workspace.
The inverse kinematics error, and the robot's manipulability for this trajectory can be seen in Figure 4.29.

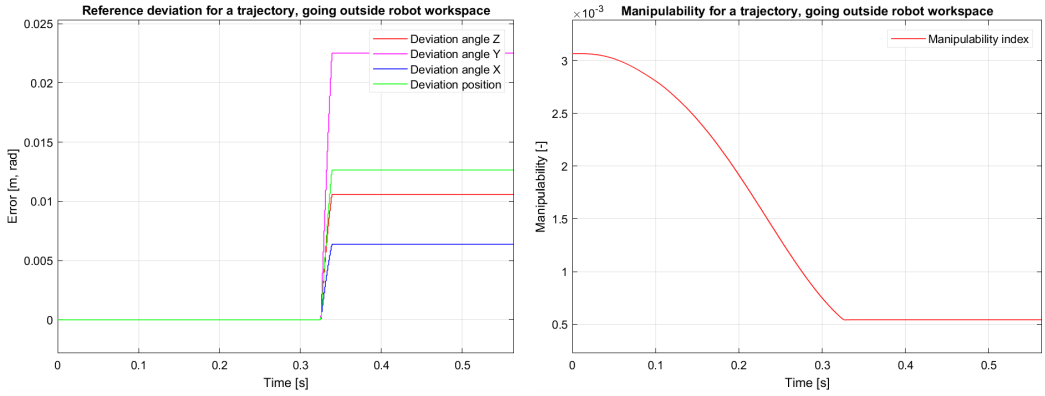


(a) (b)

Figure 4.29: (a): The trajectory deviation computed from the inverse kinematics error. (b): The manipulability of the robot, computed by the determinant of $\mathbf{JJ}^T$.

As you can see, the inverse kinematics solution is able to comply with the desired end effector pose for a very long time up until the end of the simulation. The important thing to note here is that as soon as the inverse kinematics error spikes upwards, the manipulability also becomes zero. So the configuration where the inverse kinematics could not find a solution anymore is a singularity configuration. This robot configuration resulted in the robot arm being fully extended. This means that no matter what, the inverse kinematics solution could not find a solution anymore to an end effector position even higher, as the physical limit of the arm was reached. This fully extended configuration can be seen in Figure A.7.

However, it is not always the case that the inverse kinematics error is completely unavoidable. Looking at a different, more complex scanning motion, it can be seen that the inverse kinematics solution runs into trouble before reaching its singularity. This simulation result can be seen in Figure 4.30



(a) (b)

Figure 4.30: (a): The trajectory deviation computed from the inverse kinematics error. (b): The manipulability of the robot, computed by the determinant of $\mathbf{J}\mathbf{J}^T$.

As you can see, in this case, the inverse kinematics solution is not able to compute a correct solution whilst the manipulability index suggests the robot is not at its singularity configuration yet.
This all has to do with the fact that the initial guess of the inverse kinematics plays a major role in the solution computation.
This had been previously already described as a positive fact since it allowed smooth end effector movement without large jumps in joint angles.
However, this strong connection with the initial guess also means that the new solution cannot deviate much from the original configuration. Therefore, it might actually be possible for the robot to reach a certain pose, but this pose is not accessible from the configuration it is currently at.

So as said, this is very good to make sure no sudden jumps in configuration occur. However, it does limit the robot's workspace in some cases.
A solution to this could be to completely reset the inverse kinematics calculation whenever such a problem occurs. This implementation would however, cause sudden jumps in configuration, which were previously avoided.

This so-called, singularity induced inverse kinematics reset, was introduced into the main controller. However, the sudden jumps in configuration resulted in too large joint errors. This resulted in similar phenomena as observed in Figure 4.5.
Therefore it is important to keep in mind that if it would be really necessary to reach 100% of the complete workspace something like this reset could be an option. However, in terms of usability it really ruins the robot's controllability. This option is therefore greatly discouraged.

Looking back at Figure 4.30, it is very clear that an inverse kinematics error does occur at configurations of low manipulability (i.e. singularity zones [23]). This shows again the importance of optimizing the robot's base position with respect to its manipulability [35]. This manipulability optimization method was already previously introduced in subsubsection 4.1.3 and its importance only becomes more clear from these tests.
If this manipulability optimization would be performed, these inverse kinematics errors could be avoided in almost all cases.

4.5 Joint controllers

With the joint angles computed by the inverse kinematics, it is time to start actuating the joint actuators.
This is done with the use of joint controllers, which transform the joint angles into joint torques.
If one were to use this system on the real Franka robots, these joint controllers would already be built into the robots. For that reason not much attention should be spent on trying to optimize this part of the control research.
However, in order to do useful simulations, these controllers must be modeled as well.

For this control architecture, the choice was made to use a PD computed torque controller.
This is a controller which is based on the following formula.[31].

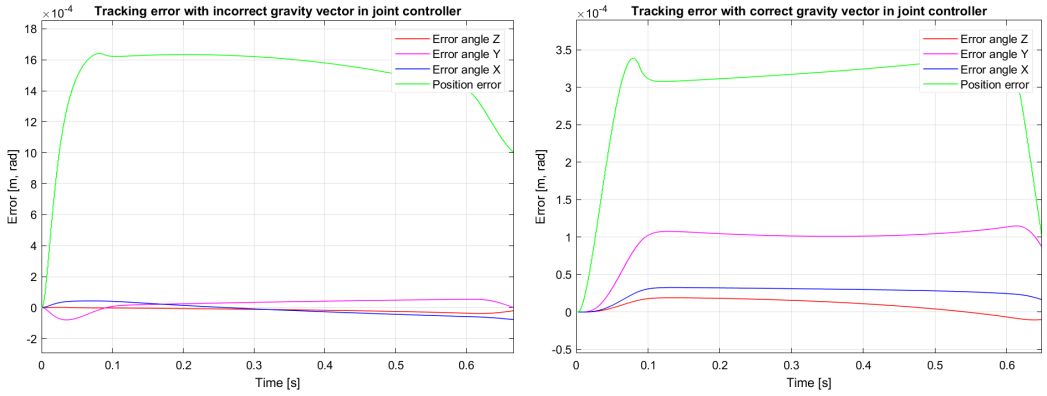
$$\boldsymbol{\tau} = M(\boldsymbol{q})\left[\ddot{\boldsymbol{q}}_d + K_d\dot{\tilde{\boldsymbol{q}}} + K_p\tilde{\boldsymbol{q}}\right] + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) \tag{4.13}$$

Where:

- $\boldsymbol{\tau}$ is the joint torque.
- $M(\boldsymbol{q})$ is the mass matrix, of the current robot configuration.
  This can be calculated using a Simulink block.
- $\boldsymbol{q}$ is the vector of the current joint angles.
- $\boldsymbol{q}_d$ is the vector of desired joint angles.
- $\tilde{\boldsymbol{q}}$ is equal to $\boldsymbol{q}_d - \boldsymbol{q}$, which is the joint angle error.
- K_d is the derivative gain.
  Which can be tuned by the control engineer, using: $K_d = 2\omega_n$
- K_p is the proportional gain.
  Which can be tuned by the control engineer, using: $K_p = \omega_n^2$
- $C(\boldsymbol{q},\dot{\boldsymbol{q}})$ is the matrix which calculates the induced joint torques from the current joint velocities.
  This can be calculated using a Simulink block.
- $\boldsymbol{g}(\boldsymbol{q})$ is the matrix which compensates for the gravitational effects on the robot.
  This can be calculated using a Simulink block.

By choosing a value for the natural frequency (ω_n) the control engineer is able to increase or decrease the controller gains. However, increasing the controller gains is only possible up to a certain point, as otherwise the computed joint torques will exceed the joint torque limits. See Table B.1.
For that reason a natural frequency of 100 was used in this research.

Lastly, it is important to address a process which this joint controller underwent.
As we saw, the computed torque is partially computed with the use of gravity compensation. For Simulink to compute this vector, it must know the direction of gravity with respect to the robot's base frame. However, if one was to place the robot upside down, the robot simulator would be made aware of this, however, the controller would not. Therefore, the controller would try to compensate gravity the opposite way.
To solve this the gravity vector of the robot would be computed in the Matlab initialization script based on the robot's base position.
The impact of having a correct gravitional vector on the error can be seen in Figure 4.31.



(a) (b)

Figure 4.31: (a): The tracking error when the joint controller used the incorrect gravity vector. (b): The tracking error when the joint controller used the correct gravity vector, which was defined in the initialization script.

From the large difference in error observed in Figure 4.31, it can be seen that a correct gravity compensation is crucial to accurately control the robot's pose.

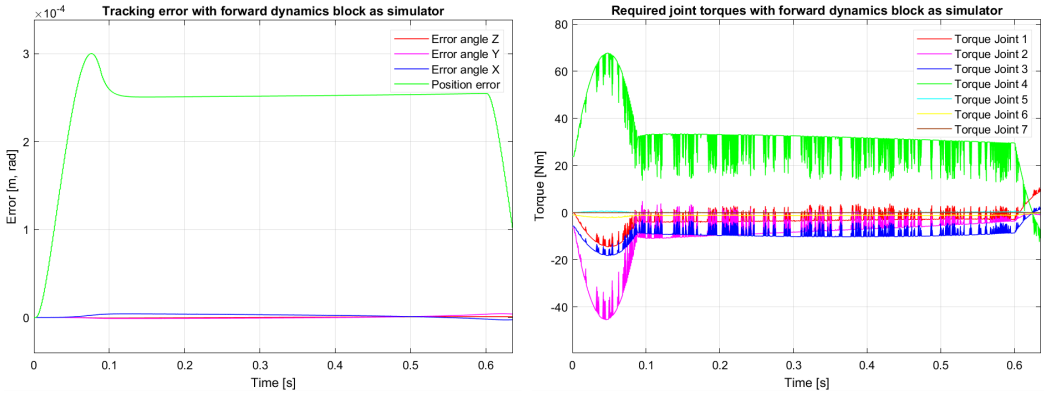
4.6 Robot simulator

The last part of this architecture is the actual dynamic simulation of the robot. This simulator will compute the robots behaviour based on the inputted torques.
For this, many options are available of which a couple have been tried or investigated.

4.6.1 Forward dynamics block

A standard block in Simulink which can be used to simulate a robot's dynamics is the 'Forward dynamics block' [43].
This block takes the current robot condition in terms of joint angles, joint velocities, joint torque and external force as input. With these inputs, it calculates the resulting joint acceleration. The calculation of this joint acceleration is based on the URDF model of the robot.
This block sort of seems like a 'black box' as it does not allow any customizability.
It also does not have a built-in visualization of the robot, which makes simulating different robot placements or environments more difficult as it will require a separate visualization method.

To analyze its performance, a simulation is performed with an arbitrary signal in which the robots error and controller-computed torque is illustrated.
The error is illustrated because it is the basis of this control architecture and the influence of the simulator on this error is therefore very important.
The controller-computed torque is also illustrated because it showed a pattern which was not present in other simulators.
These signals can be seen in Figure 4.32



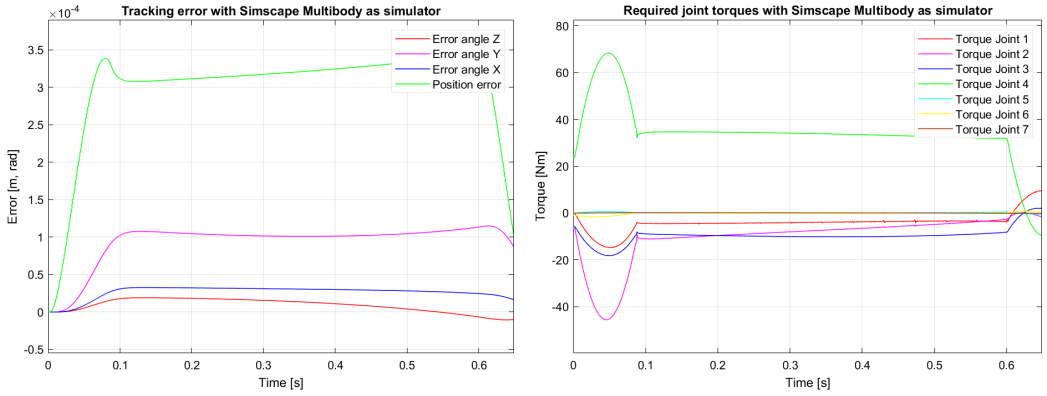
(a) (b)

Figure 4.32: (a): The tracking error when the robot's behaviour is simulated using the forward dynamics block.
(b): The require joint torques, which exhibit strange behaviours when the robot is simulated using the forward dynamics block.

By itself, the error graph does not tell much. Therefore it must be compared to other simulation options. This will be done once those simulation results are introduced in their coming subchapters. As said previously, the required torque signal seems to exhibit some strange behaviour. Whether this is dependent on this particular simulator or not will also be seen later, by comparing it to another simulator.

4.6.2 Simscape multibody

The other method of simulating robots directly in Simulink, is using Simscape Multibody [44]. This is a more advanced method of simulation which is also based on the robot's URDF model.
It is more advanced as it does have a built-in visualization and allows for more customizability in terms of robot placements, environments etc.
It is however, still lacking in a few categories such as sensor simulation and body collisions.
For the same reasons as described in the previous subchapter, this simulator will be analyzed on error and torque as this will allow simulator results to be compared. These results can be seen in Figure 4.33.



(a) (b)

Figure 4.33: (a): The tracking error when the robot's behaviour is simulated using Simscape Multibody.
(b): The require joint torques, which do not exhibit the same strange behaviour as previously described.

Looking at the error, it can be seen that they look quite alike in terms of shape. However, there are also some differences. It can be seen that the position error with the Simscape simulator is about 20% larger. On top of that, the orientation error is much more relevant. In the forward dynamics simulator, the angular errors seemed to have almost no error, but in the Simscape simulator these errors do play a significant role.
Comparing the required torques, it can be seen that with the Simscape simulator this strange behaviour is not present anymore. Which makes it a simulator dependent behaviour of which the cause is still unknown.

By comparison it can be seen that in fact there is a difference in computation method between the forward dynamics block and the Simscape Multibody model. Giving a verdict on what the causes of these differences would be, or which one is more correct is difficult at this moment.

4.6.3 Gazebo

The last simulator that was investigated is Gazebo [45].
The gazebo simulator setup consists of a co-simulation between Simulink and Gazebo. Co-simulation is required because Gazebo is an external simulation software which is not directly included in Simulink unlike the previous two examples.
This makes the setup more difficult, which is a disadvantage. However, the reason one would choose this simulator is for its advantages.
Gazebo is a more advanced method of simulating robots which is able to simulate more complex scenarios. To be specific, gazebo is able to simulate sensors, sensor noise and body collisions. This is where the previous simulators were especially lacking.

The setup of the co-simulation between Simulink and Gazebo can be done in two ways. Either using Gazebo co-simulation blocks. Or using ROS co-simulation blocks.
A choice was made to co-simulate the robot in Gazebo using the ROS co-simulation blocks. This choice was made because it is more 'universal'. If one was to change to a different simulator which also works on the ROS environment, the co-simulation model would stay relatively the same. Also controlling the real Franka robots is done using ROS.

In this chapter it was proposed that simulation environments would be compared. However, in the case of the Gazebo simulator this is not very useful. This is because the input method of the Gazebo simulation works much different from the other two simulators.
The gazebo simulator takes joint positions as input, instead of Torques. On top of that, the Gazebo simulator takes a future position goal as input, for this position goal one has to determine a time it

has to reach it. However, this time has to have a value of around at least 0.5sec. This is completely different from the other simulators which just require a current torque input.
A very simplistic simulation was performed of which the error signal can be seen in Figure 4.34.

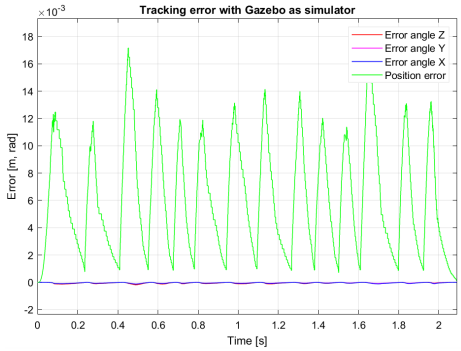


Figure 4.34: The tracking error with a simplistic Gazebo co-simulation

As you can see, this is a very poor simulation in terms of error amount.
The main reason for this is the different input method Gazebo uses. With this method only the position goal was sent to the robot. So the velocity target was set at zero for every waypoint. As we saw in Figure 4.8, this is disastrous for the control of a robot.
So a next step for improving the robot's control with a Gazebo simulation is figuring out the best method of sending future position goals, and adding velocity goals to it as well.

4.6.4 Real Franka Emika Panda

The last method of actuating and sensing robot conditions is the real Franka Emika Panda robot. As explained in the previous subchapter, the real Franka robots use ROS to be operated. Therefore, if one was to solve the Gazebo simulator, the same co-simulation architecture could be used for the real Franka robot as well.
To send future position and velocity targets, one has to send messages using Publishing blocks. To receive sensor data, one has has to receive messages using a Subscriber block.
Because this research focused more on the control architecture rather then the real robot operation, this subject together with the Gazebo simulator, was not much more explored.

4.6.5 Simulator comparison

As we have seen in the last subchapters, three main simulators have been investigated. The Forward Dynamics block, the Simscape Multibody environment and the Gazebo co-simulation.

Because of the poor setup used in the Gazebo co-simulation, the results were also very poor. For that reason, only a recommendation can be given to explore this actuation method more which will enable a better simulation result. Apart from the recommendation, this Gazebo co-simulation is disregarded from the simulator comparison choice.

Then the Forwards Dynamics block and Simscape Multibody environment remain. In terms of simulation accuracy it is hard to say which simulator computes the most correct results. This is because it is not known what the correct solution actually is.
However, in terms of usability and customizability, the Simscape Multibody environment is far superior. The proposed goal of this report, is to make a control and simulation environment which is able to control and simulate many different options. If the simulator is however, not able to customize robot base positions for example, it is also not possible to test the controller on this situation.
So for the amount of customizability, and relative ease-of-use, the Simscape Multibody environment was chosen to simulate the robot's.

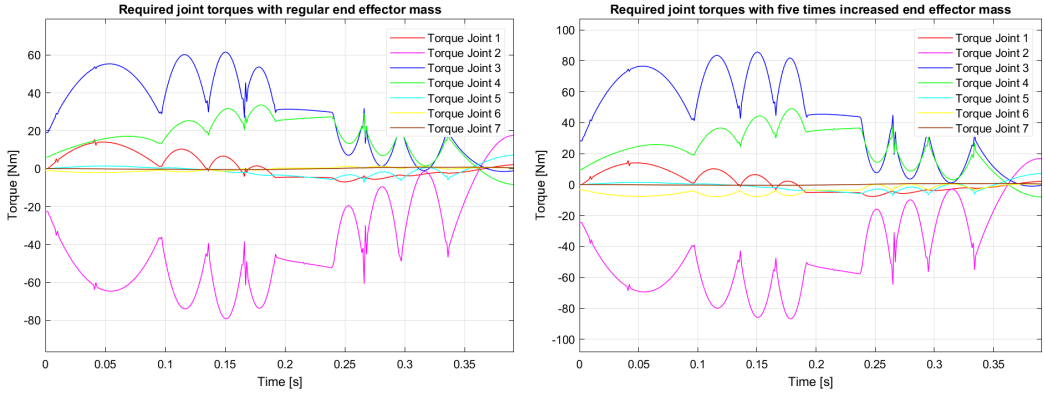
As said, it is hard to determine the absolute accuracy of this simulator. However, it is important to understand the controller choices made in this report are based on relative comparisons. Every time the a controller choice was tested, there performances were compared relative to each other. Therefore, the absolute accuracy of the simulator is not too important as only the relative controller performances were compared.

5 Subquestion specific research

In this report, a couple of subquestions are investigated. Many of these questions were the basis of some of the design processes described in the previous chapter "The process". However, an extra subquestion which was not fully investigated in that chapter is the non-identical robot combination as it is quite a separate part of the research.

5.1 Different end effector mass

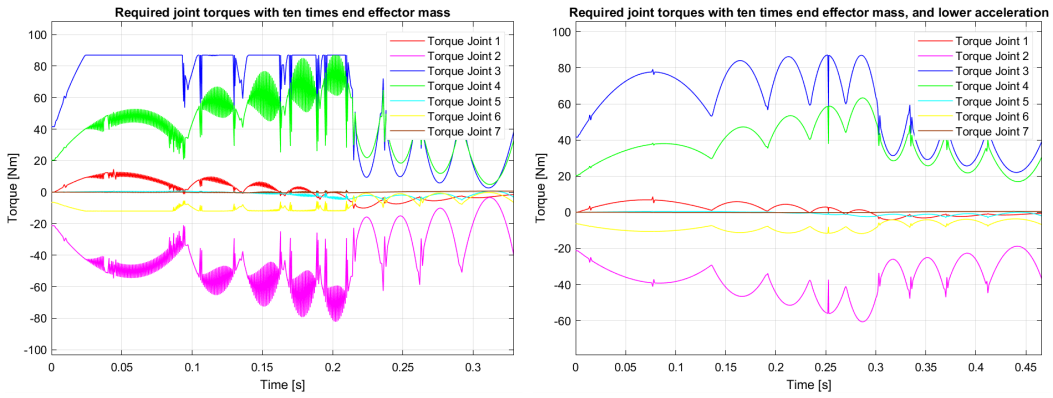
The first option of having a non-identical robot combination is with two Franka robot's with different end effector masses.
The proposed controller is based on having a very robust, user-customizable, architecture. This means that the controller should be able to handle different situations and environments. This is very beneficial for using non-identical robots.
If the end effector mass of the follower robot was to be increased, the required joint torques would also increase because the gravitational and inertial forces would increase.
The original mass of a Franka end effector is 0.7[kg]. The effect of increasing the end effector mass by five times, on the required joint torques can be seen in Figure 5.1



(a) (b)

Figure 5.1: (a): The required joint torque for a regular end effector mass on an arbitrary trajectory. (b): The required joint torque for a five times heavier end effector mass on the same, arbitrary trajectory.

As you can see, the required joint torque increased. However, the increase is very minor even though the end effector mass increased from 0.7 [kg], to 3.5 [kg].
In this scenario, the mass increase resulted in so little torque increase, the user would not even have to change any controller options such as the maximum acceleration.
If one is to increase the end effector mass even more, by 10 times for example, the required torque reaches the joint torque limits. This can be seen in Figure 5.2a. However, as said, because this controller is very robust and user-customizable, it is very easy to slightly decrease the maximum allowed acceleration. By decreasing this, the trajectory is again able to be followed without reaching torque limits. This can be seen in Figure 5.2b



(a) (b)

Figure 5.2: (a):The required joint torque for a ten times heavier end effector mass on an arbitrary trajectory. *Note: The strange lines have appeared here as soon as one of the torques reaches its limit. Exactly why this happens is uncertain.*
(b): The required joint torque for a ten times heavier end effector mass on the same, arbitrary trajectory, with a tuned maximum acceleration.

By tuning the acceleration, one is able to accurately control a different end effector mass. In this case the maximum end effector acceleration had to be decreased from $10m/s^2$ to $5m/s^2$.
By tuning the acceleration of the follower, with the heavier end effector, the follower is able to complete the leader-follower trajectory in combination with the regular leader again. If it is desired that both end effector have the same acceleration, the exact transformation leader-follower algorithm can be chosen by the user to ensure that both end effector move at the same acceleration and velocity.

In summary, it is easy to see that even with these different types of situations the controller is always able to succeed in controlling both robots the way the user desires.

5.2 Different robot mass

Previously, the effect of a different end effector mass was investigated.
It was shown that a higher end effector mass needed a lower acceleration to satisfy joint torque limits. This is obvious from the the fact that the inertial- and gravitational forces scale with mass. However, a ten times increase in end effector mass did not require a ten times decrease in acceleration because the end effector is only a part of the complete robot. Therefore, it is also interesting to see the effect of changing the complete robot's mass. As a more concrete relation between mass and joint torques might exist there.

5.2.1 Existing forces

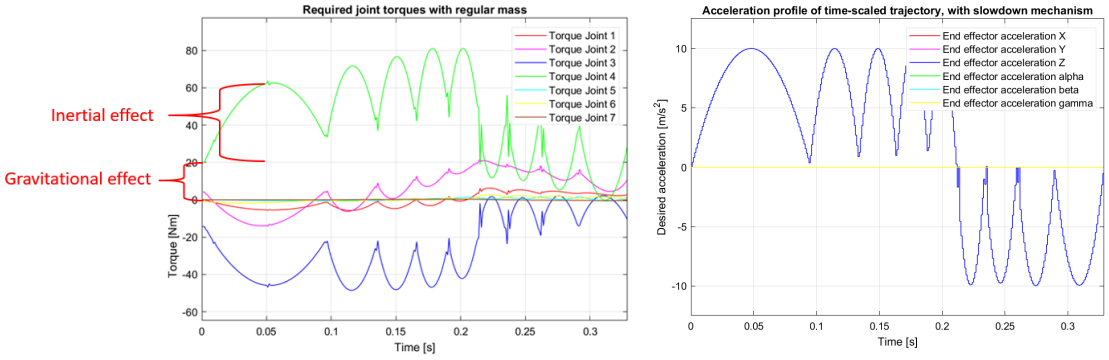
First of all, it is important to understand that the force which play a role in the determination of the joint torque are the inertial- and gravitational force.

The gravitational force is always present on the robot regardless of its condition, so whether or not it is moving, accelerating etc.

The inertial force however, is applicable whenever the joints are accelerating. This does not necessarily mean the end effector is also accelerating. Since an end effector with constant velocity can still require accelerations of joints because every end effector position requires different joint configurations.
However, investigating the joint accelerations required for a constant end effector velocity is almost impossible because every imaginable trajectory will require different joint accelerations for different constant end effector velocities. For that reason, when talking about inertial forces, in the coming chapter, it will be assumed that the accelerations in joint space only correspond to accelerations of

the end effector. As explained, this is not necessarily true, but from experiments it has proven to be close enough for the scope of this research.

To visualize these two effects, inertial and gravitational, the following typical acceleration-to-torque comparison is very useful, seen in Figure 5.3.



(a) (b)

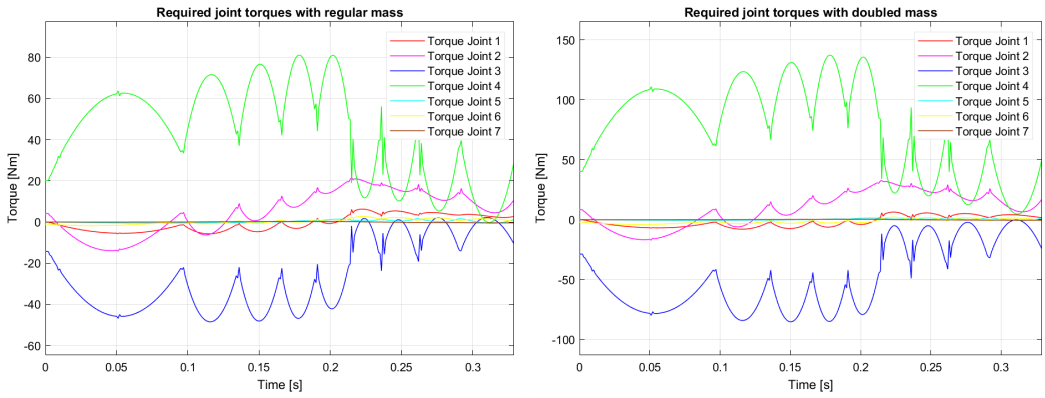
Figure 5.3: (a):The required joint torque for an dynamically changing acceleration profile. (b): The desired dynamically changing acceleration profile.

To see the effect of the gravitational force, on the required torque it is good to look at the required joint torque at t=0. At this time the robot has no velocity, no acceleration and no error. So the required torque at t=0, is solely to counteract the force of gravity.
Then, to see the effect of the inertial forces, the first 'bump' in torque can be associated with the first 'bump' in acceleration. It is clear that the relation between end effector acceleration and joint torque is strong as they display the same shapes.
To measure the influence of the end effector acceleration to the joint torque, the first 'bump' is always particularly the most interesting as here the other influences, such as error and end effector velocity (which causes extra joint accelerations) is the lowest of all 'bumps'.

With the fact that the present forces are gravitational or inertial, the buttons one can turn on to influence the joint torques is either, the mass, or the acceleration. Since the mass is the changed value in this research question, the variable the control engineer should tune is the acceleration.
To figure out how this acceleration should be tuned depending on the altered robot mass, both forces will be analyzed.

5.2.2 Influence of changing mass on the gravitational force.

First of all, let's remove the joint torque limits for now and investigate what happens to the joint torque if the complete mass of all robot components is doubled. The original versus the doubled version torques can be seen in Figure 5.4.



(a) (b)

Figure 5.4: (a):The required joint torque for a robot with regular mass on an arbitrary trajectory. (b): The required joint torque for a robot with doubled mass on the same, arbitrary trajectory.

At the beginning of the simulation, the velocity and acceleration of the end effector, and therefore joints as well, is zero. That means that at t=0, the only torque the joints have to produce is to overcome the gravitational force.
Looking at the torques at t=0, is is apparent that the doubled mass, also requires exactly double the torque.
So this experiment backs up the theory that the gravitational force has a linear relationship with the robot mass.

5.2.3 Influence of changing mass on inertial force.

In Figure 5.3, the inertial effect was defined. Looking at the doubled mass comparison in Figure 5.4, a less expected result is seen. It would have been expected that, similarly to the gravitational effect, the inertial effect would have scaled linearly with the mass. However, this is not the case, when looking at the exact scaling factor, the inertial effect only increased by a factor of 1.6, whilst the mass was doubled.

Since there was no linear relationship, an additional experiment was done. Multiple mass multiplier were tested and their inertial effect was written down. All these robots with altered masses performed the exact same trajectory. The results can be seen in Table 5.1

Mass Multiplier	Inertial effect size	Inertial effect ratio	Computed ratio (=1+0.6(x-1))
0.5	29.5	0.70	0.70
1	43	1.00	1
2	69	1.60	1.60
3	95	2.21	2.20
4	121	2.81	2.80
5	148	3.44	3.40
6	175	4.06	4.00
10	280	6.51	6.40
20	540	12.5	12.4

Table 5.1: The experimental data, of which the last column represents the expected inertial effect ratio based on the proposed formula.

With these data, a formula could be computed which calculates the expected inertial effect ratio, based on the mass multiplier. So if a mass multiplier is known, by how much will the original inertial

effect of that trajectory scale.
This formula came out to be Equation 5.1

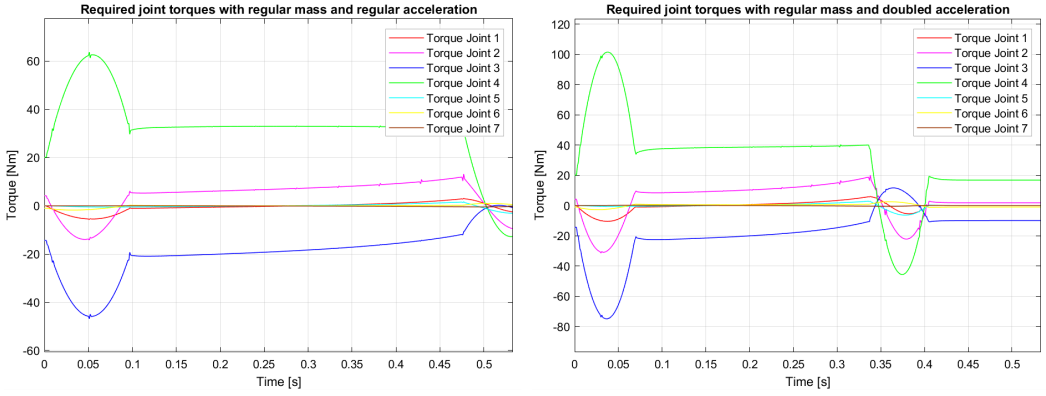
$$Ratio = 1 + 0.6(x - 1) \tag{5.1}$$

Where x is the mass multiplier.
The computed ratio values using this equation are shown in the last column of Table 5.1. These expected values seem to come quite close to the experimental data.

The reason the mass to inertial factor ratio is not 1:1 is still uncertain.
It could have to do with the assumption made about the joint acceleration only being linked to end effector accelerations, which is not fully correct as explained.
To verify this as the origin of this non-linear relationship, it is important to test the relation between varying the end effector acceleration and the inertial effect.

5.2.4 Influence of changing acceleration on inertial force.

By keeping the mass and trajectory the same, but only varying the acceleration, the effect of acceleration on the inertial force can be investigated. A comparison between $10m/s^2$ and $20m/s^2$ can be seen in Figure 5.5.



(a) (b)

Figure 5.5: (a):The required joint torque for a robot with an acceleration of $10m/s^2$.
(b): The required joint torque for a robot with an increased acceleration of $20m/s^2$.

As you can see, the regular inertial effect is from 20 to around 60 [Nm], whilst the inertial effect, for the doubled acceleration robot, is from 20 to 100 [Nm]. So the inertial force does scale linearly with the acceleration.
The fact that the inertial force does scale linearly to the end effector acceleration shows that the assumption made in this chapter does hold to a significant extent.
This however, makes the mystery of the non-linear relationship between mass and inertial force unexplained.

5.2.5 Final tuned acceleration relationship

As explained, the to-tune variable for the control engineer is the maximum allowed acceleration. So with the following relations a plan of how to tune the acceleration can be made.

- The linear relation between the gravitational force and mass.
- The non-linear relation between the inertial force and mass.
- The linear relation between the inertial force and acceleration.

With all these relations, the torque of an altered-mass robot can be related as seen in Equation 5.2.

$$T_{new} = MM \cdot T_{G,old} + \frac{a_{new}}{a_{old}} \cdot [1 + 0.6(MM - 1)] \cdot T_{I,old} \tag{5.2}$$

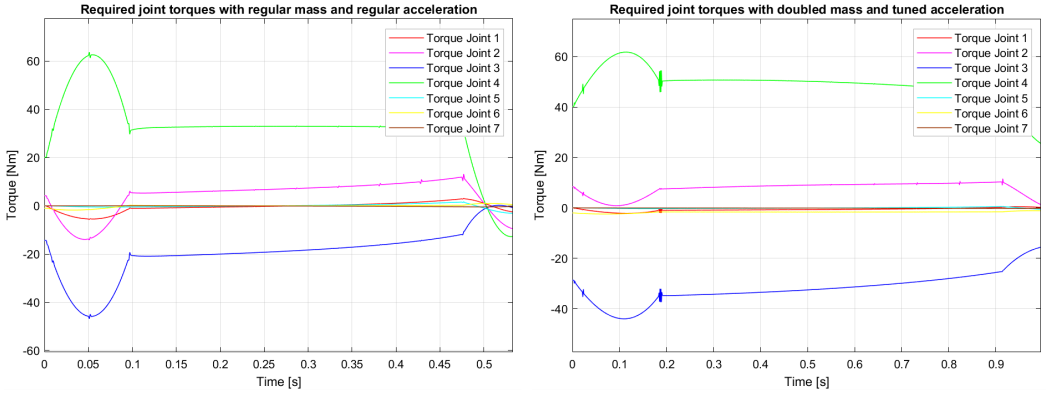
Rewriting this to solve for the new acceleration gives Equation 5.3.

$$a_{new} = \frac{T_{new} - MM \cdot T_{G,old}}{[1 + 0.6(MM - 1)] \cdot T_{I,old}} \cdot a_{old} \tag{5.3}$$

Where:

- a_{new} is the tuned acceleration which should be used in order to satisfy the chosen new torque
- T_{new} is the new peak torque, chosen by the user.
- MM is the mass multiplier. By how much is the mass of the original robot scaled.
- $T_{G,old}$ is the torque associated to the gravitational force in that particular robot configuration/trajectory for the original robot.
- $T_{I,old}$ is the torque associated to the inertial force in that particular robot configuration/trajectory for the original robot.
- a_{old} is the maximum acceleration used in the original robot.

From this relation one is able to compute the maximum end effector acceleration based on the mass multiplier, the original robot conditions and the desired joint torque.
To verify this, let's have a look at a robot with regular mass, and a maximum acceleration of $10m/s^2$ on an arbitrary trajectory. This trajectory is performed once. This result can be seen in Figure 5.6a. If this trajectory is performed once, Equation 5.3, is able to use this information to calculate what sort of tuned maximum acceleration is required when the robot's mass is altered, depending on the desired peak torque, chosen by the user.
From this original robot situation, seen in Figure 5.6a, the values for $T_{G,old}$ and $T_{I,old}$ can be found. Their values are 20[Nm] and 42.5[Nm], respectively.
Now the robot is made two times heavier. On top of that, the user chooses that the new maximum peak torque must be 60[Nm], instead of the original 62.5[Nm]. From Equation 5.3, the computed new maximum acceleration comes out to be $2.94m/s^2$.
The resulting torque can be seen in Figure 5.6b.



(a) (b)

Figure 5.6: (a):The required joint torque for a regular robot with an acceleration of $10m/s^2$. (b): The required joint torque for a robot with double mass with specifically tuned acceleration of $2.94m/s^2$.

As you can see in Figure 5.6b, the Joint torque 4, which the acceleration tuning was based on, peaks at 61.8[Nm]. The acceleration was tuned to satisfy a maximum acceleration of 60[Nm]. So the result of tuning the acceleration based on Equation 5.3 is very close.
The slight inaccuracy of 3% most likely results from the non-perfect relations determined in the previous subchapter such as the relation between the inertial force and mass, which was, unexpectedly, non-linear.

In the above example, the acceleration tuning was based on the 'Torque Joint 4' signal since it was the largest, and therefore most dangerous for the torque limits. However, this acceleration tuning can be based on whichever torque signal wants to be directly tuned for.

5.2.6 Different robot mass overview

In the beginning of this section it was mentioned that by altering the end effector mass alone, no real informative decision on acceleration tuning could be given. For that reason this complete robot mass altering question was investigated.
In this chapter it was indeed discovered that a more concrete relation between mass alterations and maximum acceleration exists. From testing this relation has shown to prove very useful as in most mass alterations the predicted acceleration is able to satisfy torque design choices.
Being able to tune accelerations to fit certain torque limits is very useful as it will allow complex trajectories to be followed by the robot without running into torque limits.

5.3 Different follower robot

The last non-identical robot combination is the most extreme situation.
Previous options had focused on altering masses of particular robot parts or complete robots. However, in this case, one of the robots is a completely different robot (i.e. heterogeneous robot combination)
The robots which will be investigated are the Kinova Gen3 robot and the Universal UR10 robot.

5.3.1 Kinova Gen3 robot

Similarly to the Franka robot, the Kinova Gen3 robot is a 7DOF robot.
Apart from that similarity, the robot will function different because of its physical differences in dimensions, mass, joint (torque) limits etc.
These physical differences will have to most impact on the robot's workspace and torque restrictions.

Workspace comparison

First of all, lets investigate the robot's workspace.
The workspace provided by Kinova [46], can be seen in Figure B.2. Comparing this workspace to the workspace of the Franka robot [6], seen in Figure B.1, it can be seen that both workspaces are quite similar. This is because both robots have similar arm length. Though, the workspace of the Franka robot seems to be a bit larger.

Instead of looking at the theoretical workspaces, the workspaces were also investigated by testing both robots of many different trajectories, such as circles with varying radii or straight lines forward or upward for example.
From these tests the theoretical workspace similarity was confirmed as both robots were able to follow similar trajectories. However, the Franka robot is able to reach out a bit further due to its slightly longer arm.

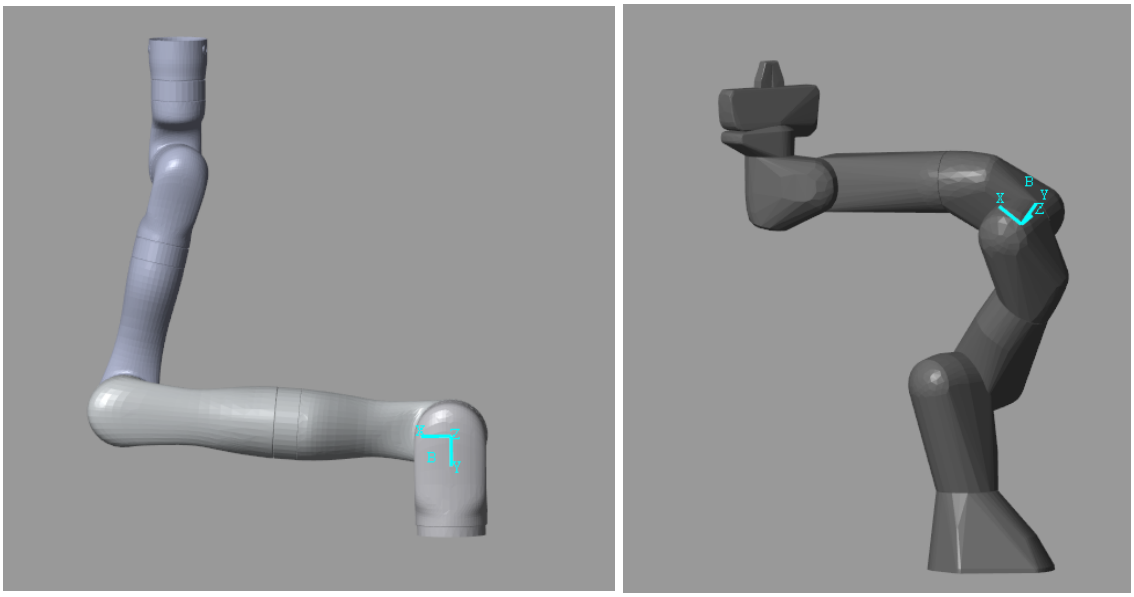
The last method of comparing workspaces is by looking at the manipulability index across certain positions [34]. Previously, a manipulability optimization was performed for the Franka robot, seen in Figure A.3. This manipulability optimization showed that for that trajectory, the x-coordinate should ideally be 0.35[m].
This exact same trajectory is now also optimized for the x-coordinate for the Kinova Gen3 robot. This optimization, seen in Figure A.8, showed the optimal x-coordinate to be 0.2[m].

This again shows the usefulness of a manipulability optimization as it shows that it is very robot dependent what end effector pose is optimal.

The Kinova Gen3's workspace has been evaluated with respect to the Franka's workspace. It turns out that the Kinova Gen3 has a slightly smaller workspace due to its shorter arm length.
The manipulability analysis shows that the Kinova Gen3 robot has a different optimal operating point.
In summary, the Kinova Gen3's robot seems to have a less favourable workspace for the tested trajectories. However, the proposed manipulability optimization should allow to avoid most negative effects of this poorer workspace.

Torque comparison

Compared to the Franka robot, the Kinova Gen3 robot acts very different.
Because of the physical built-up of the Kinova Gen3 robot, a lot of stress can build up in specific joints. The reason for this is its direct alignment of joints with long links in between. This creates a very large moment on particular joints in particular configurations.
A comparison example of this can be seen in Figure 5.7.

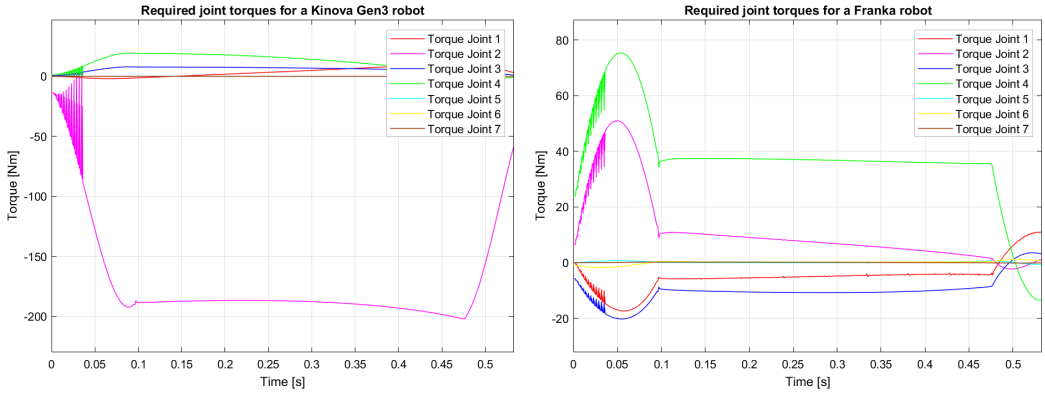


(a) (b)

Figure 5.7: (a):A very stressful position for the joint 2. (The axis frame of this joint is blue for clarity).
(b): A much less stressful position the Franka robot, with the same end effector pose. (A similar stressful joint, Joint 4, is marked blue as well here)

In this comparison, both robots have the same end effector pose with respect to its base. However, the Kinova Gen3 robot is in a much more stressful position when evaluating Joint 2, close to the robot base. This is because a big part of the robot has a very large distance perpendicular to the gravity vector, pointing downwards.
The Franka robot has a somewhat similar shape. However, this shape is much less stressful for two reasons.
The first is that only the last part of the robot has to be held up by this stressed joint. In the Kinova case, the part which had to be held up by the stressed joint was a much larger part of the robot.
The second reason is the the stressed joint axis, for the Franka robot, is not exactly perpendicular to the gravity force. Therefore, can the gravity force be more evenly distributed to other joints which are also at some angle to the gravity force.

This large dependence on a single joint requires this joint to produce a lot of torque. For an equal trajectory, the required torques of both robots looks as follows. See, Figure 5.8



(a) (b)

Figure 5.8: (a): The required joint torque for a Kinova Gen3 robot.
(b): The required joint torque for a Franka robot.

As you can see, the Kinova Gen3 is much more reliant on a single joint to follow this particular trajectory causing this joint torque to become very large.
On top of this large required joint torque, the physical torque limits, seen in Table B.1, for the Kinova Gen3 are also lower. Which makes this phenomenon even more problematic.
For that reason, must the trajectory not only be optimized in terms of manipulability, as explained previously. The trajectory must also be optimized to overcome such single-joint-dependent robot configurations.

5.3.2 Universal UR10 robot

The universal UR10 robot is a very similar robot to the Kinova Gen3 and Franka robot. However, it is a 6 DOF robot, unlike the Franka and Kinova Gen3 robot. Which means that it is no longer a redundant robot [31]. Which in theory would mean that it is less manipulable in most configurations. Therefore, it is very interesting to compare its manipulability analysis to the manipulability analysis of the original, Franka robot.
This manipulability analysis was performed on the same exact trajectory again and optimized for the x-coordinate. Apart from the optimal x-coordinate, an important result of this analysis are the absolute manipulability values retrieved. It is expected that these will be lower, due to its lower amount of DOFs.
The result of this manipulability analysis can be seen in Figure A.9 and Figure A.10.

As it turns out, the optimal x-coordinate for highest manipulability for this robot is around 0.9[m], for this trajectory.
The Franka robot was able to manipulate itself at an x-coordinate of 0.0 or 0.1 [m]. This is not the case for the universal UR10 robot.
However, the universal UR10 robot is able to manipulate up until an x-coordinate of 1.1[m]. Whilst the limit of the Franka robot was already at an x-coordinate of 0.6[m].
So from this analysis a control engineer would immediately be able to understand the pros and cons of the universal UR10 robot compared to the Franka robot.

In terms of absolute numbers, the manipulability index of the universal UR10 robot seems to be higher in most cases. This was unexpected as it had a lower amount DOFs.
So from this it can be seen that a manipulability analysis might not be a good comparison tool between robots but is more applicable to compare trajectories of the same robot.

Even though this manipulability analysis may not portray it, it is important to state the useful difference between a redundant and non-redundant robot.
A redundant manipulator will have many more non-unique inverse kinematics solutions for an arbitrary end effector pose [31]. This is a huge advantage for robots operating autonomously as it allows the robot to choose different joint configurations which all satisfy the same end effector pose. This is useful as it gives the robot more options to avoid obstacles.

The take-away message of this comparison between the Universal UR10 and the Franka robot is that again, the optimal manipulability coordinates are very robot dependent. However, more importantly, the manipulability analysis should be used to optimize for a specific robot, but should not be used to compare manipulability indices between robots.
As in this case the manipulability analysis would have convinced the control engineer that the 6DOF robot would be more manipulable then the 7DOF robot. This is counterintuitive looking at the current understanding of redundant robots in literature [47].

6 Results

Now that the research procedure and design process have been documented it is important to evaluate the result of this control architecture in a concise manner.

In this report a control architecture was designed and simulated using Matlab and Simulink.
This control architecture was built to satisfy high configurability by the user. This high configurability was explained to be important for the many different use cases of leader-follower robot systems.
Concretely, the configurable factors included in the control approach is the following.

- Choice of trajectory
- Choice of waypoint sample rate.
- Choice of leader-follower algorithm. Specifically:
    1. Conventional leader-follower algorithm
    2. Leaderless algorithm
    3. Perfect transformation algorithm
- Positioning and orientation of follower base w.r.t. leader base.
- Positioning and orientation of follower end effector w.r.t leader end effector.
- Positioning of robots' bases to allow optimal manipulability along the trajectory.
- Choice of end effector velocity, follower and leader values can differ if desired.
- Choice of end effector acceleration, follower and leader values can differ if desired.
- Choice of error calculation. Specifically useful for the 'perfect transformation algorithm'.
- Choice to allow the robot to deviate from its trajectory or not.
- Choice to let the controller optimize performance by making a trade-off between performance and accuracy (i.e. Time scaling).

Next to allowing wide configurability, the control architecture is optimized on many aspects.
The aspects which have been designed to allow the greatest possible performance and accuracy are the following.

- Position target has been designed to have the perfect continuous signal without artefacts around waypoint 'hits'.
- Velocity target has been designed to allow for continuous velocity and acceleration.
- Acceleration target has been designed to help satisfy joint torque limits.
- Start-to-end trajectory has been designed to start and end at zero velocity whilst having continuous position, velocity and acceleration in between.
- A forward-looking slowdown mechanism has been designed to optimize decelerations. Again, to help satisfy joint torque limits.
- Waypoint-to-waypoint trajectory generator was designed to ensure continuous position, velocity and acceleration.
- A Simscape Multibody based simulator was designed to allow all, previously stated, configurable aspects to be simulated.

Next to these configurable aspects and optimizations, this report has also suggested plans as how to go about controlling other robot systems.
These robot systems and their plans are the following.

- Robots with different masses. A prediction relationship is proposed. This relates conditions of the original robot on a particular trajectory, to how a robot with a different mass could solve that trajectory.
- Heterogeneous robot combinations. It is shown that the proposed manipulability analysis is a good method for determining the optimal zone of a robot's workspace. It is shown that the coordinates of these optimal zones can vary greatly amongst robots.
- Operating close to robot singularities. Again, with the use of manipulability analyses it is proposed to try and avoid singularity zones. With the emphasis on zones, because the inverse kinematics solver is shown to 'get stuck' before fully reaching singular configurations.
  To decrease these singularity zones a inverse kinematics reset is proposed, but is ill-advised.
- Autonomous navigation, this is less concretely discussed. However it is repeatedly designed for by having high-configurability, optimizing for manipulability and comparing redundant to non-redundant robots.

7 Conclusion

With all the results gathered and described, it is time to answer the proposed research questions of this report.

7.1 Main research question

The main research question that was proposed for this report was as follows.

How can you accurately control two robots performing trajectories based on the leader–follower principle? The trajectories will be moving towards-, away- and around a shared goal with the end effectors pointing towards each other. These trajectories should be performed with a maximum position error of 1 [mm]. On top of that, the trajectory control algorithm should be able to ensure safety.

As described in the results chapter, many different aspects in the controller were designed to accurately control the two robots. Of this list, the most important design choices which lead to the greatest amount of accuracy and performance are the following.
The concept of continuous position, velocity and acceleration signals [15], [16].
This was solved by using the optimized target values, an optimized start-to-end trajectory method and an optimized waypoint-to-waypoint generator.
To properly actuate the robot based on these continuous signals, the robot is controlled in position using the inverse kinematics block and in velocity by using the derivative method.
In addition, prediction methods and forward-looking algorithms are designed to satisfy joint torques whilst increasing performance demands [11].
Further, a trajectory time scaling algorithm was designed which is able to trade of performance and accuracy.

In addition to optimizing for accuracy and performance, safety was also taken into consideration.
To allow for real-time collision avoidance, a robot manipulator must be able to easily and quickly manoeuvre itself. To ensure this, a manipulability optimization was performed which will aid any future collision avoidance algorithms.
Next to collisions, another risk factor is strange, unexpected robot movements. This is solved by incorporating the deviation detection algorithm.

The proposed hypothesis of this research question at the beginning of this research was the following.

In order to properly track the proposed trajectories it is important to transform the target waypoints to smooth point-to-point trajectories which will enable a control strategy based both on position and velocity control of the robots. The trajectory should be able to avoid collisions and trade-off accuracy and velocity.

Comparing this hypothesis to the above research answer, many similarities are noticed. However, the hypothesis seems to focus more on the actual joint control of the robot, which was later explained in the report to be less important. The research answer to this question has put more thought into optimizing the trajectory in many different ways so that any decent joint controller can satisfy performance and accuracy targets.

7.2 Secondary research questions

This report also focused on a couple of secondary research questions, their answers will be evaluated here.

7.2.1 Trajectory profile

What trajectory profile is most optimal to allow the robots to be controlled in position and velocity?

The design process clearly showed that a trajectory profile which was able to satisfy the most amount of continuous signals on position, velocity, acceleration and jerk level was superior. The design process was only able to get up to a trajectory level which satisfied continuous position, velocity and acceleration. The discontinuous behaviour of the jerk signal also showed to cause minor accuracy decreases.
With this in mind the research answer to this question is that a trajectory should be used which generates continuous displacement, velocity, acceleration and jerk signals.
The hypothesis on this question was as follows.

In order to both accomplish position and velocity control a trajectory method where these parameters are not predetermined is required. This is most applicable in a polynomial profile. The order of polynomial should be chosen to satisfy continuous displacement, velocity, acceleration and jerk.

This hypothesis is perfectly in line with the research answer. This is also expected since the hypothesis was based on strong literature evidence [15], [16].

7.2.2 Leader-follower principle

What sort of leader–follower principle is most applicable in this medical use case?

In this report, the imaginable leader-follower use cases in a medical sense were able to be classified into three algorithms.

1. Conventional leader-follower algorithm
2. Leaderless algorithm
3. Perfect transformation algorithm

The conventional leader-follower algorithm was proposed for situations where only the leader's trajectory would be known [20], [36]. Whilst also having no real interest in the exact transformation between robots. An example use case would be a situation the two robots are moving towards a target, whilst the leader robot has to autonomously decide the trajectory in real-time [37].
The leaderless algorithm, is very similar in the case that the transformation between robot is not as important, for example when the robots are moving towards a target. However, in this case both robots can progress beyond each other, which requires both robots to known there trajectory beforehand. This could be if no autonomous navigation is required or if both robots can independently autonomously decide their trajectory.
The last algorithm, the perfect transformation algorithm, is very useful for delicate, collaborative operations. Such as x-ray scans where both robot's have to fulfill a task collaboratively [18], [19].
The proposed hypothesis was as follows.

The leader-follower principle will take on different strategies dependent on the current movement. Moving away- and towards a goal can be done at different speeds for the robots as this will not require the robots to have an exactly defined transformation between each other. During the scanning phase, this transformation is, however, very important. Therefore, for the moving around a goal trajectory, it will require the controller to take into account both robots and their transformation.

Again, the hypothesis is very similar to the research answer.
The only factor missing in the hypothesis is the distinction between conventional leader-follower algorithms and a leaderless algorithm.

These are quite similar algorithms, however, they require different scenarios or use cases. For that reason, a distinction between conventional leader-follower and leaderless is very useful.

7.2.3 Non-identical robots

What happens to the controllability and accuracy if the follower robot is not the same as the leader robot?

The control of non-identical robots, such as heavier followers or heterogeneous robot combinations was also investigated extensively. The research had shown that the biggest influence of a 'new' robot would be the joint torques and the workspace. The joint torques were solved by lowering the acceleration or altering the trajectory to be less single-joint-dependent. In terms of workspace, the workspace was analyzed using a manipulability analysis which helps the user determine the optimal robot base placement based on the desired global end effector position.
In addition to optimizing manipulability, the accuracy and performance are also optimized by keeping joint torques below their limits. This is solved by altering the maximum end effector acceleration. A systematic process is proposed which has proven to function well.
If a non-identical robot would be used, a difference in acceleration would exist between leader and follower robot. The controller is able to take these differences into account and could constrain or unconstrain these differences based on the chosen leader-follower algorithm.
The proposed hypothesis for this research was as follows.

If a correct control architecture is designed, the accuracy should not be influenced by the type of robot, or its environment, as long as the desired waypoints are in the robot's workspace and the independent joint accuracy is of the same order. A 'correct' control architecture would in this case consist of a controller which takes into account the difference in dynamics, the position and the velocity of both robots, when making its trajectory choices.

In terms of controllability and trajectory choices, the hypothesis is very similar to the proposed research answer.
In addition, the research answer also introduced the workspace differences between robots by optimizing their manipulability.
This workspace solution was not proposed in the hypothesis but has shown to be important in overcoming the workspace differences between robots.

7.2.4 Operating close to robot singularities.

What happens to the controllability and/or accuracy when the follower robots is tasked to operate close to its workspace boundary (i.e. singularity region).

In this report singularities were investigated. It was seen that singularities should not be considered as a single position but more as a zone [23]. This is a side effect of the inverse kinematics method chosen.
A method of resetting this inverse kinematics method is proposed, but is ill-advised because of its non-smooth nature.
Instead of trying to minimize the singularity zone, a better method of working with singularities is to completely avoid getting within close proximity of them [24], [25]. To ensure this, a manipulability optimization method is proposed which is able to find the optimal robot base position for a certain global end effector goal.
The hypothesis which was established for this question is as follows.

The follower robot will have difficulties controlling itself precisely when it is near its singularity. This will cause its tracking error to converge to zero slower. This in turn will cause the robot to perform its trajectory slower. Which, if designed for, will limit the speed of the leader-follower trajectory as well.

In this case, the hypothesis and research answer are quite different. The researched showed that, the follower moving near its singularity did not observe any difficulty up until reaching the inverse

kinematics problem. This disallowed the follower to even operate in very close proximity of its singularities. However, as said, a method of completely eliminating this singularity issue is proposed.

7.2.5 Autonomous navigation

To what extent are leader-follower systems able to implement autonomous navigation. Also, what sort of differences in terms of accuracy and performance can be observed between a pre-determined trajectory or a autonomously generated trajectory?.

This is the least researched or designed subquestion. The proposed autonomous navigation algorithm has not been produced. However, the current control architecture seems very capable of allowing this.
This is because of the amount of configurability. Currently, the configurability is given to the user to decide. However, this decision could just as easily be given to an autonomous algorithm.
This means that the algorithm would only have to consist of a 'brain' determining the desired position, velocity, acceleration, leader-follower algorithm etc. and the controller itself would be able to control the robots based of those preferences.
In addition, the proposed manipulability optimization will allow the robot to quickly manipulate itself, if necessary, to avoid a collision.
Comparing this to the proposed hypothesis.

As this mapping of the environment and collision avoidance will increase the complexity of the control system, it can be assumed that the performance of the robots will decrease whenever they are tasked to autonomously navigate towards the desired pose, as opposed to simply following a predetermined trajectory.

It is hard to say whether the expected hypothesis is also the actual result since no autonomous navigation algorithm was designed. However, as said, the confidence in this controller's ability to be adaptable to an uncertain trajectory is high.

8 Recommendations

Looking at the proposed results, everything is derived from a simulator. Such a simulation does not take into account external disturbances, sensor noise etc. It is therefore recommended to start incorporating this control architecture onto more complex simulators or even the real Franka Emika Panda robot. This connection between the Simulink architecture and the real hardware has been briefly mentioned in this report. However, a more detailed method of this connection is explored in another BEP project. This project is therefore also highly recommended in combination with the proposed control architecture.

Looking at the trajectory generation.
The waypoint-to-waypoint trajectory generator may be increased to a higher order polynomial function which will allow for a continuous jerk signal. This will help smooth out the trajectory even further and eliminate unnecessary accuracy losses.

Looking at configurability.
The current configurable controller is based of user-inputted values such as the maximum allowed end effector acceleration. This acceleration must be determined by the user via methods explained in this report. This is still a manual method since it relies heavily on the robots physical attributes such as mass, dimension, torque limits. Further, it also depends on the tasked trajectory.
It was found that an acceleration of $10m/s^2$ was a safe choice in most cases. However, some easier trajectories might have been possible with even higher accelerations.
One might explore the possibility of letting the controller automatically determine the maximum allowed acceleration. This will prove challenging because of the many influences but will be very beneficial for optimal performance.

Another interesting challenge which might prove useful is the combination of leader-follower algorithms.
As it stands, the user is able to choose a trajectory and a leader-follower algorithm associated with that trajectory. However, it might be interesting to have trajectory where the leader-follower algorithm is not necessarily constant.
For example, a complete trajectory could be made where first the robots independently autonomously navigate towards a patient's broken arm. After having arrived at this location, the robots can then collaboratively start the x-ray scan by moving around the broken arm.
In this trajectory it would be beneficial to have a leaderless algorithm for the first part of the trajectory and a perfect transformation algorithm for the second part of the trajectory.

Although, shortly introduced, the current control architecture is not capable of autonomously navigating the robots throughout an environment.
Whilst keeping the medical use case in mind it is easy to understand the importance of autonomous navigation due to the dynamical nature of working in proximity of humans.
So it is highly recommended to incorporate this navigation method into the control architecture to make sure collisions are avoided.

References

[1] O. O. Obadina, M. A. Thaha, Z. Mohamed, and M. H. Shaheed, "Grey-box modelling and fuzzy logic control of a Leader–Follower robot manipulator system: A hybrid Grey Wolf–Whale Optimisation approach," *ISA Transactions*, no. xxxx, 2022. [Online]. Available: https://doi.org/10.1016/j.isatra.2022.02.023

[2] S. C. Liu, D. L. Tan, and G. J. Liu, "Robust leader-follower formation control of mobile robots based on a second order kinematics model," *Zidonghua Xuebao/Acta Automatica Sinica*, vol. 33, no. 9, pp. 947–955, 2007.

[3] Q. Lu, Z. Miao, D. Zhang, L. Yu, W. Ye, S. X. Yang, and C. Y. Su, "Distributed leader-follower formation control of nonholonomic mobile robots," *IFAC-PapersOnLine*, vol. 52, no. 15, pp. 67–72, 2019.

[4] O. Barnes, "Robots give surgeons a helping hand," 01 2022. [Online]. Available: https://www.ft.com/content/2c47aaba-29e3-4f6a-b1ef-6037fa68513d

[5] B. Hu, "Self-triggering in Vehicular Networked Systems with State-dependent Bursty Fading Channels," pp. 16–17, 8 2017. [Online]. Available: http://arxiv.org/abs/1708.02347

[6] WiredWorkers, "Franka Emika Panda Collaborative Lightweight Robot," 03 2022. [Online]. Available: https://wiredworkers.io/cobot-brands/cobot-franka-emika-panda/

[7] D. Shi, J. Zhang, Z. Sun, G. Shen, and Y. Xia, "Composite trajectory tracking control for robot manipulator with active disturbance rejection," *Control Engineering Practice*, vol. 106, no. November 2020, p. 104670, 2021. [Online]. Available: https://doi.org/10.1016/j.conengprac.2020.104670

[8] J. Moreno-Valenzuela, "Time-scaling of trajectories for point-to-point robotic tasks," *ISA Transactions*, vol. 45, no. 3, pp. 407–418, 2006.

[9] A. Tika, N. Gafur, V. Yfantis, and N. Bajcinca, "Optimal scheduling and model predictive control for trajectory planning of cooperative robot manipulators," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9080–9086, 2020. [Online]. Available: https://doi.org/10.1016/j.ifacol.2020.12.2136

[10] X. Liu, C. Qiu, Q. Zeng, A. Li, and N. Xie, "Time-energy Optimal Trajectory Planning for Collaborative Welding Robot with Multiple Manipulators," *Procedia Manufacturing*, vol. 43, pp. 527–534, 2020. [Online]. Available: https://doi.org/10.1016/j.promfg.2020.02.174

[11] B. Olofsson and L. Nielsen, "Path-tracking velocity control for robot manipulators with actuator constraints," *Mechatronics*, vol. 45, pp. 82–99, 2017.

[12] H. Wang, Q. Zhao, H. Li, and R. Zhao, "Polynomial-based smooth trajectory planning for fruit-picking robot manipulator," *Information Processing in Agriculture*, vol. 9, no. 1, pp. 112–122, 2021. [Online]. Available: https://doi.org/10.1016/j.inpa.2021.08.001

[13] L. Gouzenes, "Generation of Collision-free Trajectories for Mobile and Manipulator Robots," *IFAC Proceedings Volumes*, vol. 16, no. 20, pp. 265–272, 1983. [Online]. Available: http://dx.doi.org/10.1016/S1474-6670(17)61615-X

[14] G. Carabin and R. Vidoni, "Energy-saving optimization method for point-to-point trajectories planned via standard primitives in 1-DoF mechatronic systems," *International Journal of Advanced Manufacturing Technology*, vol. 116, no. 1-2, pp. 331–344, 2021.

[15] H. Wang, H. Wang, J. Huang, B. Zhao, and L. Quan, "Smooth point-to-point trajectory planning for industrial robots with kinematical constraints based on high-order polynomial curve," *Mechanism and Machine Theory*, vol. 139, pp. 284–293, 2019. [Online]. Available: https://doi.org/10.1016/j.mechmachtheory.2019.05.002

[16] J. Long, Y. Tian, W. Chen, J. Leng, and Y. Wang, "Locating, trajectory planning and control of an underwater propeller cleaning manipulator," *Ocean Engineering*, vol. 243, no. November 2021, p. 110262, 2022. [Online]. Available: https://doi.org/10.1016/j.oceaneng.2021.110262

[17] X. Zhang and J. Sun, "Almost equitable partitions and controllability of leader–follower multi-agent systems," *Automatica*, vol. 131, p. 109740, 2021. [Online]. Available: https://doi.org/10.1016/j.automatica.2021.109740

[18] C. c. Meng and X. y. Zhang, "Distributed leaderless formation control for multiple autonomous underwater vehicles based on adaptive nonsingular terminal sliding mode," *Applied Ocean Research*, vol. 115, no. June, p. 102781, 2021. [Online]. Available: https://doi.org/10.1016/j.apor.2021.102781

[19] J. Huang, W. Wang, C. Wen, J. Zhou, and G. Li, "Distributed adaptive leader–follower and leaderless consensus control of a class of strict-feedback nonlinear systems: a unified approach," *Automatica*, vol. 118, p. 109021, 2020. [Online]. Available: https://doi.org/10.1016/j.automatica.2020.109021

[20] C. I. Aldana, E. Nuño, and L. Basañez, "Leader-follower pose consensus for heterogeneous robot networks with variable time-delays," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 19, pp. 6674–6679, 2014.

[21] B. Wang, H. Ashrafiuon, and S. Nersesov, "Leader–follower formation stabilization and tracking control for heterogeneous planar underactuated vehicle networks," *Systems and Control Letters*, vol. 156, p. 105008, 2021. [Online]. Available: https://doi.org/10.1016/j.sysconle.2021.105008

[22] L. Cheng, Z. G. Hou, and M. Tan, "Decentralized adaptive leader-follower control of multi-manipulator system with uncertain dynamics," *IECON Proceedings (Industrial Electronics Conference)*, pp. 1608–1613, 2008.

[23] P. P. Rebouças Filho, S. P. Suane, V. N. Praxedes, J. Hemanth, and V. H. C. de Albuquerque, "Control of singularity trajectory tracking for robotic manipulator by genetic algorithms," *Journal of Computational Science*, vol. 30, pp. 55–64, 2019.

[24] Q. Liu, W. Tian, B. Li, and Y. Ma, "Kinematics of a 5-axis hybrid robot near singular configurations," *Robotics and Computer-Integrated Manufacturing*, vol. 75, no. November 2021, p. 102294, 2022. [Online]. Available: https://doi.org/10.1016/j.rcim.2021.102294

[25] P. P. Rebouças Filho, S. P. Suane, V. N. Praxedes, J. Hemanth, and V. H. C. de Albuquerque, "Control of singularity trajectory tracking for robotic manipulator by genetic algorithms," *Journal of Computational Science*, vol. 30, pp. 55–64, 2019.

[26] H. D. Bui, H. Nguyen, H. M. La, and S. Li, "A Deep Learning-Based Autonomous Robot Manipulator for Sorting Application," *Proceedings - 4th IEEE International Conference on Robotic Computing, IRC 2020*, pp. 298–305, 2020.

[27] L. Da Silva Assis, A. Da Silva Soares, C. J. Coelho, and J. Van Baalen, "An evolutionary algorithm for autonomous robot navigation," *Procedia Computer Science*, vol. 80, pp. 2261–2265, 2016.

[28] A. Perrusquía, R. Garrido, and W. Yu, "Stable robot manipulator parameter identification: A closed-loop input error approach," *Automatica*, vol. 141, p. 110294, 2022. [Online]. Available: https://doi.org/10.1016/j.automatica.2022.110294

[29] B. Emara, M. B. Emara, A. W. Y. Eman, P. Stief, J.-y. Dantan, A. Etienne, and A. Siadat, "ScienceDirect ScienceDirect Digital Twinning for Closed-Loop Control of a Three-Wheeled Digital Twinning for Closed-Loop Control of a Three-Wheeled Omnidirectional Robot Omnidirectional Mobile Robot new methodology to analyze the functional and physical a," *Procedia CIRP*, vol. 107, no. 2021, pp. 1245–1250, 2022. [Online]. Available: https://doi.org/10.1016/j.procir.2022.05.139

[30] Mathworks, "URDF Primer - MATLAB Simulink." [Online]. Available: https://www.mathworks.com/help/physmod/sm/ug/urdf-model-import.html#:%7E:text=URDF%2C%20or%20Unified%20Robotics%20Description,animatronic%20robots%20for%20amusement%20parks.

[31] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Hoboken, NJ, United States: Wiley, 2020.

[32] G. Chen, L. Zhang, Q. Jia, and H. Sun, "Singularity analysis of the redundant robot with the structure of three consecutive parallel axes," *Advances in Intelligent Systems and Computing*, vol. 213, pp. 779–791, 2014.

[33] N. Vahrenkamp, T. Asfour, G. Metta, G. Sandini, and R. Dillmann, "Manipulability analysis," *IEEE-RAS International Conference on Humanoid Robots*, no. 3, pp. 568–573, 2012.

[34] H. Kawashima and M. Egerstedt, "Manipulability of leader-follower networks with the rigid-link approximation," *Automatica*, vol. 50, no. 3, pp. 695–706, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.automatica.2013.11.041

[35] N. M. Ben Lakhal, L. Adouane, O. Nasri, and J. Ben Hadj Slama, "Safe and adaptive autonomous navigation under uncertainty based on sequential waypoints and reachability analysis," *Robotics and Autonomous Systems*, vol. 152, p. 104065, 2022. [Online]. Available: https://doi.org/10.1016/j.robot.2022.104065

[36] J. Zhang, H. Zhang, S. Sun, and Z. Gao, "Leader-follower consensus control for linear multi-agent systems by fully distributed edge-event-triggered adaptive strategies," *Information Sciences*, vol. 555, pp. 314–338, 2021. [Online]. Available: https://doi.org/10.1016/j.ins.2020.10.056

[37] B. C. Min, R. Parasuraman, S. Lee, J. W. Jung, and E. T. Matson, "A directional antenna based leader–follower relay system for end-to-end robot communications," *Robotics and Autonomous Systems*, vol. 101, pp. 57–73, 2018. [Online]. Available: https://doi.org/10.1016/j.robot.2017.11.013

[38] X. Li, X. Gao, W. Zhang, and L. Hao, "Smooth and collision-free trajectory generation in cluttered environments using cubic B-spline form," *Mechanism and Machine Theory*, vol. 169, no. October 2021, p. 104606, 2022. [Online]. Available: https://doi.org/10.1016/j.mechmachtheory.2021.104606

[39] H. Wang, Q. Zhao, H. Li, and R. Zhao, "Polynomial-based smooth trajectory planning for fruit-picking robot manipulator," *Information Processing in Agriculture*, vol. 9, no. 1, pp. 112–122, 2022. [Online]. Available: https://doi.org/10.1016/j.inpa.2021.08.001

[40] B. Gravell and T. Summers, "Centralized collision-free polynomial trajectories and goal assignment for aerial swarms," *Control Engineering Practice*, vol. 109, no. February, p. 104753, 2021. [Online]. Available: https://doi.org/10.1016/j.conengprac.2021.104753

[41] F. Causa and G. Fasano, "Multiple UAVs trajectory generation and waypoint assignment in urban environment based on DOP maps," *Aerospace Science and Technology*, vol. 110, p. 106507, 2021. [Online]. Available: https://doi.org/10.1016/j.ast.2021.106507

[42] C. Lauretti, T. Grasso, E. D. Marchi, and S. Grazioso, "A Geometric Approach to Inverse Kinematics of Hyper-Redundant Manipulators for tokamaks maintenance," *Mechanism and Machine Theory*, vol. 176, no. June, p. 104967, 2022. [Online]. Available: https://doi.org/10.1016/j.mechmachtheory.2022.104967

[43] "Joint accelerations given joint torques and states - Simulink." [Online]. Available: https://www.mathworks.com/help/robotics/ref/forwarddynamics.html

[44] "Simscape Multibody." [Online]. Available: https://www.mathworks.com/products/simscape-multibody.html

[45] "Gazebo." [Online]. Available: https://gazebosim.org/features

[46] "Discover our Gen3 robots," 01 2022. [Online]. Available: https://www.kinovarobotics.com/product/gen3-robots

[47] A. M. Bader and A. A. Maciejewski, "The kinematic design of redundant robots for maximizing failure-tolerant workspace size," *Mechanism and Machine Theory*, vol. 173, no. December 2021, p. 104850, 2022. [Online]. Available: https://doi.org/10.1016/j.mechmachtheory.2022.104850

[48] "Generate polynomial trajectories using B-splines - MATLAB bsplinepolytraj." [Online]. Available: https://www.mathworks.com/help/robotics/ref/bsplinepolytraj.html

[49] U. Robots, "Universal Robots - Max. joint torques." [Online]. Available: https://www.universal-robots.com/articles/ur/robot-care-maintenance/max-joint-torques/

Appendix A Figures

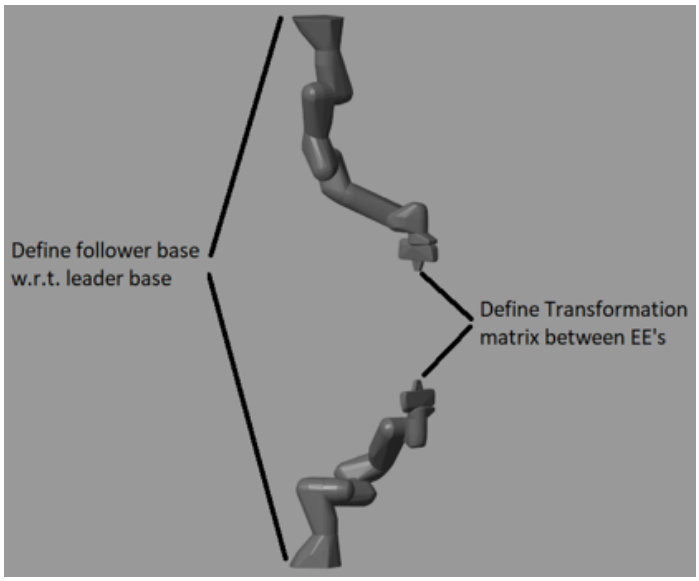


Figure A.1: The user defined positioning of robot bases and end effectors with respect to each other

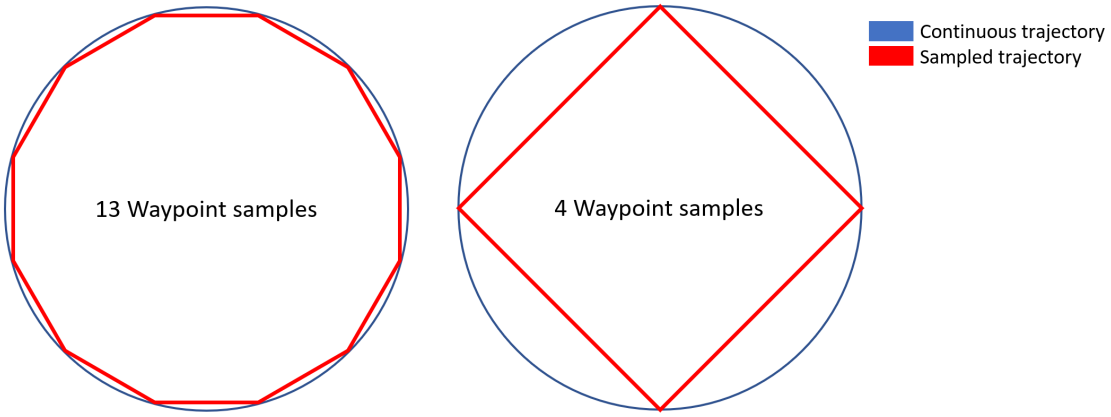


Figure A.2: An exaggerated waypoint sample rate difference on a 'complex' trajectory

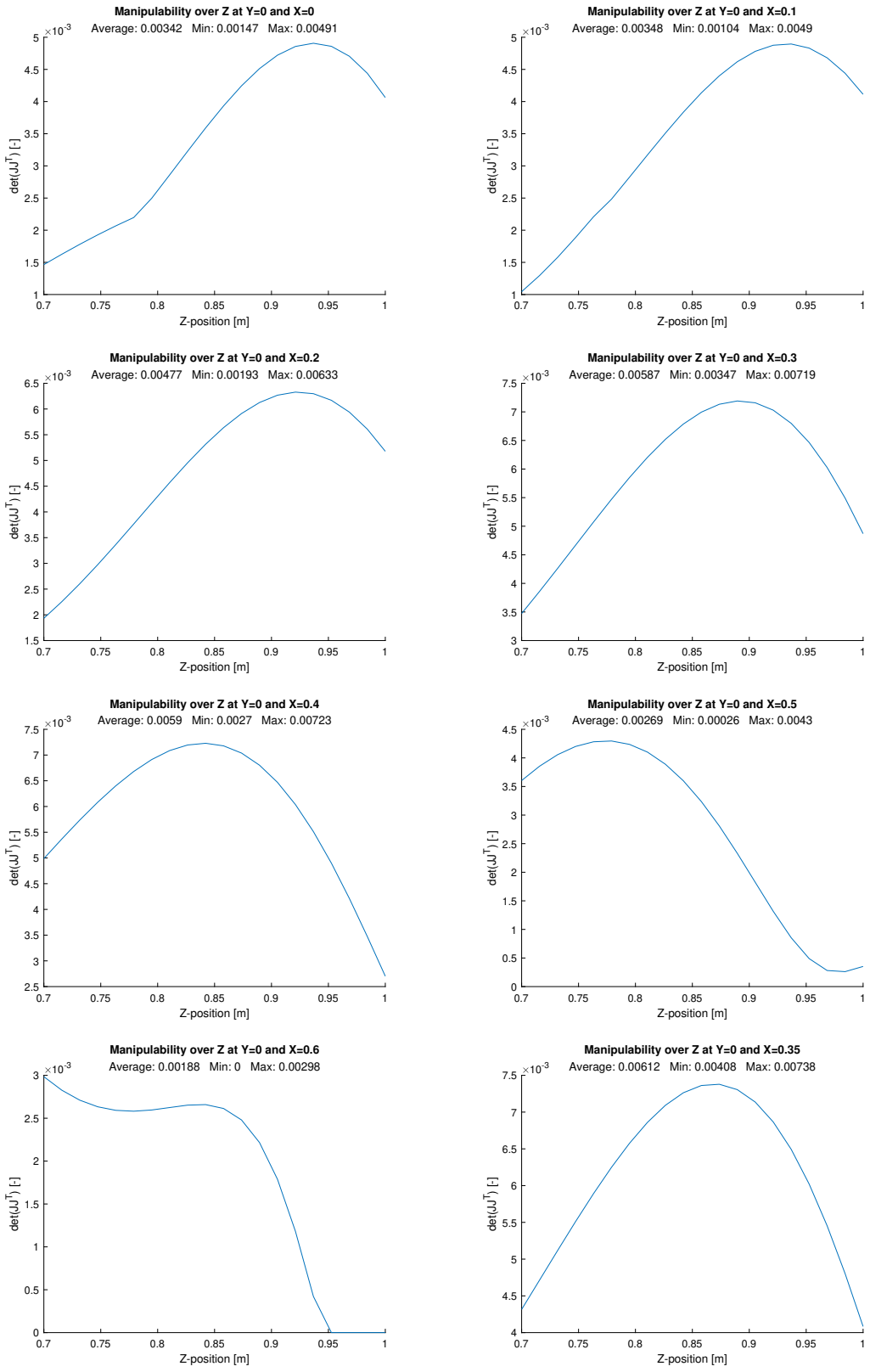


Figure A.3: Manipulability analysis results for a straight upwards trajectory. Optimized for x-coordinate. Best results found at 0.3[m] and 0.4[m]. Lastly, intermediate value 0.35[m] tested and turned out to be the optimal x-position for this trajectory.

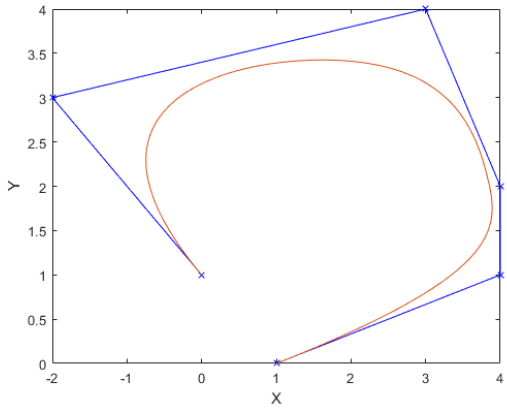


Figure A.4: An example of B-spline generated trajectory.[48]

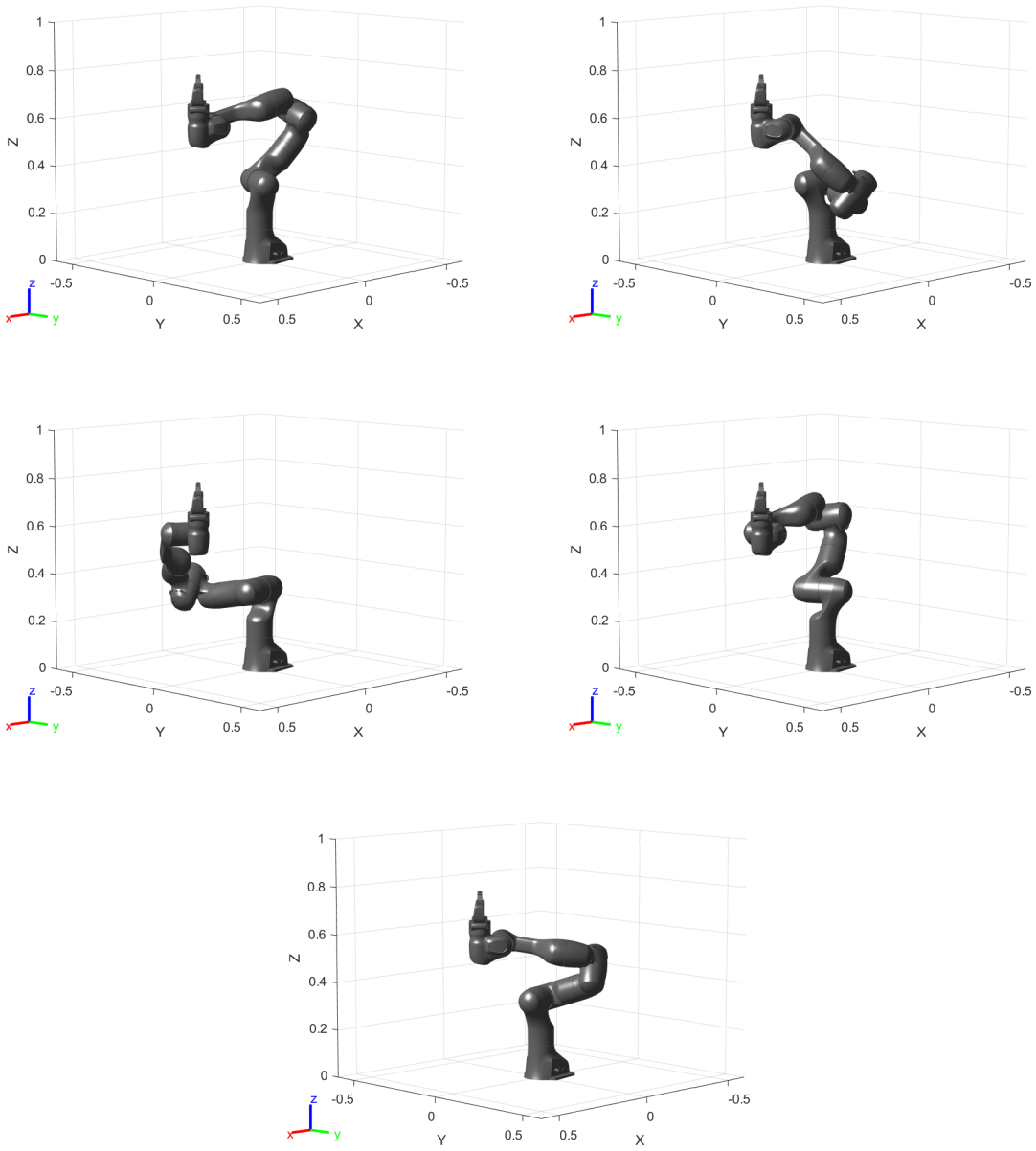


Figure A.5: Five, clearly different, inverse kinematics solution which satisfy the same end effector position. (There are even more, less noticeable, different solutions)

```
1  function joint_velocity = fcn(end_effector_velocity, jacobian)
2
3  x = pinv(jacobian);
4  joint_velocity = x*end_effector_velocity;
```

Figure A.6: The computation method used to calculate the joint velocities based on the computed Jacobian and end effector velocity.

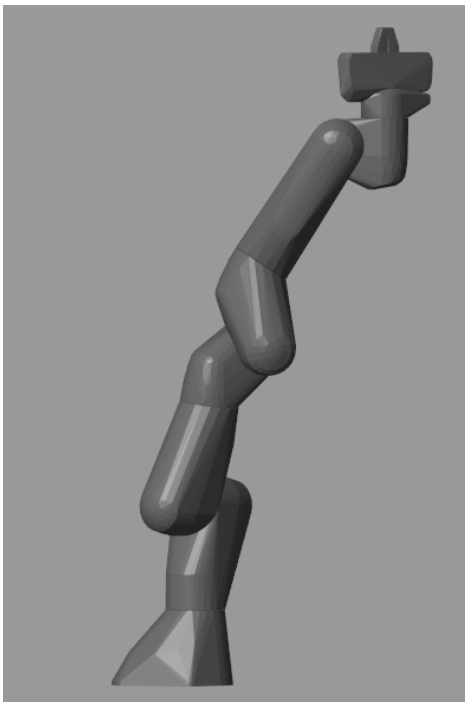


Figure A.7: The singular configuration with the robot arm fully extended.

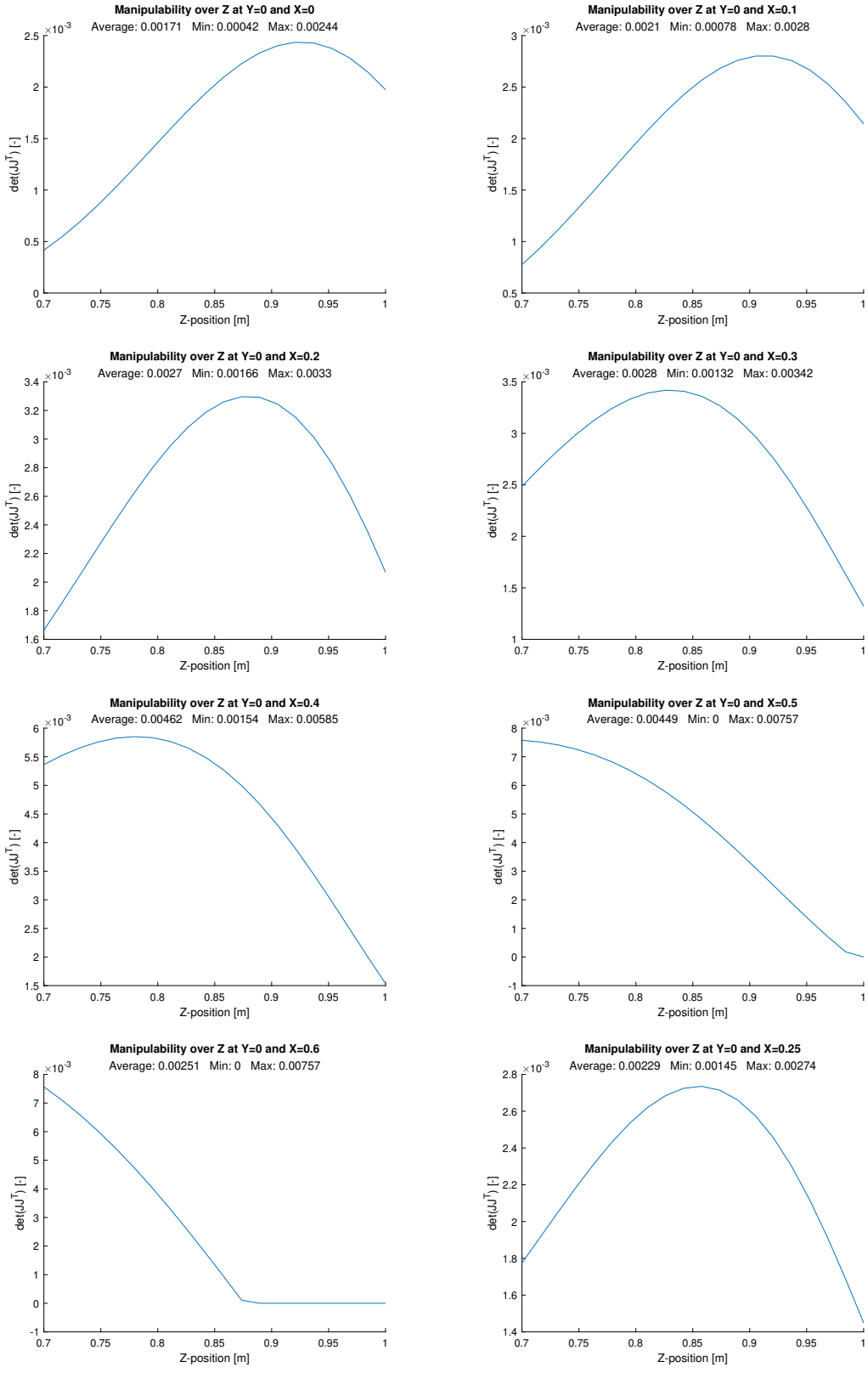


Figure A.8: Manipulability analysis results for Kinova Gen3 robot.
Optimized for a straight upwards trajectory. Optimized for x-coordinate. Best results found at 0.2[m] and 0.3[m], since they have the highest minimum manipulability. Lastly, intermediate value 0.25[m] tested and turned out to have a lower minimum as 0.2[m].

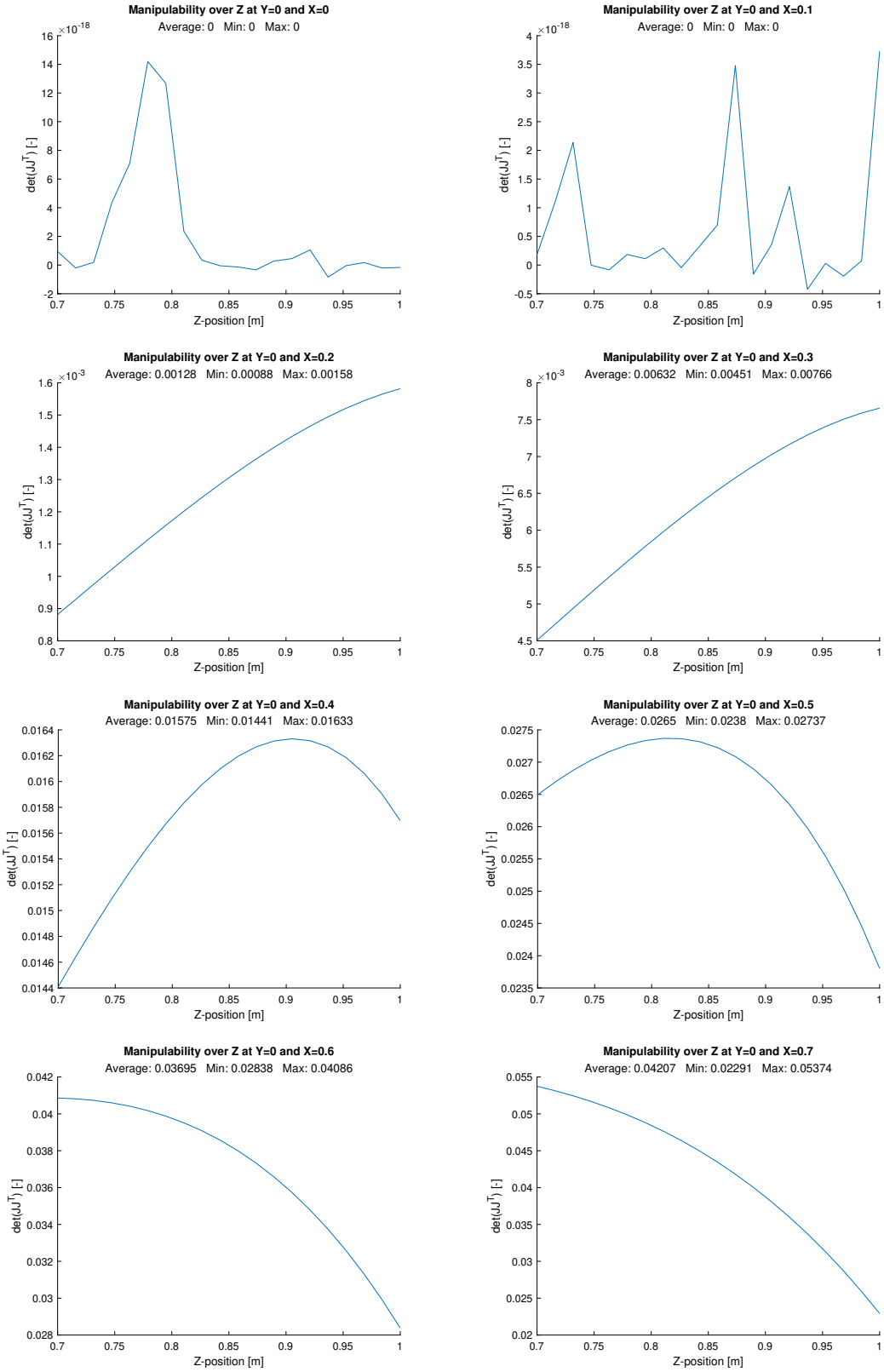


Figure A.9: First part of the universal UR10's manipulability analysis. Rest of x-coordinates on the next page.

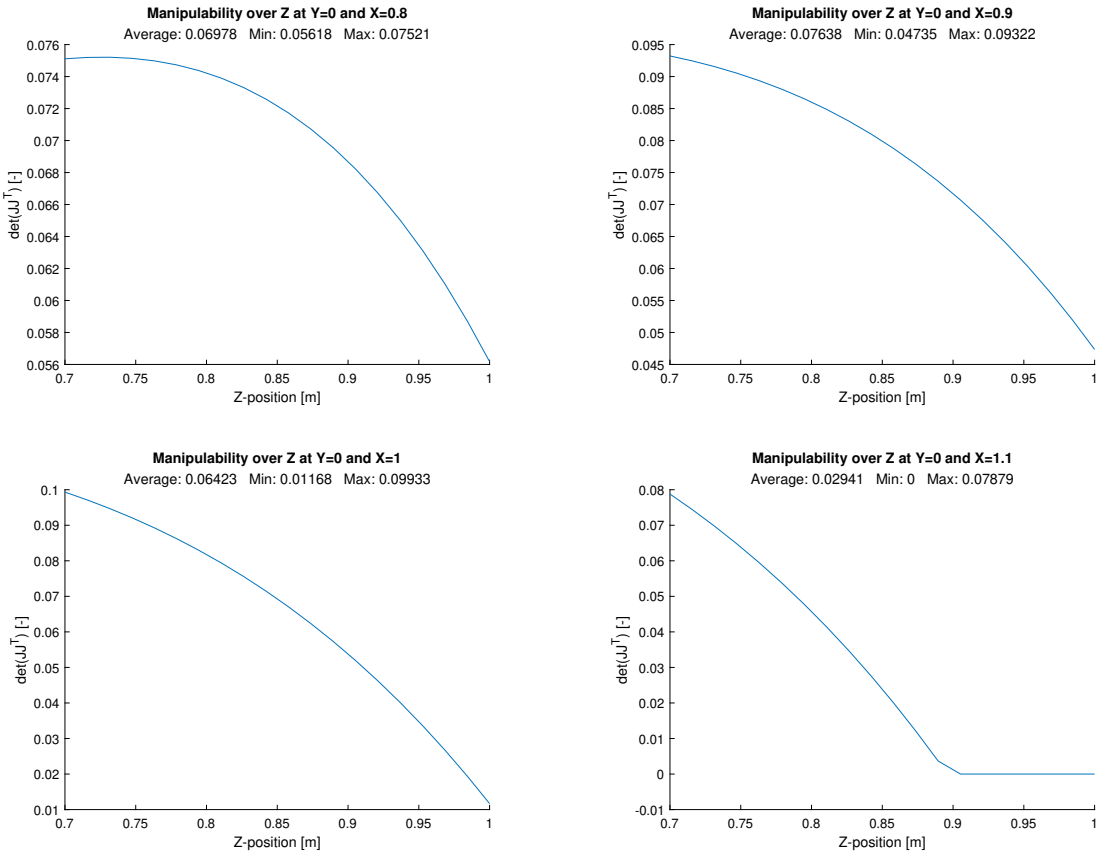
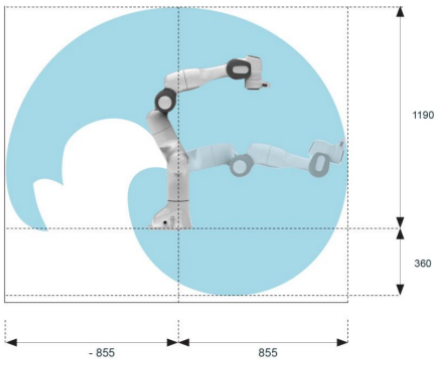
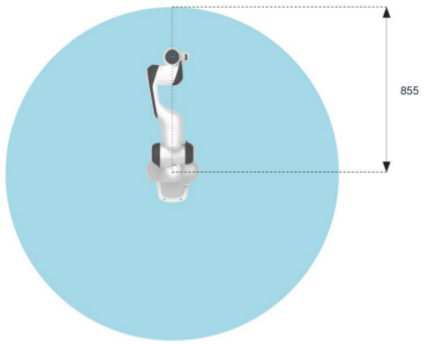


Figure A.10: Second part of the universal UR10's manipulability analysis. The most optimal x-coordinate turned out to be 0.9[m]. Which is much different from the Franka robot.

Appendix B Robot data



1. Arm workspace side view



2. Arm workspace top view

Figure B.1: The theoretical workspace of a 7DOF Franka robot [6].

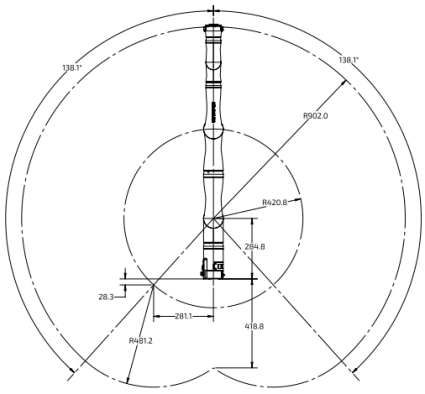


Figure B.2: The theoretical workspace of a 7DOF Kinova Gen3 robot [46].

Torque limit	Franka	Kinova Gen3	Universal UR10
Joint 1	±87	±54	±330
Joint 2	±87	±54	±330
Joint 3	±87	±54	±150
Joint 4	±87	±54	±56
Joint 5	±12	±34	±56
Joint 6	±12	±34	±56
Joint 7	±12	±34	N.A.

Table B.1: Table including all joint torque limits of the robots discussed in this report.[6],[46],[49]