ME41020
Space Robotics
Group 17

Roel Djajadiningrat, 4012739
Martin Kooper, 4208641
Jeffrey Minnaard, 4122399
Nick Tsutsunava, 4085396

April 26, 2017

Instructor Dr. A. Schiele

Contents

1 Introduction

The goal of this project, broadly speaking, is to model an industrial robot manipulator and its movement over a specified path. The main focus in the execution of this task lies on the robot's constraints. Over the past few weeks a functioning model which fulfills these requirements was created with Matlab, Simulink and SimMechanics. Additional features have been added as well.

In this report the parametrization of an industrial 6-degrees-of-freedom robotic manipulator, planning of its trajectory and its actual movement through all quadrants with time, position and velocity constraints are discussed. Additionally, a Graphical User Interface is presented allowing for an interactive input of target points and the possibility to select multiple generic solutions corresponding to the manipulator's physical configuration at different speeds with different degrees of motion easing.

All of the code, including CAD model and Simulink files, can be found in our open source GitHub repository[1] for further review and experimentation. The `README.MD` contains a list of interesting files highlighting important aspects of the system.

2 KUKA KR10

The manipulator that has been chosen for this project is the KUKA KR10-R1100-sixx-CR. The manipulator, manufactured by KUKA, is a compact six-axis robot that is designed for particularly high working speeds. It can be mounted on the floor, ceiling or wall and the compact dimensions of the manipulator allow for high precision operations in confined spaces.
This manipulator was chosen for the project as it excels in speed and manoeuvrability, which are two of the key interests in this project. The robot can be set into multiple configurations to reach the same end-effector position and is therefore interesting to model.
In this chapter some basic information[2] concerning the manipulator is presented. The manipulator's basic properties can be seen in Table 1.

Table 1: Basic Data KUKA KR10-R1100-sixx-CR

Number of controlled axes	6
Volume of working envelope	5.14 m^3
Weight	5 kg
Rated payload	5 kg
Maximum total load	10 kg
Maximum reach	1101 mm
Footprint	320 x 320 mm

[1] `https://github.com/Ortix92/space-robotics/`
[2] KR AGILUS sixx HM-SC Specification, Version: Spez KR AGILUS sixx HM-SC V2 (2016)

Table 2 and 3 show the motion range and the maximum speed with rated payload of the robot axis. The corresponding defined directions are shown in Figure 1 and 2.

Table 2: Axes motion range

Axis	Motion range
1	±166°
2	−188°/45°
3	−116°/153°
4	±185°
5	±110°
6	±350°

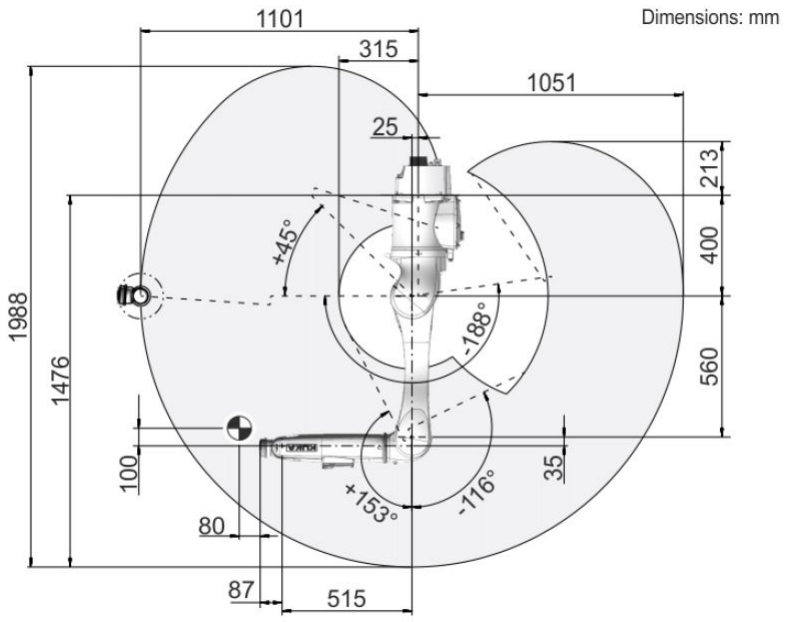

Figure 1: Working envelope[2]

Table 3: Speed with rated payload

Axis	Speed with rated payload
1	$300°/s$
2	$225°/s$
3	$225°/s$
4	$381°/s$
5	$262°/s$
6	$414°/s$

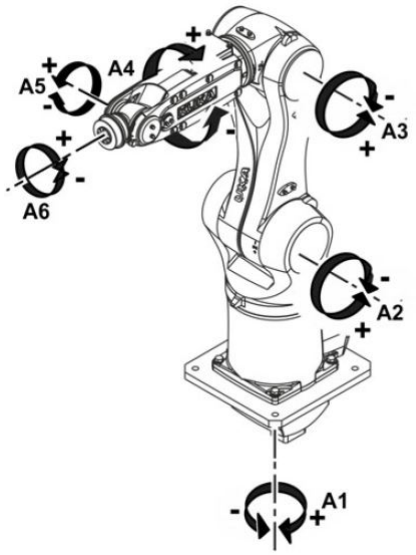

Figure 2: Axes direction of rotation[2]

The joint speeds and motion range can be used to eventually check the validity of the model. As the robot properties are known, the model can be created. The first step is the parametrization of the manipulator.

3 Denavit-Hartenberg Parametrization

The manipulator that is to be parametrized is a 6-DOF manipulator as shown in Figure 3.

In order to eventually obtain joint angles as a result of a Cartesian position input of the end effector through inverse kinematics, parameters have to be described that allow for computations with direct kinematics. The computation of the direct kinematics is recursive and is obtained by products of homogeneous transformation matrices. These A-matrices describe the rotation and the position of one link of the manipulator with respect to the previous one. It is convenient to express this transformation with standardized parameters. The Denavit-Hartenberg parameters are used to this end.

The four Denavit-Hartenberg parameters are:

- a_i: the length of link i
- α_i: the twist of link i
- d_i: the offset of link i
- θ_i: the angle of joint i

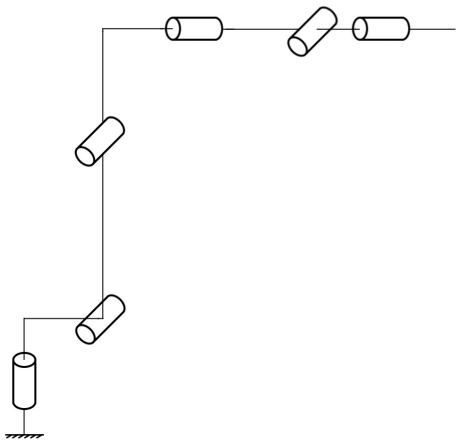

Figure 3: 6-DOF manipulator

3.1 DH-convention

The Denavit-Hartenberg or DH-parameters are obtained through the DH-convention, a standardized way of setting up the parameters used in the A-matrices. Firstly, the different joint axis systems of the manipulator are to be set up. The longitudinal axes of the joints are labeled joint axes $z_0 ... z_{n-1}$. The base frame with x_0 and y_0 is then constructed through a right-hand frame.
Subsequently, for each link, excluding the end-effector, the origin o where the common normal to z and the previous z intersects z. If these two z's intersect the origin is to be located here. If they are parallel, the origin is to be located at the joint center.

Next, x is established along the common normal between the previous link's z and the current z through the origin. If the current and previous z intersect, it x is to be established in the direction normal to the plane of the current and previous z.

Lastly y is added to the frame according to the right-hand rule.
For the end-effector frame, z is alligned with the previous z when the last joint is revolute. x and y are added to the frame as before.

When the DH-convention is applied to the 6-DOF manipulator of Figure 3 the

frame system in Figure 4 is obtained.

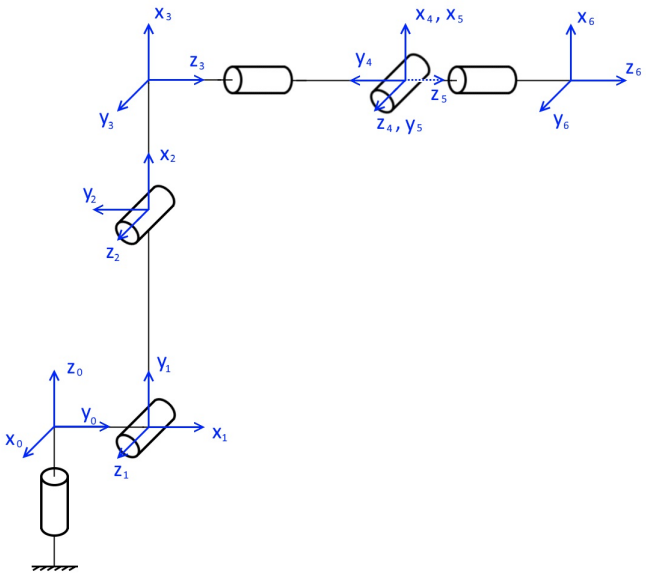

Figure 4: 6-DOF manipulator with joint frames

Now that the joint frames are known, the four DH-parameters for each link can be determined.

For link $(i = 1, ..., n)$

- a_i: the distance along x_i from the intersection of the x_i and z_{i-n} axes to origin o.
- α_i: The angle required to rotate z_{i-1} in alignment with z, measured about the positive x_i direction.
- d_i: The distance along z_{i-1} from origin o_{i-1} to the intersection of the x_i and z_{i-1} axes.
- θ_i: the angle required to rotate x_{i-1} in allignment with x_i measured about the positive z_{i-1} direction.

For the 6-DOF manipulator the link parameters are shown in Figure 5

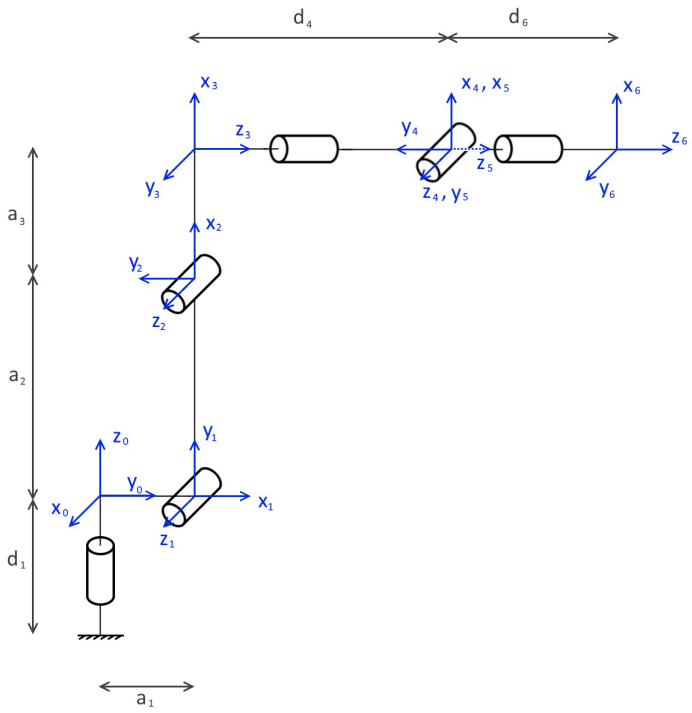

Figure 5: 6-DOF manipulator with joint frames and link parameters

3.2 DH-Parameters

For the KUKA KR-10 the zero-position is selected as shown in Figure 6.

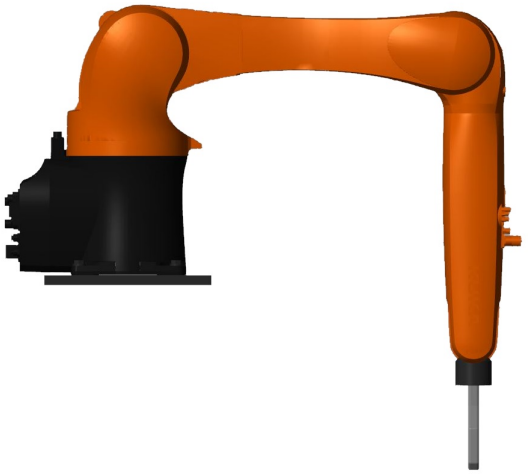

Figure 6: KUKA KR10

Following the DH-convection described above and taking into account the manipulator dimensions as mentioned in the KUKA KR10 specifications the following DH-parameters are obtained:

Table 4: KUKA KR10 DH-parameters

	θ	d	a	α
1	θ_1	0.4	0.025	$\frac{\pi}{2}$
2	θ_2	0	0.560	$\frac{\pi}{2}$
3	θ_3	0	0.035	$-\frac{\pi}{2}$
4	θ_4	0.515	0	$\frac{\pi}{2}$
5	θ_5	0	0	$-\frac{\pi}{2}$
6	θ_6	0	0	0

These parameters will be used to construct the forward kinematics.

4 Forward Kinematics

Forward kinematics is the principle of using the joint angles in order to calculate the position of the end effector. Compared to the inverse kinematics, this is easy since it requires only straightforward matrix multiplication.

In order to find the position of the end effector we need to get the individual transformation matrices from the base frame to each joint. With the DH parameters we can derive the first transformation matrix from the base to the first joint as follows.

$$T_1^0 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1\cos\alpha_1 & \sin\theta_1\sin\alpha_1 & a_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1\cos\alpha_1 & -\cos\theta_1\sin\alpha_1 & a_1\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 & d_1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

We can follow this procedure for all joints which will give us

$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5$$
$$T_{\text{tool}}^0 = T_6^0 \cdot T_{\text{tool}}$$

where T_{tool} is a homogeneous translation matrix for the toolhead.

This can be easily implemented in MATLAB with a for loop over all the joints.

Furthermore, our forward kinematics function also returns the *orientation* vector, used for orientating the end effector to the target, based on the full transformation matrix T_{tool}^0. The orientation vector is merely vector representation of the Euler angles of the rotation matrices used as shown below.

$$T_{\text{orientation}} = R_x(\psi_1) \cdot R_y(\psi_2) \cdot R_z(\psi_3)$$

We are interested in the angles ψ_1, ψ_2 and ψ_3. We can extract the top left 3×3 block from T_{tool}^0 to get the transformation matrix and plug that into `rotationMatrixToVector()`, which is a built in MATLAB function which returns a vector of the Euler angles.

5 Inverse Kinematics

In order to plan end-effector motions by positioning the robot end-effector to certain coordinates in Cartesian space the inverse kinematics of the KUKA KR-10 are analyzed. Eventually, the joint angles that correspond to the end-effector Cartesian coordinates are to be found. This allows for end-effector motion planning by input of joint actuator trajectories. The decoupling[3,4,5] method is used to this end.

[3]Khatamian, A. “Solving Kinematics Problems of a 6-DOF Robot Manipulator.” (2015)
[4]Sinha, G. “Inverse Kinematics and path planning simulation for Robotic arm in MATLAB"
[5]Spong, M. “Robot Dynamics and Control" (2004)

As shown in the previous chapter the end-effector or tool head can be described through a multiplication of the translation matrices:

$$T_6^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5$$
$$T_{\text{tool}}^0 = T_6^0 \cdot T_{\text{tool}}$$

The inverse kinematics problem can be decoupled into two sub-problems: the inverse position and the inverse orientation kinematics.

In this chapter, the first three joint angles will be determined corresponding to a given position of the wrist origin. This is done through a geometric approach. To make sure the computation will be as fast as possible, Pythagoras' theorem will not be used to calculate lengths. Then, the values of the final three joint variables corresponding to a given orientation with respect to the frame of the last calculated angle are to be found with an inverse orientation approach. This last problem can be seen as one based on a spherical wrist, where a set of Euler angles is to be found corresponding to a given rotation matrix R or old-fashioned geometrical analysis can be performed.

5.1 Inverse position

First, calculations according to the inverse position are performed.

5.1.1 Joint 1

Figure 7 shows a diagram of a position vector and the first joint angle θ_1. Calculation of the first joint angle according to Figure 7 yields:

$$r = \sqrt{P_x^2 + P_y^2}$$
$$\theta_1 = \text{atan2}(P_y, P_x) \quad \text{or}$$
$$\theta_1 = \text{atan2}(P_y, P_x) + \pi$$

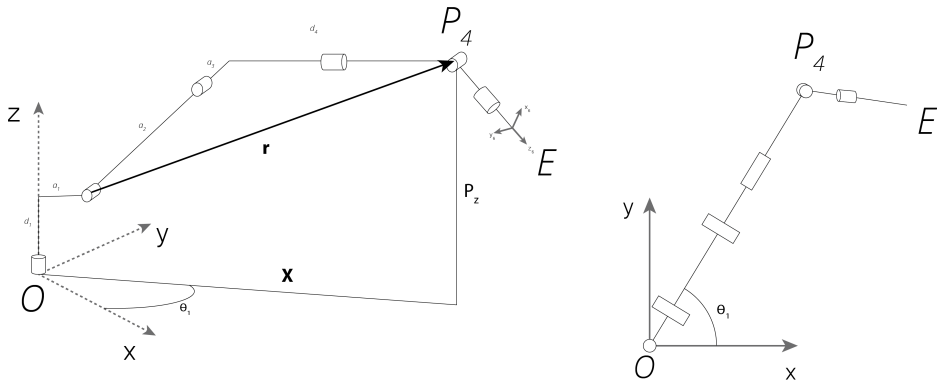

Figure 7: Calculation of the first joint angle

5.1.2 Joint 2

Now that the angle of joint 1 is known, with Figure 8 we can solve for the second joint. The vector X is constructed in the xy-plane:

$$X = P_x \cdot \cos(\theta_1) + \sin(\theta_1 - a_1)$$
$$r = \sqrt{X^2 + (P_z - d_1)^2}$$
$$\phi = \text{acos}\frac{a_2^2 - d_4^2 - a_3^2 + X^2 + (P_z - d_1)^2}{2 \cdot a_2 \cdot r}$$
$$\theta_2 = \text{atan}(P_z - d_1, X) + n_2 \cdot \phi$$

Here, θ_2 is constructed with n_2, which is a configuration setting which applies to the elbow configuration.

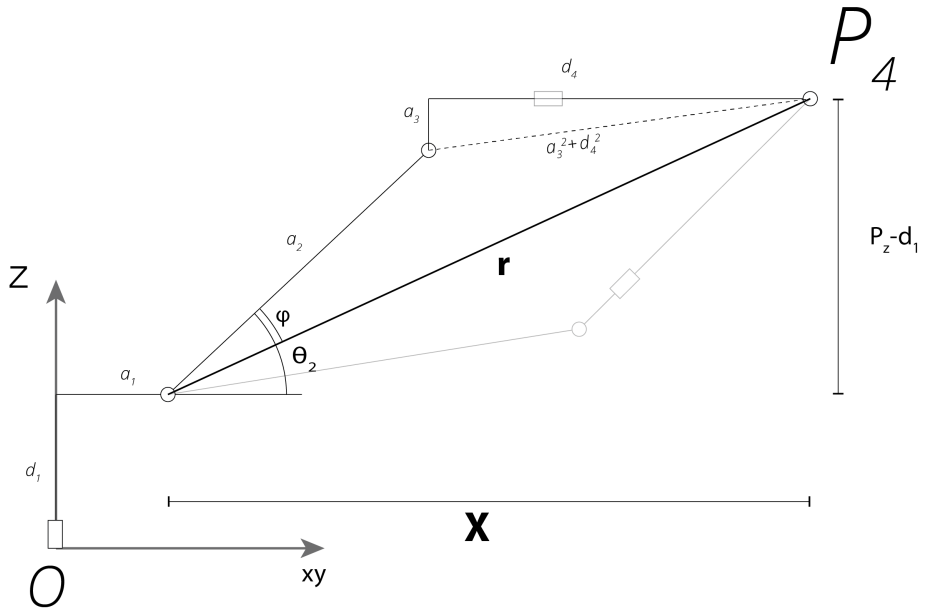

Figure 8: Calculation of the second joint angle

5.1.3 Joint 3

The third joint angle can be computed in a similar way. Figure 9 shows two diagrams which allow for the expression of θ_3.

Distances nu and du are constructed:

$$nu = \cos(\theta_2) \cdot x + \sin(\theta_2) \cdot (P_z - d_1) - a_2$$
$$du = \sin(\theta_2) \cdot x - \cos(\theta_2) \cdot (P_z - d_1)$$

And the third joint angle is calculated with:

$$\theta_3 = \text{atan2}(a_3, d_4) - \text{atan2}(nu, du)$$

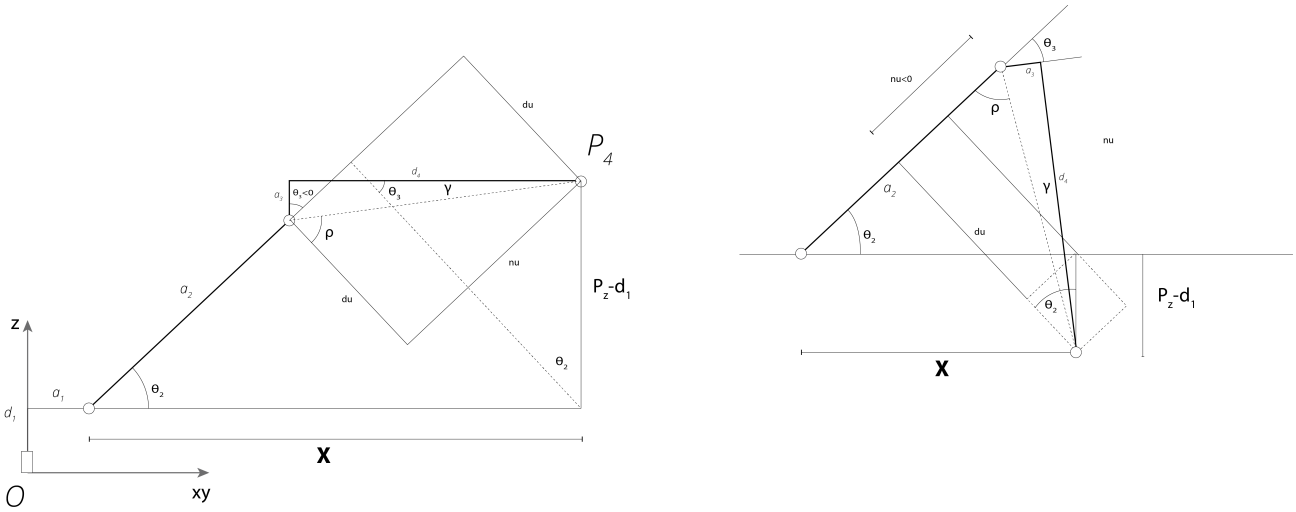

Figure 9: Calculation of the third joint angle

5.2 Inverse Orientation

Now that the first joint angles have been calculated in correspondence to a given wrist origin position, the inverse orientation approach can be used to compute the last three joint angles through Euler angle parametrization. As mentioned before this can be seen as the parametrization of a spherical wrist, where a set of Euler angles is to be found corresponding to a given rotation matrix R, as shown in Figure 10.

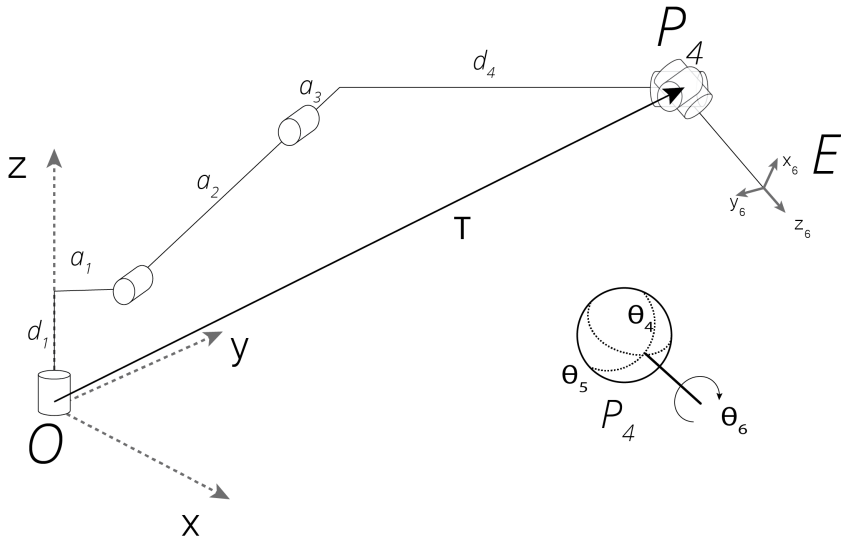

Figure 10: Calculation of the joint angles 4, 5 and 6

The rotation matrix R_6^3 is given by multiplication of the A-matrices, as mentioned in chapter 3:

$$R_6^3 = A_4 A_5 A_6$$

For the final three variables an equation can be set up as:

$$R_6^3 = (R_3^0)^T R$$

The Euler angle solution applied to this equation yields expressions for the final three angles.

However, instead of an expression in rotation matrices a geometrical approach as seen before can be used. Below a geometrical approach combined with decoupling is performed.

5.2.1 Joint 4

Support vectors Y and $M_{1,2}$ are set up as shown in Figure 11:

$$\begin{aligned} Y &= \cos(\theta_1) \cdot A_x + \sin(\theta_1) \cdot A_y \\ M_2 &= \sin(\theta_1) \cdot A_x - \cos(\theta_1) \cdot A_y \\ M_1 &= \cos(\theta_2 - \theta_3) \cdot Y + \sin(\theta_2 - \theta_3) \cdot A_z \end{aligned}$$

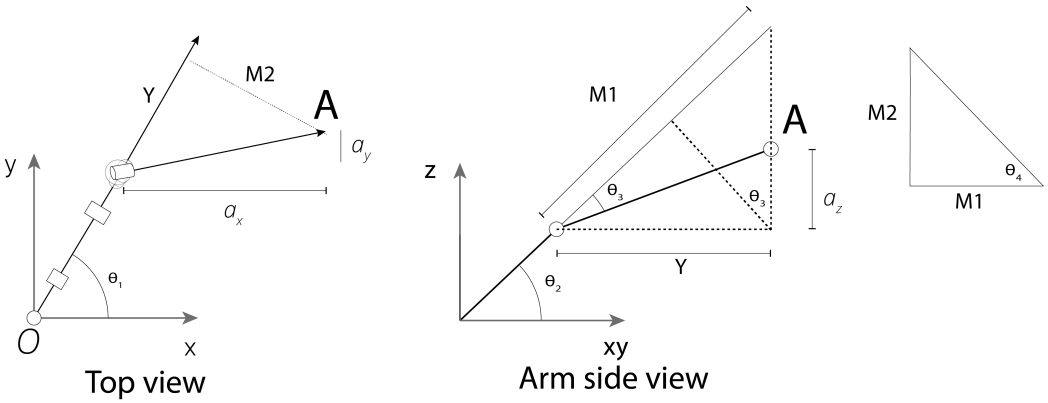

Figure 11: Calculation of the joint angles 4

These support vectors allow for an expression for θ_4:

$$\theta_4 = \text{atan2}(n_4 \cdot M_2, n_4 \cdot M_1) - \text{atan2}(Nu, Du)$$

Where n_2 and n_4 are configuration settings as before and Nu and Du are as before.

5.2.2 Joint 5

Joint 5 and 6 are derived in the same geometric manner as joint 4. For joint 5 new support vectors Nu and M_3 are set up:

$$\begin{aligned} Nu &= -\cos(\theta_4) \cdot M_1 - \sin(\theta_2) \cdot M_2 \\ M_3 &= -A_z \cdot \cos(\theta_2 - \theta_3) + Y \cdot \sin(\theta_2 - \theta_3) \cdot A_z \end{aligned}$$

Allowing for:

$$\theta_5 = \text{atan2}(Nu, M_3)$$

5.2.3 Joint 6

Finally, an expression for joint 6 is constructed. Here, the orientation of the tool frame O_c is used. The components of the end-effector position O are denoted by O_x, O_y and O_z. The wrist center components of O_c are denoted by x_c, y_c and z_c and the distance along the z-axis of the tool head is denoted by z_{th} such that:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} O_x - z_{th} r_{13} \\ O_y - z_{th} r_{23} \\ O_z - z_{th} r_{33} \end{bmatrix}$$

Support vectors Z, $L_{1,2,3}$, $A_{1,3}$ and new Nu and Du are set up:

$$\begin{aligned} Z &= \cos(\theta_1) \cdot O_x + \sin(\theta_1) \cdot O_y \\ L_2 &= \sin(\theta_1) \cdot O_x - \cos(\theta_1) \cdot O_y \\ L_1 &= \cos(\theta_2 - \theta_3) \cdot Z + \sin(\theta_2 - \theta_3) \cdot O_z \\ L_3 &= \sin(\theta_2 - \theta_3) \cdot Z + \cos(\theta_2 - \theta_3) \cdot O_z \\ A_1 &= L_1 = \cos(\theta_4) + L_2 \cdot \sin(\theta_4) \\ A_3 &= L_1 = \sin(\theta_4) - L_2 \cdot \cos(\theta_4) \\ Nu &= -A_1 \cdot \cos(\theta_5) - L_3 \cdot \sin(\theta_5) \\ Du &= -A_3 \end{aligned}$$

Which leads to an expression for the final angle:

$$\theta_6 = \text{atan2}(Nu, Du)$$

6 Trajectory Planning

We have created 3 different trajectory generators for this robot arm, using two distinct approaches. All work by entering series of points and orientations in the GUI. These get processed and a list of joint angles is generated.

6.1 Type I and II: Interpolating in task space

As mentioned earlier we have developed multiple methods of trajectory generation. The first method is shown in figure 12 and interpolates between two consecutive locations from a predefined series, in task space (XYZ), in a straight line. For each pair of locations a spline is drawn and evaluated, returning XYZ coordinates. The spacing of the locations determines velocity and acceleration in XYZ. The spacing is determined using a Sigmoidal Membership Function (SMF), which also means velocity and acceleration is zero at each location entered. Furthermore, the planner checks if the task space locations are not in a sphere

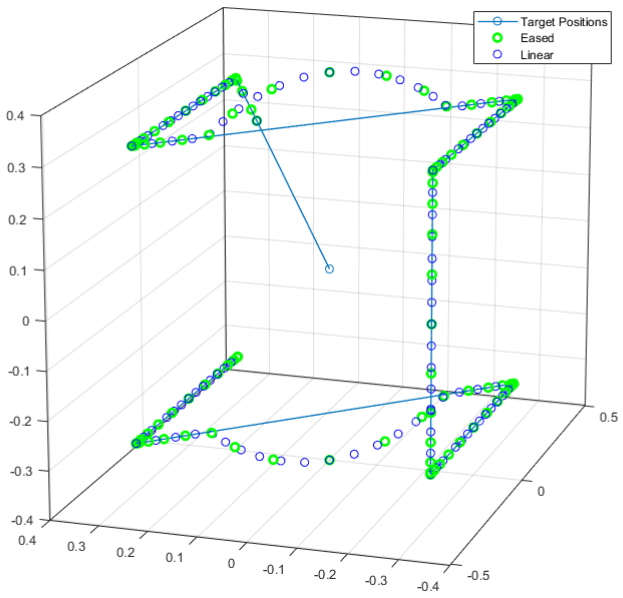

Figure 12: Point to point (PTP) interpolation in task space.

around the center. If they are, the points are moved to the edge of the sphere. This prevents the EEF from moving through itself if two target locations lie at opposite sides of the robot. The locations are then passed into the Inverse Kinematics (IK) function, which calculates all the joint angles for each location. We create smooth motion by defining a large amount of sampling points between any two locations. The planner calculates how many steps it will interpolate with based on the sample rate and maximum velocity.

The second method is an adjustment of the first and can be seen in 13. This

time we interpolate in task space using a spline that passes through all locations from a predefined series. This creates a smooth motion in task space, with start and end velocity of the movement equal to zero.

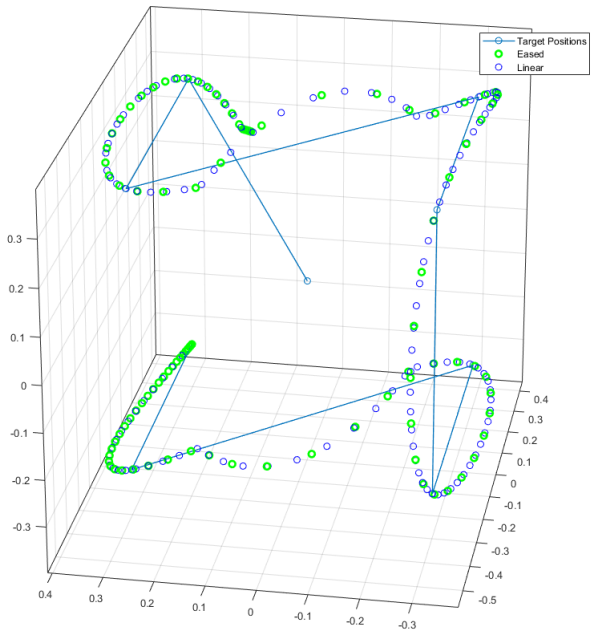

Figure 13: Spline interpolation in task space.

6.1.1 SMF in detail

Matlab has a built in SMF function, which returns a distribution composed of a two linear parts and two parabolas. The inflection points determine the start and end where the parabolas blend into the linear line. This function can easily be adjusted to LSPB or any other distribution function. Figure 14 shows the SMF distribution.

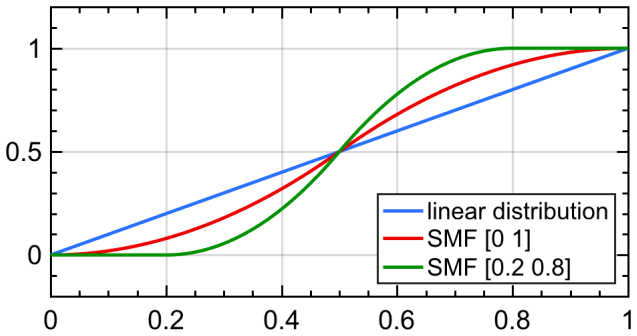

Figure 14: SMF distribution using different inflection points

6.2 Type III: Interpolating in joint space

The third method is a more advanced and flexible approach. It runs IK for all XYZ locations and interpolates between the joint angles found using quintic splines, as shown in figure 15. Time, location, velocity and acceleration have to be defined for each point. By default the time at each location is calculated using travel distance and maximum XYZ velocity. Joint velocity and acceleration can be put any value, but ought to be calculated.

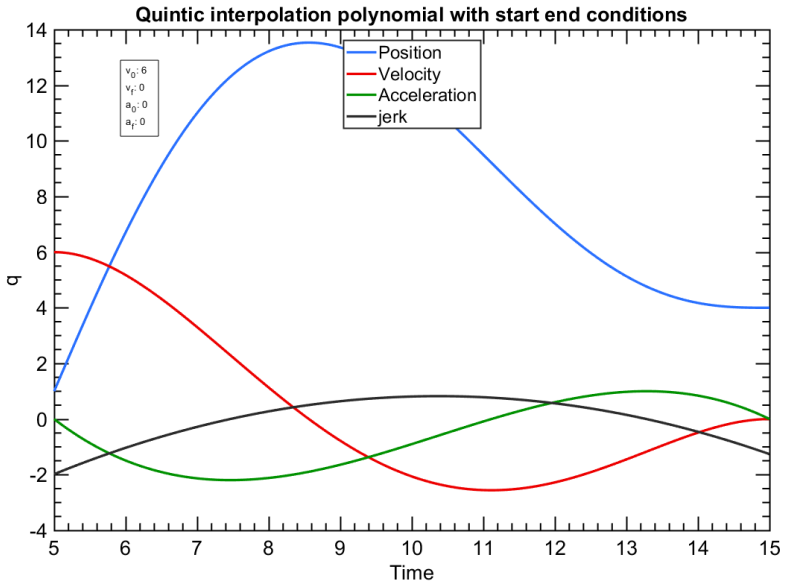

Figure 15: Quintic interpolation spline for specific start-end conditions. The range of t and q can be seen from the axes. The start and end conditions can easily create overshoot.

If we enter the calculated joint angles into our Forward Kinematics (FK) algorithm, we can plot the trajectory the EEF will make in task space, such as in figure 16. When we add points in a straight line, we can see that using quintic interpolation generates oscillatory motion in straight paths. When we set the the joint velocity over all intermediate points to be constant, we can see that the trajectory is now slightly smoother, but oscillations are also bigger as in figure 17. The trajectories generated are heavily dependent on the arm configuration chosen (elbow up/down etc.). Because we force our IK into a specific configuration, the IK function can return joint angles π and $-\pi$ for two locations close in task space, which gives a large range of q when we interpolate. This then yields strange and inefficient motion and can be seen in figure 18. This is mainly because there is no linear relationship between joint angles and task space location.

We can also plot the joint angles and their interpolations for the different start/end criteria.

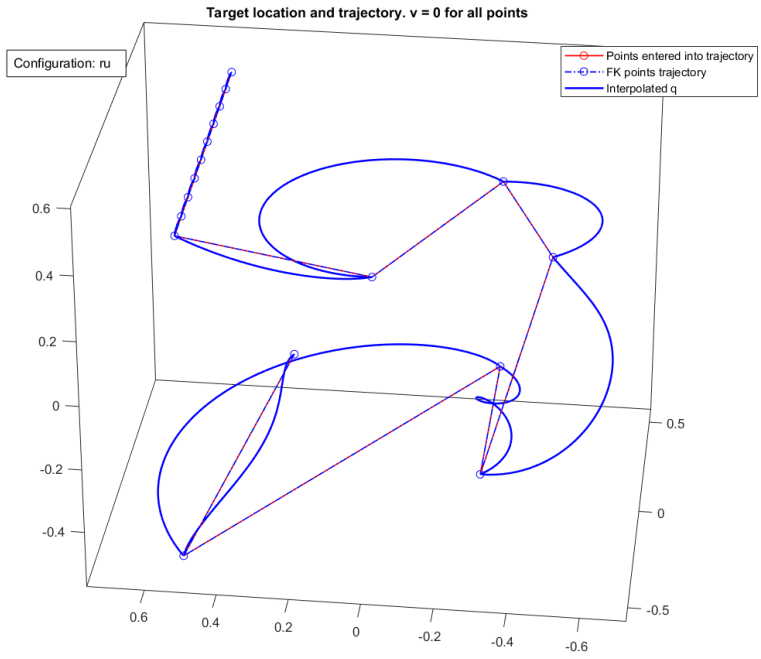

Figure 16: Quintic interpolation between joint angles. v = a = 0 for all joints.

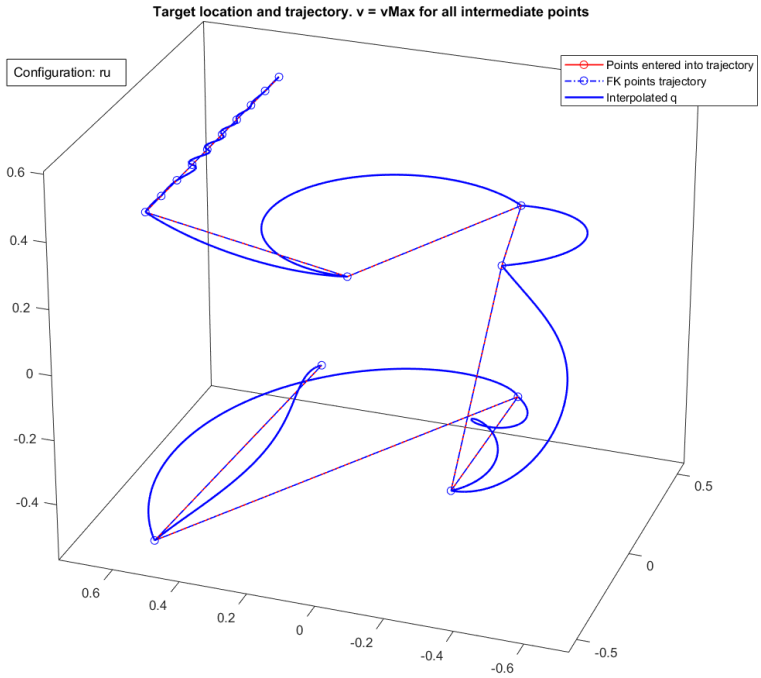

Figure 17: Quintic interpolation between joint angles. v = vMax at all intermediate points. a = 0. Note oscillations in the linear movement part.

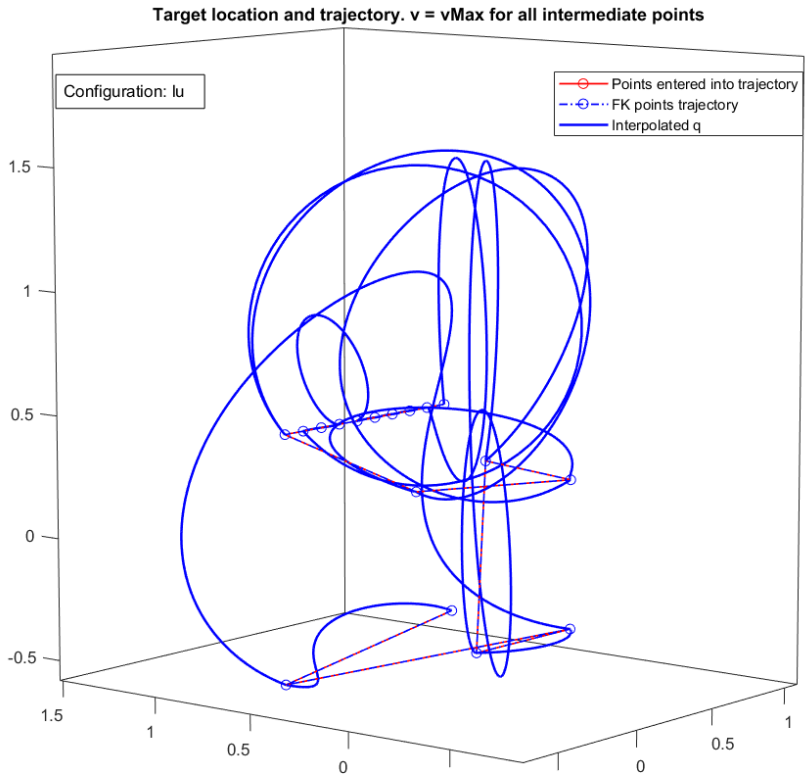

Figure 18: Trajectory generated using a different arm configuration. The IK algorithm sometimes returns $q = \pi$ and $q = -\pi$ for two close points, and the resulting high Δq will create these trajectories. This is because our IK algorithm enforces a particular configuration.

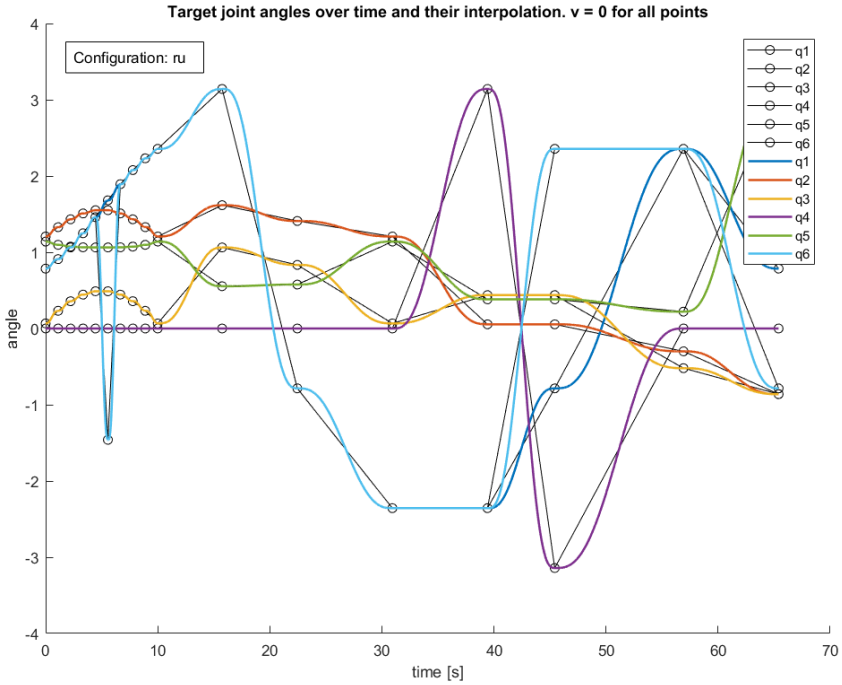

Figure 19: Quintic interpolation between joint angles. v = a = 0 at all points.

6.3 Advantages and drawbacks

A major advantage to interpolating locations in task space is that we can finely control the EEF XYZ motion. The biggest drawback is that you get no information about joint angles and their derivatives and it is hard to implement constraints on these.

The big advantage of interpolation in joint space is its versatility. It yields maximum control over joint positions, constraints and timing. Provided that you define a lot of points, even linear motion can be approximated. When all sets of (q) calculated by the IK are admissible, all interpolated values will adhere to joint constraints by default. A big drawback is that it is much more complex to implement properly. To increase control over EEF movement in task space, joint velocities and accelerations should be calculated for each point, but since there is no linear relation between joint angles and XYZ, this is not trivial.

6.4 Recommendations

Improving the trajectory planning at its current state is best done by optimising the IK algorithm. Trajectories can be made much faster and more efficient if the optimal set of joint angles can be selected *before* interpolation. This could be done by calculating all configurations for all positions and choosing the set that represents minimal Δq, or adheres to another criterion. This means that the

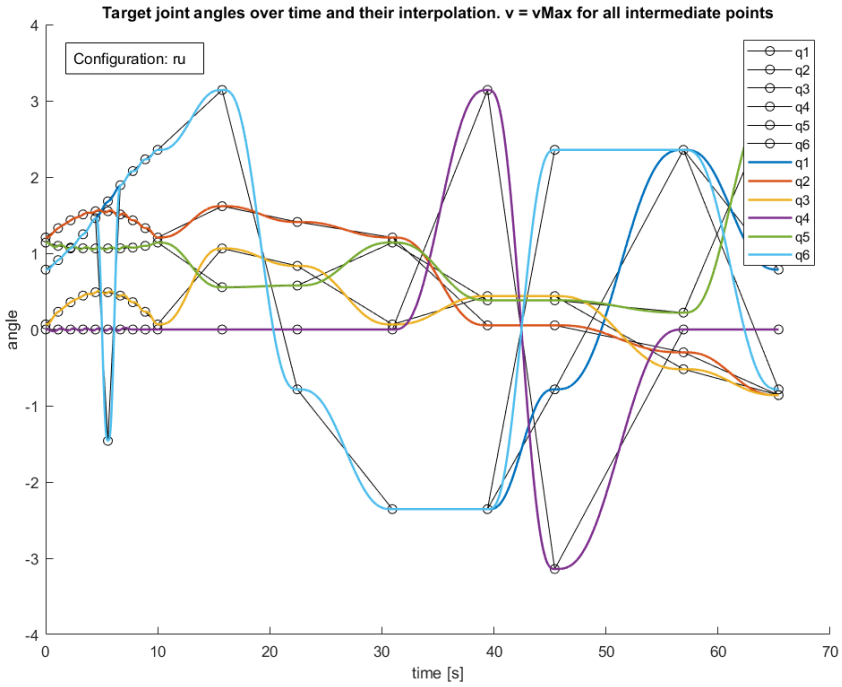

Figure 20: Quintic interpolation between joint angles. v = vMax at all intermediate points. a = 0. Top view. Note the oscillations and overshoot increase.

IK algorithm should be able to output multiple configurations in one trajectory. To have more control over task space movement, such as constant motion over straight lines, joint velocities and accelerations should be calculated for each task space location.

7 CAD & Simulink

In order to control and visualize the KR10 we combined the power of Simulink and SolidWorks. This decision to use Simscape Multibody over SPANviewer was made due to the fact that not everyone in the team was using a Windows PC which was required to run SPANviewer. We found it necessary that everyone could run and visualize the system without using Virtual Machines or dual boot which greatly impedes efficiency with tight schedules. Since SPANviewer would only be used for visualization we did not see any benefit over SimScape Multibody.

After the robotic arm was imported into SolidWorks it was stripped from unnecessary parts and a toolhead was added. Finally, we exported the model to a Simscape Multibody compatible file, which is in essence a Simulink model.

Simscape Multibody is a simulation toolbox for multibody simulation. It allows definition of various bodies, links and joints. The export plugin in SolidWorks automatically creates the rigid bodies and the revolute joints in a Simulink model.

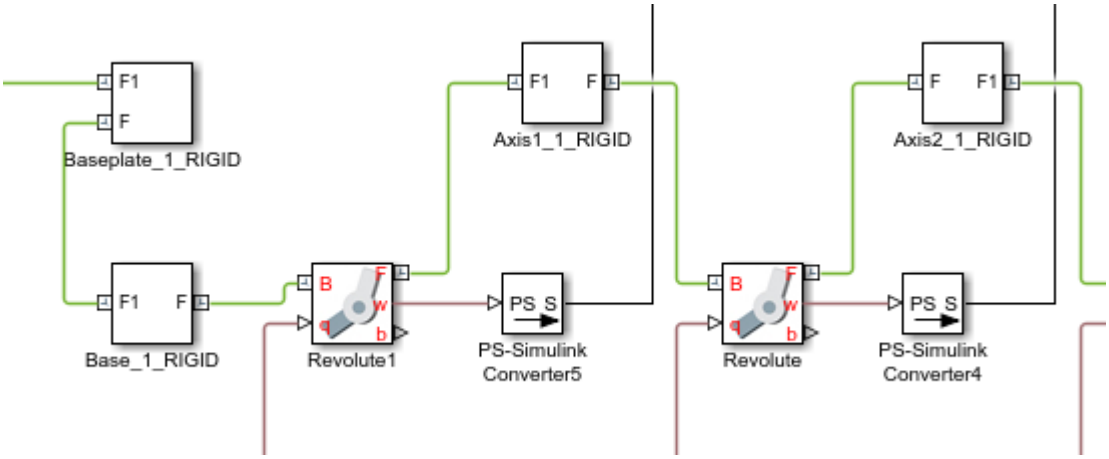

Figure 21: Section of the SimScape model with revolute joints and rigid bodies

In figure 21 we can partially see what the export plugin delivers.

Each revolute joint has an in-port for the joint angle. These angles are received over a UDP socket using a MATLAB script which updates the model after each received package.

7.1 Recommendations

Unfortunately the Simscape Multibody model runs at a variable time step making it difficult to control the real time aspect of visualization. Various attempts have been made to pace the simulation with soft real-time but this did not yield desireable results. One of the main problems is that the values are sent one by one and thus, received one by one. Instead, it would be better to send the entire range of joint angles and store them in a buffer. After the last joint angle has been sent a flag can be set triggering the SimScape model to read in the joint angles at a specified pace which the PC can handle.

Furthermore, some sort of feedback to the GUI would also have been nice. Attempts have been made to accomplish this using various callback methods built into MATLAB and Simulink. These callbacks worked as intended, however, updating elements of the GUI take a long time which slows down the entire system. The GUI is not meant to be updated in real time, therefore, attempts for two-way communication have been dropped for performance reasons. Even though this functionality has been removed from the `master` branch it is still available in commit `18ebfa647`[6] for potential further work.

[6] `https://github.com/Ortix92/space-robotics/tree/18ebfa647e`

8 Graphical User Interface

The piece of software which ties everything together is the Graphical User Interface (GUI). The GUI has been made in MATLAB's App Designer which is the successor to GUIDE. It uses an Object Oriented approach which we made heavy use of.

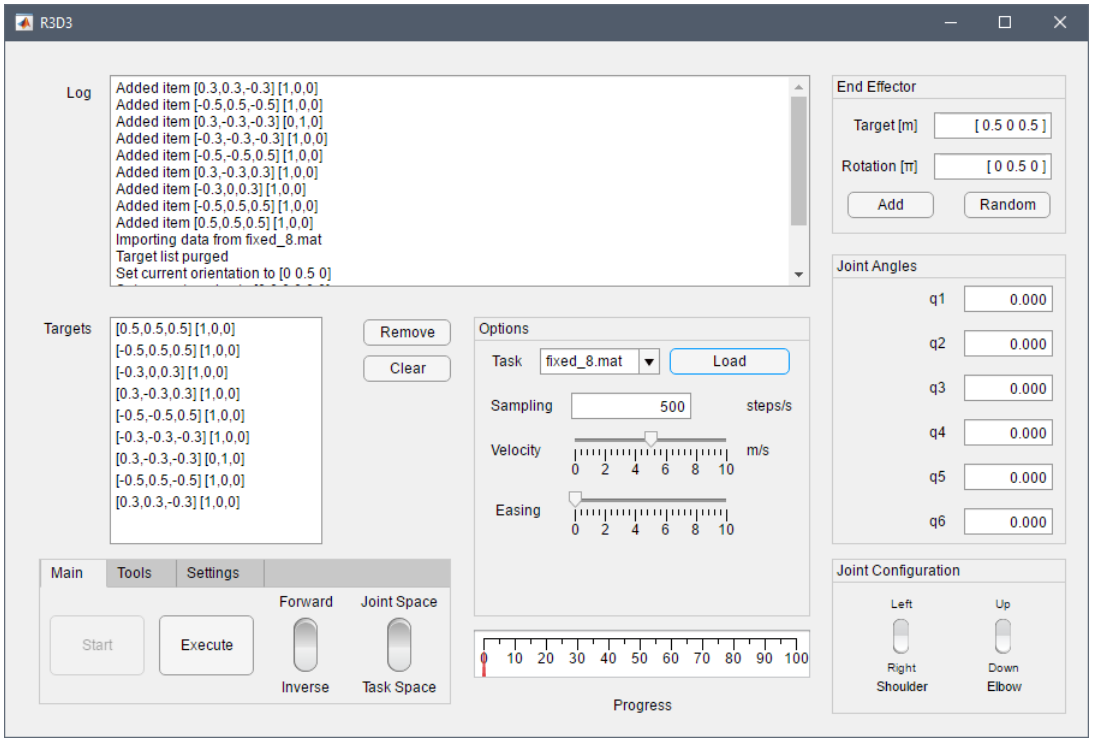

Figure 22: The GUI handles the UDP communication and allows interfacing with various aspects of the robot arm.

The interface, as can be seen in figure 22, has plenty of options to control the arm. It has two main functioning settings: forward and inverse kinematics.

The forward kinematics allows the user to enter joint angles. Based on a setting which can be selected it will either directly broadcast the values over UDP or runs it through our `forwardKinematics()` function first to get the end effector position and orientation and then through the `inverseKinematics()` function to calculate the joint angles. This can be used to validate the algorithm since both settings should yield the same end effector configuration.

In the Inverse Kinematics mode the user enters locations for the end effector and the orientation. The orientation should be given as Euler angles in π radians in vector form. The entries correspond to rotations about the x, y and z axes, respectively. Once the fields have been filled the point can be added. Validation has been implemented to prevent points outside the range of the robot arm. An additional button has also been added to create random points.

Another interesting feature is the options panel where the user can load 8 random points or a preselected set which goes through all 8 quadrants. The sampling rate (amount of samples per second), the velocity (m/s) of the end effector and amount of easing (no units, subjective) can also be set. The 8 random points are generated inside a shell which consist of an intersection of 2 spheres where the inner sphere contains disallowed points. This can be visualized in figure 23 where the permissible points are within $-2 < r < 2$.

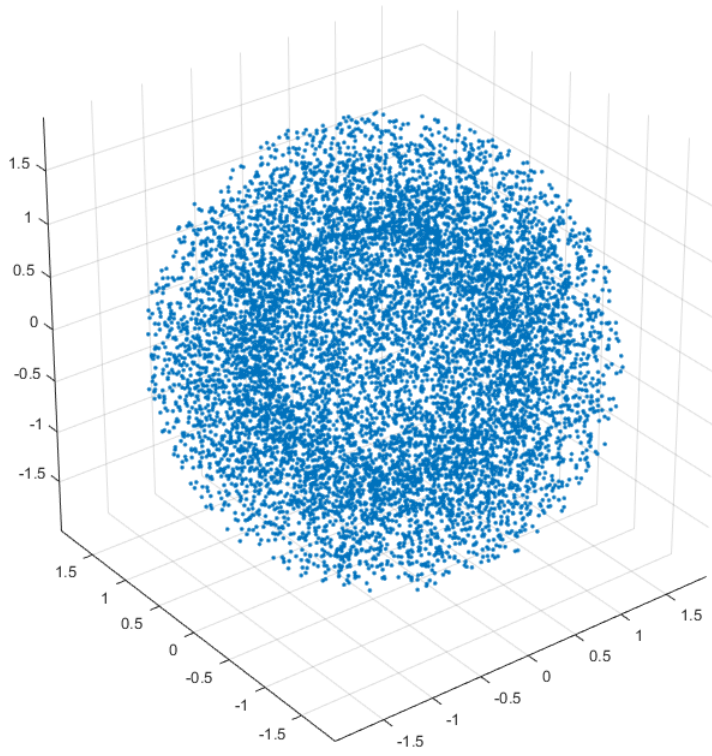

Figure 23: A shell with points permissible between $-2 < r < 2$. A faint ring can be seen around the disallowed region.

8.1 Recommendations

Even though the GUI functions properly there is a lot of room for improvement. Due to lack of time these ideas have not been implemented but are very much feasible. These recommendations have been listed below.

- Input validation for all fields.
- Improved target point validation with actual working envelope.

- Improved random point generation by using actual working envelope.
- Switch to enable/disable plots.
- Faster GUI implementation with e.g. Python.
- Synchronous updating between GUI and Simulink without UDP.
- Map velocity and acceleration constraints to target points.
- Improve determinism of calculations for soft real-time performance.

9 Conclusion

The goal of this project was to model an industrial robot manipulator and its movement over a specified path. This task has been executed successfully, ensuring correct trajectory planning and movement through all quadrants with fitting constraints concerning time position and velocity. A Graphical User Interface has been conceived which allows for an interactive input of target points, either individually or with a file, and the possibility to select multiple generic solutions corresponding to the manipulator's physical configuration at different speeds with different degrees of motion easing.

The biggest challenge during the design of the model turned out to be the inverse kinematics. Whereas normally calculations for the inverse kinematics are done through rotation matrix calculations for the inverse orientation, our solution was a result of a geometrical approach which in hindsight may have been devious. But a functioning model has been achieved through this method, and linkage between the GUI, kinematics, trajectory and visualization has been established.

We found this course and in particular this project to be very interesting and educational. In no other course that we have followed we were handed a project where the project task was to design and model a practical robot from start to end. Even though more complex dynamics have not been taken into account in this project, we found it to be very useful and satisfying to conclude the course with a visualization of a functioning, controllable, self-built robot manipulator.

10 Contribution of each indivual member

Part	subject	Roel	Martin	Jeffrey	Nick
Inverse Kinematics	Derivation	X	X	X	
	Implementation	X	X	X	X
	Report	X			
Forward Kinematics	Derivation	X	X	X	X
	Implementation	X	X	X	X
	Report				X
Trajectory planning	Derivation	X	X	X	X
	Implementation		X		
	Report		X		
GUI and visualisation	Derivation			X	X
	Implementation			X	X
	Report				X