34753: Robotics Fall 2024

Final project - Group 3

Ava Margaux Bennett (s242184)

Moritz Genzwürker (s223293)

Caroline Victoria Lange (s242182)

Sophie Katharina Clara Rücker (s242180)

Louise Julie Sternholdt-Sørensen (s173937)

Simon Therkildsen (s193962)

Handed in:

14th September 2025

Contents

1 Introduction

Group 3's Final Project Report demonstrates the group's collaborative effort to apply and practice the theoretical concepts and topics learned in the Robotics course 34753, on a real robotic system. The project is structured into five main parts, each focussing on a critical aspect of robotics, including kinematics, trajectory planning, singularities and statistics, dynamics and computer vision and control. This knowledge was then applied to program a simple 4-DOF AX-12A servo motor-based robotic arm to complete a chosen task. Each section of this project is summarised in the following report with the full solutions implemented in the accompanying MATLAB file.

2 Kinematics

2.1 Direct Kinematic Transformations

Forward kinematics is used to calculate the position and orientation of a robot's end-effector given its joint parameters. In particular, the Denavit-Hartenberg (DH) convention provides a standardized method so that the homogeneous transformation can be represented by the following four parameters:

$$\begin{aligned}
\theta_i &: \text{ angle from } x_{i-1} \text{ to } x_i \text{ about } z_{i-1} \\
d_i &: \text{ distance from } o_{i-1} \text{ to } x_i \text{ (along } z_{i-1}) \\
a_i &: \text{ distance from } z_{i-1} \text{ and } o_i \text{ (along } x_i) \\
\alpha_i &: \text{ angle from } z_{i-1} \text{ to } z_i \text{ about } x_i
\end{aligned}$$

Each transformation matrix A_i represents the transformation from frame $i-1$ to frame i. The general form of the DH transformation matrix is:

$$A_i = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)\sin(\theta_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the given 4-link manipulator robot, the transformation matrices are given by:

$$A_1 = \begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & a_2\sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & a_3\cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & a_3\sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & a_4\cos(\theta_4) \\ \sin(\theta_4) & \cos(\theta_4) & 0 & a_4\sin(\theta_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 1 & 0 & 0 & -15 \\ 0 & 1 & 0 & 45 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix A_5 doesn't follow the DH convention as it is the frame of the end effector. So, we just use the homogeneous transformation matrix instead.

Position and orientation of the Robot stylus T_4^0 can be found by multiplying together the individual transformation matrices, where $d_1 = 50$ mm, $a_2 = 93$ mm, $a_3 = 93$ mm, $a_4 = 50$ mm, $\alpha_1 = \frac{\pi}{2}$ as given in the schematic.

$$T_4^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \tag{2.1}$$

$$\mathrm{T}_4^0 = \begin{bmatrix} \cos(\theta_2+\theta_3+\theta_4)\cos(\theta_1) & -\sin(\theta_2+\theta_3+\theta_4)\cos(\theta_1) & \sin(\theta_1) & \frac{1}{1000}(\cos(\theta_1)(50\cos(\theta_2+\theta_3+\theta_4)+93\cos(\theta_2+\theta_3)+93\cos(\theta_2))) \\ \cos(\theta_2+\theta_3+\theta_4)\sin(\theta_1) & -\sin(\theta_2+\theta_3+\theta_4)\sin(\theta_1) & -\cos(\theta_1) & \frac{1}{1000}(\sin(\theta_1)(50\cos(\theta_2+\theta_3+\theta_4)+93\cos(\theta_2+\theta_3)+93\cos(\theta_2))) \\ \sin(\theta_2+\theta_3+\theta_4) & \cos(\theta_2+\theta_3+\theta_4) & 0 & \frac{\sin(\theta_2+\theta_3+\theta_4)}{20}+\frac{93}{1000}\sin(\theta_2+\theta_3)+\frac{93}{1000}\sin(\theta_2)+\frac{1}{20} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As well as the position and orientation of the robot camera T_5^0,

$$T_5^0 = T_4^0 \cdot T_5^4 = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5 \tag{2.2}$$

$$\mathrm{T}_5^0 = \begin{bmatrix} \cos(\theta_2+\theta_3+\theta_4)\cos(\theta_1) & -\sin(\theta_2+\theta_3+\theta_4)\cos(\theta_1) & \sin(\theta_1) & \frac{1}{1000}(cos(\theta_1)(35\cos(\theta_2+\theta_3+\theta_4)-45*\sin(\theta_2+\theta_3+\theta_4)+93*\cos(\theta_2+\theta_3)+93*\cos(\theta_2))) \\ \cos(\theta_2+\theta_3+\theta_4)\sin(\theta_1) & -\sin(\theta_2+\theta_3+\theta_4)\sin(\theta_1) & -\cos(\theta_1) & \frac{1}{1000}(sin(\theta_1)(35\cos(\theta_2+\theta_3+\theta_4)-45*\sin(\theta_2+\theta_3+\theta_4)+93*\cos(\theta_2+\theta_3)+93*\cos(\theta_2))) \\ \sin(\theta_2+\theta_3+\theta_4) & \cos(\theta_2+\theta_3+\theta_4) & 0 & \frac{93}{1000}\sin(\theta_2+\theta_3)+\frac{93*\sin(\theta_2)}{1000}+\frac{\sqrt{130}\cos(\theta_2+\theta_3+\theta_4-\arctan\frac{7}{5})}{200}+\frac{1}{20} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.2 Inverse Kinematic Transformation

Obtaining the position and orientation of a given link as a function of joint angles q is trivial once the forward kinematics $H(q)$ is known. Doing the opposite, i.e. solving the system of equations $H(q) = [o_4^0, x_4^0]^T$ for q, is nontrivial, but a simpler method can be obtained through the two following observations. Consider the illustration of the robot in figure (2.1), with a desired position o_4^0 and orientation x_4^0 of the end effector:

- Only the first joint allows rotation around the global z-axis, so q_1 can be determined from the x-, and y-component of o_4^0.
- Since the length of the 4'th link a_4 is known, the position of point p can be determined from o_4^0 and x_4^0. It is then possible to solve for q_2 and q_3 such that link 3 is at point p.

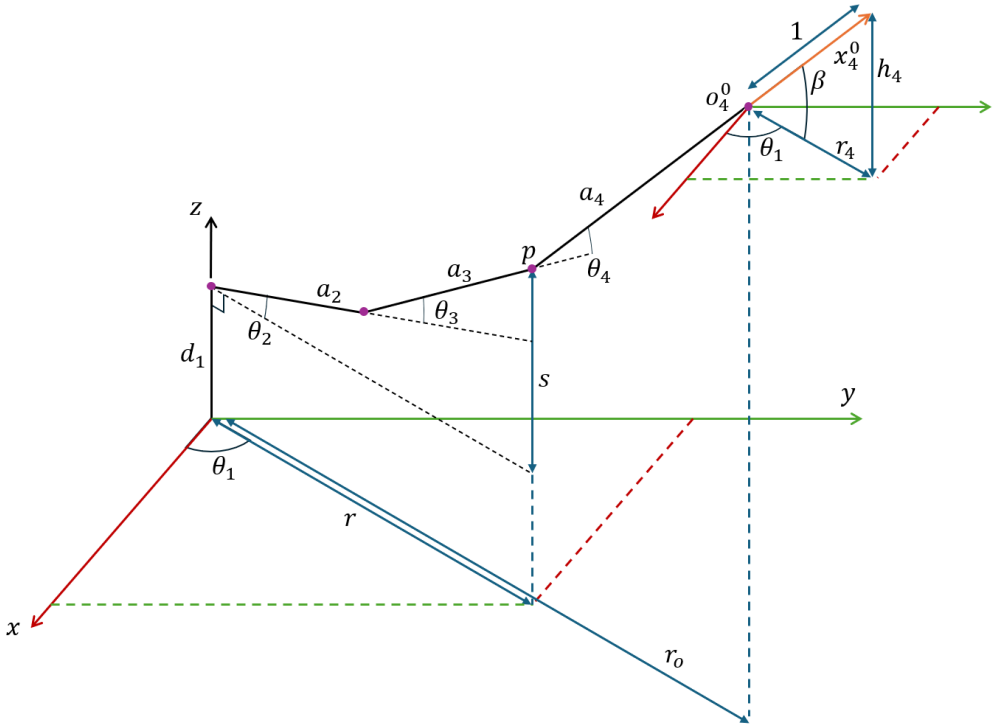


Figure 2.1: Sketch of robot configuration for computing inverse kinematics

With these two observations, q is determined as follows. q_1 is computed from o_4^0

$$q_1 = \text{Atan2}(o_{4,y}^0,\ o_{4,x}^0) \tag{2.3}$$

where $o_{4,x}^0$ and $o_{4,y}^0$ are the x, and y-components of o_4^0. The position of p is computed from o_4^0 and x_4^0

$$p = o_4^0 - a_4 \cdot x_4^0 \tag{2.4}$$

It should be noted that while o_4^0 can be chosen freely[1], x_4^0 cannot, since the orientation of the end effector partially depends on θ_1. For example, if $\theta_1 = 0$, the end effector orientation will always be parallel to the global xz-plane. Instead, o_4^0 and β are chosen freely, and then x_4^0 is computed from these values. Since x_4^0 is a unit vector, h_4 and r_4 must be scaled accordingly. h_4 is computed directly from β and r_4 is obtained from the Pythagorean theorem.

$$h_4 = \sin(\beta), \qquad h_4^2 + r_4^2 = 1 \quad \rightarrow \quad r_4 = \sqrt{1 - h_4^2} \tag{2.5}$$

The x and y-component of x_4^0 are obtained by scaling the x and y-component of o_4^0. First, they are normalized with r_0, then they are scaled with r_4

[1] Within the robot's reachable workspace

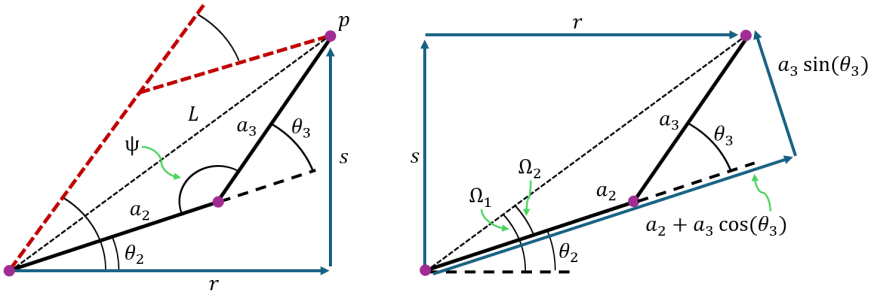


Figure 2.2: Sketch of links 2 and 3 for computing inverse kinematics. Note that the same situation is depicted on the left and right - two figures are used to avoid clutter.

$$x^0_{4,x} = \frac{o^0_{4,x}}{r_o} r_4 \qquad x^0_{4,y} = \frac{o^0_{4,y}}{r_o} r_4 \tag{2.6}$$

Link 2 and 3 are shown in figure (2.2). The law of cosines is applied to obtain the angle Ψ

$$L^2 = a_2^2 + a_3^2 - 2a_2a_3\cos(\Psi) \quad \longrightarrow \quad \cos(\Psi) = \frac{a_2^2 + a_3^2 - L^2}{2a_2a_3} \tag{2.7}$$

The angle θ_3 is equal to $\theta_3 = \pi - \Psi$, thus eq. 2.7 can be rewritten utilizing that $\cos(\pi - \Psi) = -\cos(\Psi)$. Additionally, an expression for $\sin(\theta_3)$ is obtained using the Pythagorean theorem

$$\cos(\theta_3) = -\cos(\Psi) = \frac{-a_2^2 - a_3^2 + L^2}{2a_2a_3} \qquad \sin(\theta_3) = \sqrt{1 - \cos^2(\theta_3)} \tag{2.8}$$

With these values, θ_3 is computed using

$$\theta_3 = \text{Atan2}(\pm\sin(\theta_3),\ \cos(\theta_3)) \tag{2.9}$$

where two results are obtained, one corresponding to the solid black lines in figure (2.2), and the other corresponding to the red dashed lines. The angle θ_2 is then computed using

$$\theta_2 = \Omega_1 - \Omega_2 = \text{Atan2}(s,\ r) - \text{Atan2}(a_3\sin(\theta_3),\ a_2 + a_3\cos(\theta_3)) \tag{2.10}$$

Lastly, θ_4 is computed such that the end effector has angle β wrt. the xy-plane

$$\theta_4 = \beta - \theta_2 - \theta_3 \tag{2.11}$$

2.3 Robot Configurations for Tracking of a Circle

Based on the Inverse Kinematic Transformation from Section 2.2 and using Equation 2.12, it is possible to determine a sequence of robot configurations (i.e. joint angles q_1, q_2, q_3, q_4) along the circle. These will, in the subsequent sections, be used to plan a trajectory along the perimeter of the circle.

$$p^0(\varphi) = p_c^0 + R \begin{bmatrix} 0 \\ \cos(\varphi) \\ \sin(\varphi) \end{bmatrix} \quad \text{for } 0 \leq \varphi \leq 2\pi \tag{2.12}$$

Where $R = 32\,\text{mm}$ and $p_c = [150, 0, 120]^T$ mm. The configurations, as well as the position of the end-effector in cartesian coordinates, are given in Table 2.1.

Plotting all configurations in 3D results in the Figure 2.3.

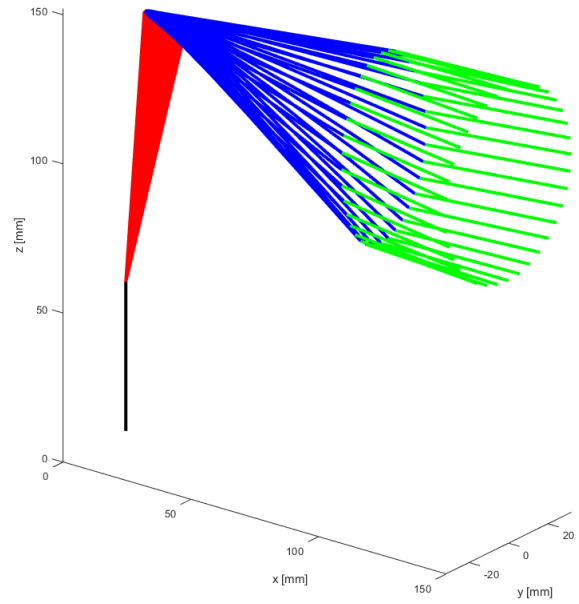


Figure 2.3: Plot of the 37 robot configurations that constitute the tracked circle.

Table 2.1: Sequence of Robot Configurations necessary for the stylus tip to track 37 equidistant points. Units are in radians and millimetres.

Conf.	φ	**x**	**y**	**z**	q_1	q_2	q_3	q_4
0	0.0000	150.0000	32.0000	120.0000	0.2102	1.4302	-1.6699	0.2397
1	0.1745	150.0000	31.5138	125.5567	0.2071	1.4437	-1.6242	0.1805
2	0.3491	150.0000	30.0702	130.9446	0.1978	1.4556	-1.5789	0.1233
3	0.5236	150.0000	27.7128	136.0000	0.1827	1.4656	-1.5354	0.0698
4	0.6981	150.0000	24.5134	140.5692	0.1620	1.4737	-1.4952	0.0214
5	0.8727	150.0000	20.5692	144.5134	0.1363	1.4801	-1.4597	-0.0204
6	1.0472	150.0000	16.0000	147.7128	0.1063	1.4848	-1.4304	-0.0544
7	1.2217	150.0000	10.9446	150.0702	0.0728	1.4880	-1.4084	-0.0795
8	1.3963	150.0000	5.5567	151.5138	0.0370	1.4898	-1.3949	-0.0950
9	1.5708	150.0000	0.0000	152.0000	0.0000	1.4904	-1.3903	-0.1002
10	1.7453	150.0000	-5.5567	151.5138	-0.0370	1.4898	-1.3949	-0.0950
11	1.9199	150.0000	-10.9446	150.0702	-0.0728	1.4880	-1.4084	-0.0795
12	2.0944	150.0000	-16.0000	147.7128	-0.1063	1.4848	-1.4304	-0.0544
13	2.2689	150.0000	-20.5692	144.5134	-0.1363	1.4801	-1.4597	-0.0204
14	2.4435	150.0000	-24.5134	140.5692	-0.1620	1.4737	-1.4952	0.0214
15	2.6180	150.0000	-27.7128	136.0000	-0.1827	1.4656	-1.5354	0.0698
16	2.7925	150.0000	-30.0702	130.9446	-0.1978	1.4556	-1.5789	0.1233
17	2.9671	150.0000	-31.5138	125.5567	-0.2071	1.4437	-1.6242	0.1805
18	3.1416	150.0000	-32.0000	120.0000	-0.2102	1.4302	-1.6699	0.2397
19	3.3161	150.0000	-31.5138	114.4433	-0.2071	1.4152	-1.7146	0.2994
20	3.4907	150.0000	-30.0702	109.0554	-0.1978	1.3992	-1.7571	0.3579
21	3.6652	150.0000	-27.7128	104.0000	-0.1827	1.3829	-1.7963	0.4134
22	3.8397	150.0000	-24.5134	99.4308	-0.1620	1.3669	-1.8311	0.4643
23	4.0143	150.0000	-20.5692	95.4866	-0.1363	1.3521	-1.8608	0.5087
24	4.1888	150.0000	-16.0000	92.2872	-0.1063	1.3393	-1.8846	0.5452
25	4.3633	150.0000	-10.9446	89.9298	-0.0728	1.3295	-1.9019	0.5724
26	4.5379	150.0000	-5.5567	88.4862	-0.0370	1.3233	-1.9125	0.5892
27	4.7124	150.0000	0.0000	88.0000	0.0000	1.3212	-1.9160	0.5949
28	4.8869	150.0000	5.5567	88.4862	0.0370	1.3233	-1.9125	0.5892
29	5.0615	150.0000	10.9446	89.9298	0.0728	1.3295	-1.9019	0.5724
30	5.2360	150.0000	16.0000	92.2872	0.1063	1.3393	-1.8846	0.5452
31	5.4105	150.0000	20.5692	95.4866	0.1363	1.3521	-1.8608	0.5087
32	5.5851	150.0000	24.5134	99.4308	0.1620	1.3669	-1.8311	0.4643
33	5.7596	150.0000	27.7128	104.0000	0.1827	1.3829	-1.7963	0.4134
34	5.9341	150.0000	30.0702	109.0554	0.1978	1.3992	-1.7571	0.3579
35	6.1087	150.0000	31.5138	114.4433	0.2071	1.4152	-1.7146	0.2994
36	6.2832	150.0000	32.0000	120.0000	0.2102	1.4302	-1.6699	0.2397

2.4 Jacobian of the Manipulator

The Jacobian matrix J relates the joint velocities $\dot{q}$ to the linear and angular velocities ξ of a robot end-effector:

$$\xi = J \cdot \dot{q} \tag{2.13}$$

where:

- $\dot{x} = [\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$ represents the linear and angular velocities.
- $\dot{q} = [\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4]^T$ are the joint velocities.
- J is the 6×4 matrix for this manipulator.

To compute the Jacobian for the 4-link robotic manipulator, we must consider the contribution of each joint to the position and orientation of the desired frame. In revolute joints, the Jacobian matrix is made up of two components: Linear Velocity Component $J_{v,i}$, given by:

$$J_{v,i} = z_{i-1} \times (o_n - o_{i-1}) \tag{2.14}$$

where:

- z_i: The z-axis of the i-th frame (rotation axis of the joint i).
- o_i: The origin of the i-th frame.
- o_n: The origin of the n-th frame (target frame, 4 or 5).

and the Angular Velocity Component $J_{w,i}$, given by:

$$J_{w,i} = z_{i-1} \tag{2.15}$$

where z_i is the z-axis of the i-th frame.

The complete Jacobian matrix J for the n-th frame is constructed by stacking the Linear velocity component, J_v and the Angular velocity component, J_w:

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} \tag{2.16}$$

The Jacobian matrix for the J_4 (end-effector) is given by:

$$J_4 = \begin{bmatrix} z_0 \times (o_4 - o_0) & z_1 \times (o_4 - o_1) & z_2 \times (o_4 - o_2) & z_3 \times (o_4 - o_3) \\ z_0 & z_1 & z_2 & z_3 \end{bmatrix}$$

Similarly, the Jacobian matrix for the J_5 (end-effector) is given by:

$$J_5 = \begin{bmatrix} z_0 \times (o_5 - o_0) & z_1 \times (o_5 - o_1) & z_2 \times (o_5 - o_2) & z_3 \times (o_5 - o_3) & z_4 \times (o_5 - o_4) \\ z_0 & z_1 & z_2 & z_3 & z_4 \end{bmatrix}$$

The Jacobian matrices J_4 (end effector) and J_5 (camera) are evaluated for joint configurations obtained along a circular path. The configurations depend on the parameter φ, which specifies the orientation of the robot on the circular trajectory.

Using the values of the Transformation matrix T_4^0 (Equation 2.1) for the robot stylus and camera T_5^0 (Equation 2.2), for each value of φ, the Jacobians J_4 (end effector) and J_5 (camera) were calculated.

For $\varphi = 0$:

$$J_4 = \begin{bmatrix} -0.0320 & -0.0685 & 0.0216 & -0.0000 \\ 0.1500 & -0.0146 & 0.0046 & 0.0000 \\ 0.0000 & 0.1534 & 0.1403 & 0.0500 \\ 0.0000 & 0.2086 & 0.2086 & 0.2086 \\ 0.0000 & -0.9780 & -0.9780 & -0.9780 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}, \quad J_5 = \begin{bmatrix} -0.0289 & -0.1125 & -0.0224 & -0.0440 & -0.0440 \\ 0.1353 & -0.0240 & -0.0048 & -0.0094 & -0.0094 \\ 0.0000 & 0.1384 & 0.1253 & 0.0350 & -0.0150 \\ 0.0000 & 0.2086 & 0.2086 & 0.2086 & 0.2086 \\ 0.0000 & -0.9780 & -0.9780 & -0.9780 & -0.9780 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

For $\varphi = \frac{\pi}{2}$:

$$J_4 = \begin{bmatrix} -0.0000 & -0.1020 & -0.0093 & -0.0000 \\ 0.1500 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.1500 & 0.1425 & 0.0500 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & -1.0000 & -1.0000 & -1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}, \quad J_5 = \begin{bmatrix} -0.0000 & -0.1470 & -0.0543 & -0.0450 & -0.0450 \\ 0.1350 & 0.0000 & 0.0000 & 0.0000 & -0.0000 \\ 0.0000 & 0.1350 & 0.1275 & 0.0350 & -0.0150 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

For $\varphi = \pi$:

$$J_4 = \begin{bmatrix} 0.0320 & -0.0685 & 0.0216 & 0.0000 \\ 0.1500 & 0.0146 & -0.0046 & 0.0000 \\ 0.0000 & 0.1534 & 0.1403 & 0.0500 \\ 0.0000 & -0.2086 & -0.2086 & -0.2086 \\ 0.0000 & -0.9780 & -0.9780 & -0.9780 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}, \quad J_5 = \begin{bmatrix} 0.0289 & -0.1125 & -0.0224 & -0.0440 & -0.0440 \\ 0.1353 & 0.0240 & 0.0048 & 0.0094 & 0.0094 \\ 0.0000 & 0.1384 & 0.1253 & 0.0350 & -0.0150 \\ 0.0000 & -0.2086 & -0.2086 & -0.2086 & -0.2086 \\ 0.0000 & -0.9780 & -0.9780 & -0.9780 & -0.9780 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

For $\varphi = \frac{3\pi}{2}$:

$$J_4 = \begin{bmatrix} 0.0000 & -0.0380 & 0.0521 & 0.0000 \\ 0.1500 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.1500 & 0.1270 & 0.0500 \\ 0.0000 & -0.0000 & -0.0000 & -0.0000 \\ 0.0000 & -1.0000 & -1.0000 & -1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}, \quad J_5 = \begin{bmatrix} 0.0000 & -0.0830 & 0.0071 & -0.0450 & -0.0450 \\ 0.1350 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.1350 & 0.1120 & 0.0350 & -0.0150 \\ 0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 \\ 0.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

2.5 Joint Velocities Along Circular Path

In the following, all variables relate to the end effector, such that, e.g. v_4^0 is rewritten to v, and $v_{4,x}^0$ is rewritten to v_x etc.

The linear and angular end effector velocities are computed from the Jacobian and the joint velocities, i.e. $J\dot{q} = \xi$ where $\xi = [v, \omega]^T$. We now want to solve it $\dot{q}$ given a desired ξ. However, this presents a system of six equations with only four unknowns - a condition referred to as "underactuated", as the robot has fewer degrees of freedom than the space it exists in. This indicates that two of the equations may be removed to obtain a system of four equations with four unknowns. As argued in chapter 2.2, the lack of joints at the end effector makes its orientation dependent on its position. A similar case is seen here, as we have full control over the linear velocities, while (some of) the angular velocities depend on the linear ones and cannot be chosen freely. Thus the linear velocities are kept in the system of equations, and it is rewritten to $J\dot{q} = [v_x, v_y, v_z, ?]$ where the ? is to be determined. Using ω_z is not possible, as this value is determined from the linear velocities. Using ω_x or ω_y would be wrong since the rotation of the end effector is a combination of both. For this reason, the z-component of the *temporal rate of the x-axis*, $\dot{x}$ is used instead, as combines ω_x and ω_y into one value $\dot{x}_z = R_{yx}\omega_x - R_{xx}\omega_y$. The Jacobian of $\dot{x}_z$ is computed using the Jacobian of ω_x and ω_y, which can be extracted directly from the Jacobian computed in section 2.4, i.e.

$$J_{\dot{x}} = R_{yx}J_{\omega_x} - R_{xx}J_{\omega_y} \tag{2.17}$$

This is used to construct the pseudo-Jacobian, where the first three rows are identical to the original Jacobian, while the fourth row is equal to eq (2.17).

$$J_{\text{pseudo}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & R_{yx} & -R_{xx} & 0 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} \{J_{v_x}\} \\ \{J_{v_y}\} \\ \{J_{v_z}\} \\ \{J_{\omega_x}\} \\ \{J_{\omega_y}\} \\ \{J_{\omega_z}\} \end{bmatrix}}_{J} \qquad \text{and} \qquad J^+ = J_{\text{pseudo}}^{-1} \tag{2.18}$$

where J is intentionally written out to highlight how R_{yx} and R_{xx} are multiplied onto J. The joint velocities are then obtained using

$$\dot{q} = J^+ \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ \dot{x} \end{bmatrix} \tag{2.19}$$

3 Trajectory Planning

This section covers the planning of a trajectory that approximates the circular path from Section 2.3.

3.1 Interpolation Polynomials

The trajectory will initially be planned along five knot-points (φ_0, φ_9, φ_{18}, φ_{27}, φ_{36}) which are described by their corresponding configurations, listed in Table 2.1.

The trajectory is to be planned so that the robot's end-effector can traverse each of the four segments in 2 seconds. The path along each segment, for each joint, is to be approximated by a polynomial of the form:

$$q_x(t) = A_{ij}t^5 + A_{ij}t^4 + A_{ij}t^3 + A_{ij}t^2 + A_{ij}t + A_{ij} \tag{3.1}$$

To simplify calculations, it can be assumed that there is zero joint acceleration at the knot points, i.e. $\ddot{q}_{1,2,3,4} = 0\,\mathrm{rad\,s^{-2}}$. Furthermore, at all knot points except φ_0 and φ_{36}, the end effector should have a velocity of $27\,\mathrm{mm\,s^{-1}}$. This means the robot's joint angles, joint velocities, and joint accelerations are known for every knot point. Realizing this, the coefficients of the polynomial in Equation 3.1 can be solved linearly as a system of 6 equations with 6 unknowns. This can be repeated for the remaining joints and segments. To solve this system of equations, two matrices are built (shown below), where matrix B contains the known information about a given robot joint and matrix A contains the 5th-degree polynomial as well as its first and second derivative for the start and end point of a given segment.

$$A = \begin{bmatrix} t_{in}^5 & t_{in}^4 & t_{in}^3 & t_{in}^2 & t_{in} & 1 \\ t_{out}^5 & t_{out}^4 & t_{out}^3 & t_{out}^2 & t_{out} & 1 \\ 5t_{in}^4 & 4t_{in}^3 & 3t_{in}^2 & 2t_{in} & 1 & 0 \\ 5t_{out}^4 & 4t_{out}^3 & 3t_{out}^2 & 2t_{out} & 1 & 0 \\ 20t_{in}^3 & 12t_{in}^2 & 6t_{in} & 2 & 0 & 0 \\ 20t_{out}^3 & 12t_{out}^2 & 6t_{out} & 2 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} q_{i,t=0} \\ q_{i,t=2} \\ \dot{q}_{i,t=0} \\ \dot{q}_{i,t=2} \\ \ddot{q}_{i,t=0} \\ \ddot{q}_{i,t=2} \end{bmatrix} \tag{3.2}$$

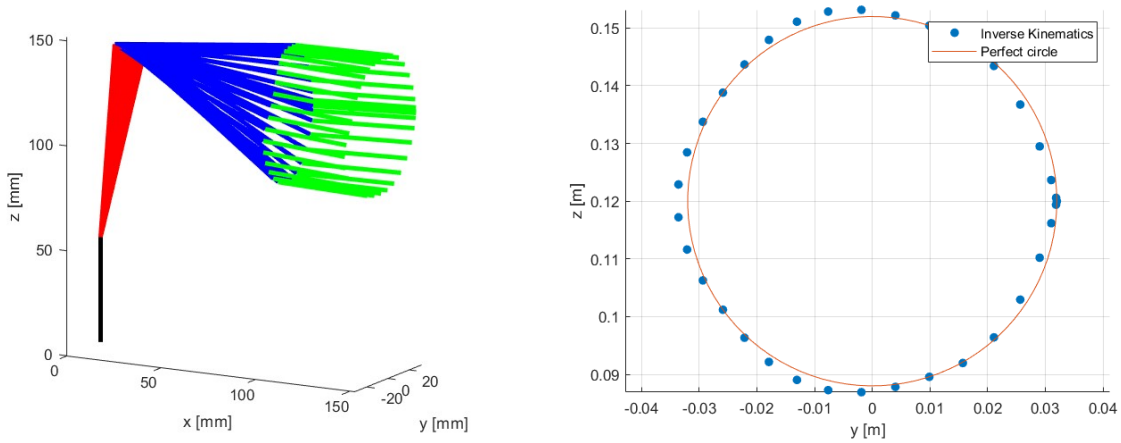
where $t_{in} = 0$ and $t_{out} = 2$. As previously mentioned, the accelerations at the knot points are zero, therefore $\ddot{q}_{i,t=0} = \ddot{q}_{i,t=2} = 0\,\mathrm{rad\,s^{-2}}$. The joint angles at the beginning and end of each segment ($q_{i,t=0}$ and $q_{i,t=2}$) have been found in Section 2.3 and are presented in Table 2.1.

Solving the system of 6 equations for all four joints, for each of the four segments, yields the coefficients listed in Table 3.1.

Table 3.1: Interpolation Coefficients for a fifth order polynomial for all four segments (A, B, C, D).

A_{i5}	A_{i4}	A_{i3}	A_{i2}	A_{i1}	A_{i0}
-0.0057	0.0395	-0.0827	0.0000	0.0000	0.2102
0.0113	-0.0565	0.0753	0.0000	0.0000	1.4302
0.0524	-0.2622	0.3496	0.0000	0.0000	-1.6699
-0.0637	0.3187	-0.4249	0.0000	0.0000	0.2397
B_{i5}	B_{i4}	B_{i3}	B_{i2}	B_{i1}	B_{i0}
-0.0057	0.0170	0.0073	0.0000	-0.1800	0.0000
0.0017	-0.0041	-0.0060	0.0000	0.0000	1.4904
-0.0113	0.0700	-0.1300	0.0000	0.0000	-1.3903
0.0096	-0.0659	0.1360	0.0000	-0.0000	-0.1002
C_{i5}	C_{i4}	C_{i3}	C_{i2}	C_{i1}	C_{i0}
0.0057	-0.0395	0.0827	0.0000	-0.0000	-0.2102
-0.0075	0.0329	-0.0323	0.0000	-0.0693	1.4302
-0.0050	0.0111	0.0217	0.0000	-0.2200	-1.6699
0.0124	-0.0441	0.0106	0.0000	0.2889	0.2397
D_{i5}	D_{i4}	D_{i3}	D_{i2}	D_{i1}	D_{i0}
0.0057	-0.0170	-0.0073	0.0000	0.1800	-0.0000
0.0204	-0.1022	0.1363	0.0000	-0.0000	1.3212
0.0461	-0.2307	0.3077	0.0000	-0.0000	-1.9160
-0.0666	0.3329	-0.4439	0.0000	0.0000	0.5949

3.2 Endeffector Path & Tuning



(a) Robot Configurations

(b) End-effector path

Figure 3.1: Plots of the robot following the path of interpolated points with a timestep of 0.2 s

The next point of study is investigating how well the trajectory defined by the coefficients from Table 3.1 approximates the circle. The positions of the robot and its end effector have been plotted every 0.2 s in Figure 3.1.

The overall robot configurations shown in Figure 3.1a still look reasonable. The trajectory of the end effector (Figure 3.1b) also seems to follow the circle reasonably well. There are, however, some deviations where the robot overshoots the desired path in the left half of the circle and undershoots it on the right. The path also does not align very well at the start and end points. This is caused by the initial velocity being $[0, 0, 0]^T \text{mm}\,\text{s}^{-1}$.

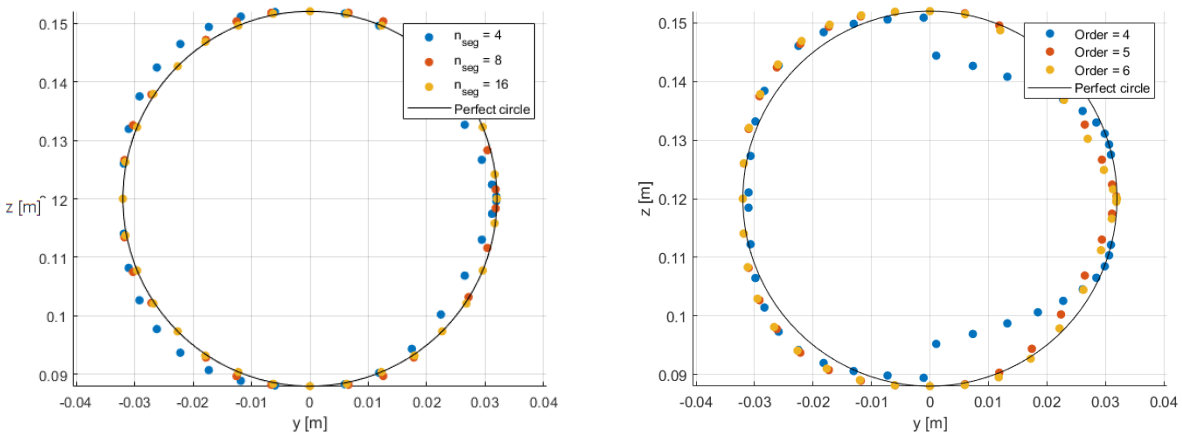
Attempting to improve the trajectory

To improve the trajectory, it is possible to vary the interpolation function and/or the number of knot points. Different interpolation functions, i.e. higher or lower-order polynomials, will limit the effects of potential over or under-fitting. Meanwhile, increasing the number of knot points will limit the impact of a poor fit on the overall trajectory of the end effector.

Both of the aforementioned improvement strategies have been tested, and their effects on the trajectory of the end effector have been plotted in Figure 3.2. As expected, increasing the number of knot points significantly improves the end effector path. Going up to 10 knot points results in a near-perfect approximation of the circle. Theoretically, there is no upper limit to how many knot points there can be; increasing their number will always positively affect the trajectory. It should be mentioned that, eventually, the improvements will be negligible and come at a great computational cost.

Improving the trajectory by tuning the interpolation function is more difficult. Figure 3.2b shows the path of the end effector for a 4th, 5th and 6th order polynomial. Clearly, neither increasing nor decreasing the number of coefficients improves the trajectory. Increasing the polynomials order past six means that one risks over-fitting. Decreasing results in an unacceptable drop in resolution. The "disconnect" that happens with the 4'th order polynomial at the top and bottom of Figure 3.2a is attributed to the fact that the polynomial can only satisfy 5 of the 6 constraints, where the z-value is the non-satisfied one.

Therefore, the ideal parameters for planning a trajectory along a circle using the 4 DOF robot are 10+ knot points with a 5th-order polynomial.



(a) Varying Knot Points with 5'th order polynomial.

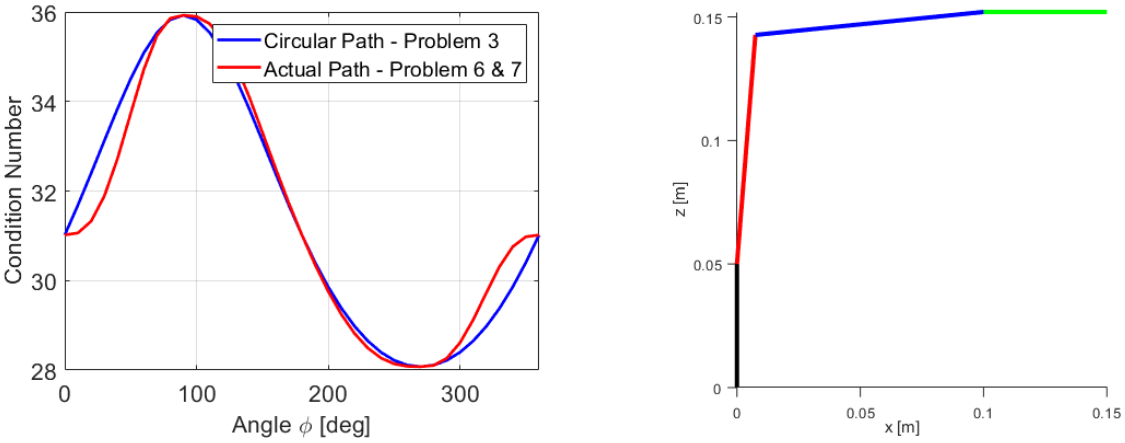
(b) Varying Polynomial Order with 4 segments.

Figure 3.2: Tuning the number of knot points and polynomial order.

4 Singularities & Statics

4.1 Condition number of the Jacobian Matrix

Figure 3.1 illustrates the configuration number of the end-effector Jacobian matrix of the robot manipulator as it follows two different paths: 1. The ideal circular path (from Problem 3), shown in blue and 2. The Actual Path (from Problems 6 & 7) is shown in red.



(a) Condition number of the Jacobian matrix for the Robot's theoretical circular trajectory vs the real path

(b) Robot configuration at highest condition number, $\varphi = 90$ [deg]

Figure 4.1: Condition number and robot configuration when tracking circle

The condition number of the Jacobian matrix gives insight into how sensitive the manipulator's movements are to control inputs. High peaks in the condition number correspond to configurations where the manipulator

is closer to a singularity. In these positions, small changes in joint angles can result in large changes in end-effector velocity or force, indicating lower control precision.

The Ideal path has a peak at $\varphi = 90$ [deg], corresponding to the point at which the robot arm is approaching singularity. The robot configuration is illustrated in figure 4.1b.

As expected, this occurs when two consecutive links (links 3 and 4) are nearly collinear. This configuration decreases the manipulators' ability to move precisely in any desired direction.

Whilst the high condition numbers of the Jacobian indicate the robot is approaching singularity, the rank remains consistently 4 along the entire path, indicating the system never fully reaches singularity for the given trajectories.

4.2 Static Joint Torques

The static torque at a given point, assuming that a weight of 1 N is acting on the end effector, can be calculated using Equation 4.1.

$$\tau = J^T(q)F \tag{4.1}$$

where $F = [0, 0, -1, 0, 0, 0]^T$N. The joint torques as the end effector tracks the circle can be calculated as a series of 36 static loads corresponding to the configurations from Table 2.1. The torques have been plotted in Figure 4.2.

All torque curves appear physical.

1. **Torque for Joint 1**: The torque is constant and zero. This makes sense since the force is acting in the z-direction and can, therefore, not produce a torque about the axis of the joint. , see Figure 2.3.
2. **Torque for Joint 2**: The torque varies slightly around $-0.15\,\mathrm{N\,m}$. Fluctuations in torque are expected from this joint since it, together with Joint 3, controls the vertical motion of the end effector. The torque is caused by the joint having to counteract gravity (i.e. the simulated load in the $-z$ direction). Comparison with Figure 2.3 shows that the joint angle does not change quadrant, meaning the torque should remain either positive or negative.
3. **Torque for Joint 3**: The analysis for Joint 2 applies here.
4. **Torque for Joint 4**: The torque of the joint is constant and negative. This makes sense since the orientation of the end effector is kept constant.

It should be noted that this approach is fundamentally flawed for a moving system. The calculation neglects the mass of the robots' joints, and since these calculations are static at every evaluation point, inertia is also not accounted for. These calculations are revised in Section 5.1.

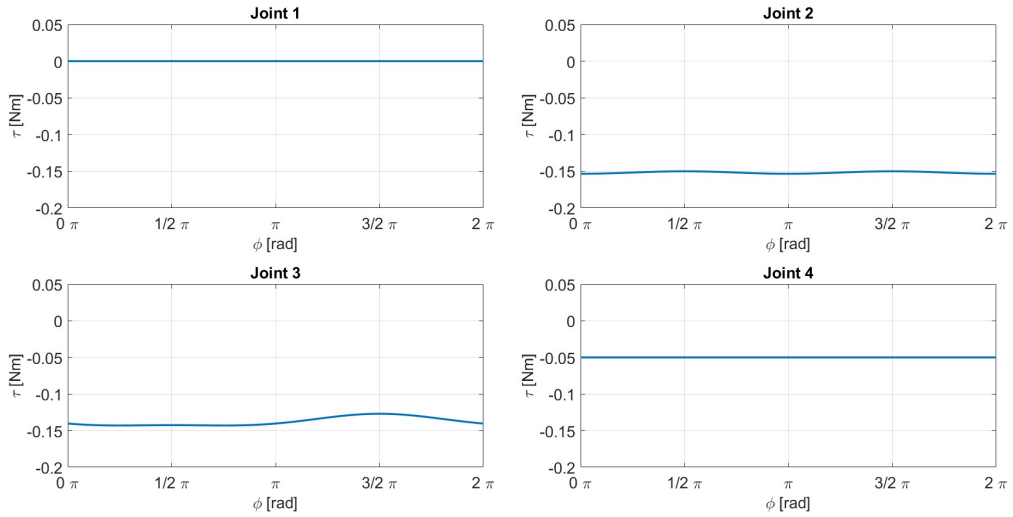


Figure 4.2: Joint Torques along circular path from Table 2.1.

5 Dynamics

5.1 Dynamic Joint Torques

To calculate the dynamic joint torques, a dynamic system is needed. The mass of link 1 is 0.06 kg and its matrix of inertia of the centre of mass frame 1 (CM1):

$$\bar{D_1} = \begin{bmatrix} I_0 & 0 & 0 \\ 0 & 0.4I_0 & 0 \\ 0 & 0 & 0.9I_0 \end{bmatrix} \tag{5.1}$$

For this, we need to first estimate the unknown I_0 by approximating the matrix of inertia of the first link by the one of a box. For this, we used the matrix of inertia of a box:

$$I_1 = \frac{m_1}{12} \begin{bmatrix} b^2 + c^2 & 0 & 0 \\ 0 & c^2 + a^2 & 0 \\ 0 & 0 & a^2 + b^2 \end{bmatrix} \tag{5.2}$$

with the given values a = 0.033 m, b = 0.051 m, c = 0.024 m and compare them with the given matrix of inertia. This gives $I_0 \approx 1.584 \times 10^{-5}$.

Furthermore are, the masses of link 2 and 3 given to be 0.08 kg, the mass of link 4 to 0.04 kg and the matrices of inertia, respectively in the frames CM2, CM3, CM4:

$$\bar{D_2} = \bar{D_3} = \begin{bmatrix} 0.45I_0 & 0 & 0 \\ 0 & 1.4I_0 & 0 \\ 0 & 0 & 1.2I_0 \end{bmatrix}$$

$$\bar{D}_4 = \begin{bmatrix} 0.5I_0 & 0 & 0 \\ 0 & 0.5I_0 & 0 \\ 0 & 0 & 0.5I_0 \end{bmatrix}$$

With this, the inertia matrices and Christoffel symbols are computed. While calculating the transformation matrices in this case, it must be noted that the centres of mass do not correspond to the origins of the coordinate systems. Therefore, the coordinate transformations need to be adapted. The general form of the equations of motion is

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau. \tag{5.3}$$

with the inertia matrix of the manipulator

$$D(q) = \sum_{i=1}^{n} m_i J_{v,c_i}^T J_{v,c_i} + J_{\omega_i}^T R_i I_i R_i^T J_{\omega_i} \tag{5.4}$$

and the matrix computed by the Christoffel symbols

$$[C(q,\dot{q})]_{kj} = \sum_{i=1}^{n} c_{ijk}(q)\dot{q}_i = \sum_{i=1}^{n} \frac{1}{2}\left(\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} + \frac{\partial d_{ij}}{\partial q_k}\right)\dot{q}_i \tag{5.5}$$

and

$$g(q) = \begin{bmatrix} g_1(q) \\ ... \\ g_n(q) \end{bmatrix} \tag{5.6}$$

where

$$g_k = \frac{\partial P}{\partial q_k} \tag{5.7}$$

The torques for the trajectory of section 3 are:

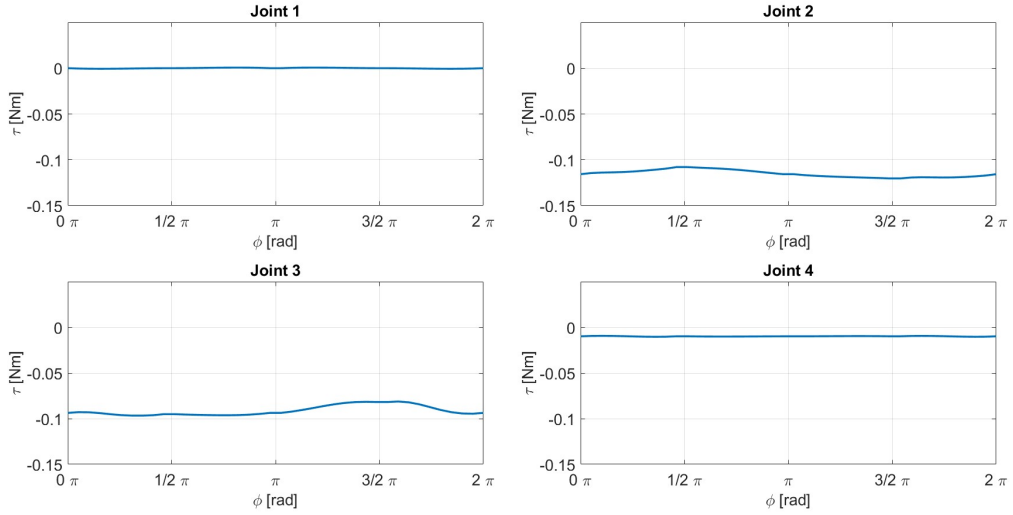


Figure 5.1: Joint Torques along the interpolated path from Section 3.2.

Those torques are in the same range as the static ones in Figure 5.1, which makes sense as the mass of the links will produce a force that is a bit smaller than the 1 N from Section 4.2, but we also have to consider the moments of inertia here.

6 Computer Vision and Control

This chapter covers the group's approach to programming a simple 4 DOF, AX-12A servo motor-based robot arm. The aim was to develop a complex robotic system based on the theory from Sections 2-5. The Robotic arm developed for this project was named 'Roberta' by the group. Throughout this report, the robot will be referred to by this name for clarity and consistency.

Selected problem: Recognize the keys of a keyboard that are placed in front of it and type: "Hello World". We have chosen to increase the challenge by creating the possibility of typing any word of the user's choosing.

6.1 Motivation for Roberta

With ever-improving technology, the precision and capabilities of modern robots are nearly limitless. Recent advances in artificial intelligence and computer vision have also unlocked new possibilities for semi-autonomous robots. With the robot used in this course, we can hardly compete with industry giants like ABB and Boston Dynamics, but the core principles in robot control can still be explored to a large extent. A "simple" task, such as typing words out on a keyboard, presents many of the same challenges as more advanced tasks. Our motivation for programming Roberta to type on a keyboard is that it reveals key insights into control, dynamics, path planning, and computer vision in a tangible and entertaining way.

6.2 Description of Setup

The setup of our chosen robotic system involved two laptops, an external keyboard and the 4 DOF, AX-12A servo motor-based robot arm. One of the laptops ran the entire Matlab file and was connected to the robot arm via a USB cable to transfer commands. Due to the limited reach of the robot arm, an external keyboard was used to ensure all necessary keys could be reached. This keyboard was then connected to a second laptop, displaying Roberta's typed message. In order to implement a similar program without using an external keyboard, the base of the robot arm would need to be fixed over the laptop's touchpad. For a clearer setup and to avoid the risk of damaging our personal devices, we opted not to type on them directly and instead chose the external keyboard set-up.

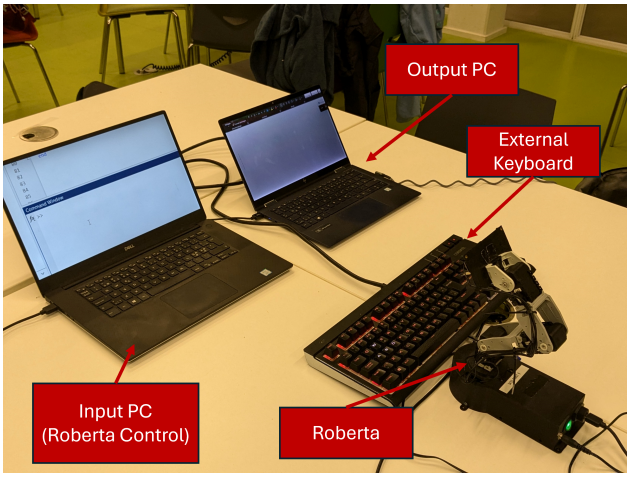


Figure 6.1: Real-world setup of robot system

6.3 Robot Arm Control

Generating path from key coordinates: The end effector follows a simple path but with a few details to be noted. The pattern is:

Press current key -> move above current key -> move above next key -> press next key

As seen above, there are two intermediate steps between each key press, so n key presses require $n+2(n-1)$ steps. The "move above ... key" is achieved by offsetting in the z-direction, i.e. if the i'th key has coordinates $[x_i, y_i, z_i]^T$, then the position above the key is simply $[x_i, y_i, z_i]^T + [0, 0, h_{offs}]^T$ where h_{offs} is the vertical offset.

Roberta always actuated the 4'th joint last, resulting in incorrect key presses, see figure 6.2. When Roberta moves down to press a key, the 2'nd and 3'rd joint actuate first, while the 4'th joint maintains its angle, causing it to hit the wrong coordinates. It then actuates the 4'th joint, thus hitting the key from the side. To fix this, we pre-rotated joint 4 before actuating joint 2 and 3, yielding the dashed line in the middle panel in figure 6.2.

Aside from the key presses and intermediate steps, the robot is programmed to start and end in its "home" position, pointing straight up.

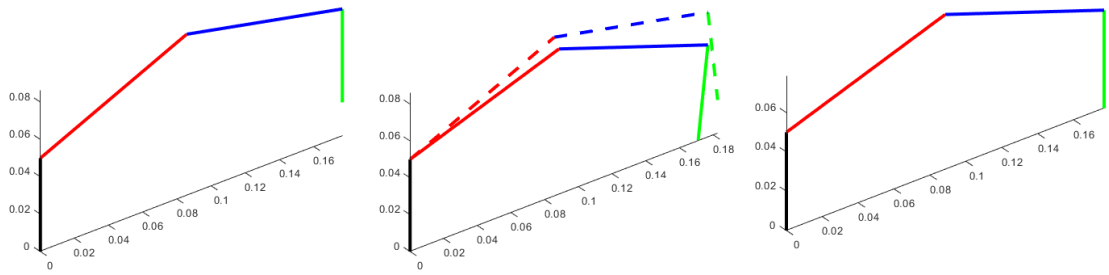


Figure 6.2: Three robot positions representing (left to right) above key, attempting to press key [solid] / adjusting q_4 (dashed), pressing key

Running Roberta

The Dynamixel servos are controlled by the DynamixelSDK library. Using a specialized function, Roberta iterates through a matrix of joint angles Q computed as described above.

$$Q = \begin{bmatrix} q_{1,1} & q_{1,2} & q_{1,3} & q_{1,4} \\ q_{2,1} & q_{2,2} & q_{2,3} & q_{2,4} \\ \vdots & \vdots & \vdots & \vdots \\ q_{n,1} & q_{n,2} & q_{n,3} & q_{n,4} \end{bmatrix} \tag{6.1}$$

This is set up in a dynamic manner, where the robot will go from joint configuration to joint configuration until it has iterated through every position in the matrix. This means that it can theoretically type texts of infinite length.

The dynamixel servos have a resolution of 0.29°. Since micro-stepping is beyond the scope of this project and offers an unnecessarily high level of control, the conversion between joint angles and steps is performed as follows:

$$rot = q \cdot \frac{180}{\pi} \cdot \frac{1}{0.29} \tag{6.2}$$

The servos are active until the positional error, calculated as the subtraction of the current position from the desired position, is below 2°. Roberta then transitions to the next set of joint instructions in the matrix.

6.4 Camera and Computer Vision

To establish the relationship between the coordinates of the keyboard keys and the robot arm, we needed to accurately map the keyboard keys and their placement in a coordinate system, as well as estimate the robot's pose and its relation to the keyboard key coordinates.

To achieve this, we used Matlab in combination with a Python script. Since we used Matlab to move the robot, it also made sense to handle the camera and computer vision there. However, Matlab's computer vision handling is more limited, so for this, we used a Python script within Matlab by passing variables from Matlab to Python and then running the script via Matlab, performing the computer vision tasks, and then returning the calculated coordinates in Matlab.

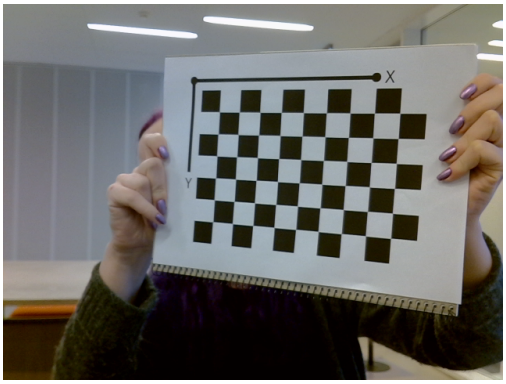
Camera calibration

Camera Calibration involves the mapping of image coordinates to real-world coordinates and is a crucial step in computer vision. For Roberta to accurately type on a keyboard, the system must:

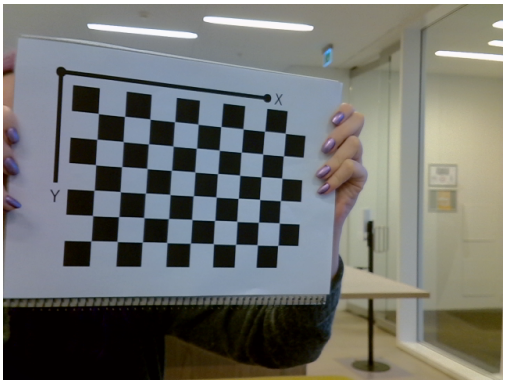
- Correct lens distortions that can cause image warping
- Calculate real-world coordinates (x,y,z) from 2D points
- Provide parameters to project 3D points on image plane

Zhang's model was used for the camera calibration to correct camera distortion and later get a pose estimation. This method utilizes a checkerboard to calibrate a camera by taking a 2D object - a plane with a

printed checkerboard - and taking multiple pictures (10-20) of the checkerboard in many different positions. Either the camera or the checkerboard must remain fixed while the other moves. For our tests we tried it both ways.

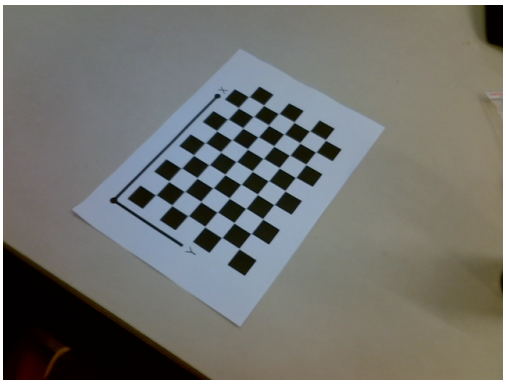


(a) Fixed Camera



(b) Fixed Camera

(c) Fixed checkerboard

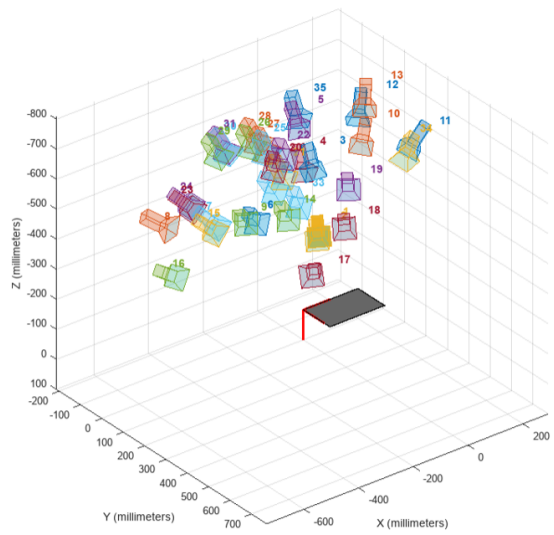


(d) Fixed checkerboard

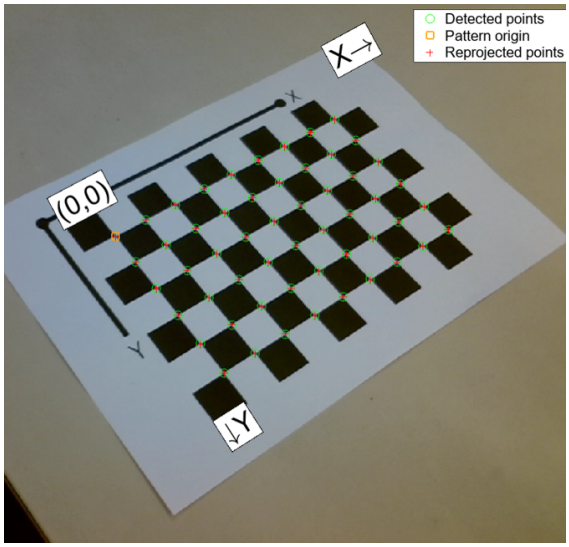
Figure 6.3: a) and b) is where the camera is fixed. c) and d) is where the checkerboard is fixed.

Each time the checkerboard (or camera) is moved, the board gets a new rotation and translation relative to the camera. This allows us to define a coordinate system where all of our points on the checkerboard plane are z=0.

Matlab's camera calibration app, which also uses Zhang's model, identified the corner correspondences between the images, which gives us points in the 2D image plane and their known 3D points of the checkerboard in the world coordinate system.



(a) The checkerboard fixed calibration visualization



(b) The detected points, pattern origin and re-projected points on an image used for calibration

Figure 6.4: (a) and (b) show the checkerboard calibration process.

From this, a homography matrix H will be computed for each image. These can then be used to estimate the camera's intrinsic parameters, the calibration matrix K, and the extrinsic parameters (R,t), which comprise the camera matrix P.

$$P = K[R \mid t] \tag{6.3}$$

The calibration matrix consists of all our intrinsic parameters (which is why it's also known as the camera intrinsic matrix), which are the camera's internal characteristics, which are focal lengths fx and fy, s for skew (between camera axes), and cx and cy is the principal point

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

The extrinsic parameters R and t, the rotation matrix and the translation vector

$$[R \mid t] \tag{6.5}$$

One thing to note, however, is that since this relies on homography, the camera matrix is not scaled. We will address how this issue is resolved in a later section.

Model training

A dataset was downloaded from Kaggle, which had about 5000∼ images of keyboards, with the entire keyboard annotated as one class, which we removed since they were unnecessary.

Using Roboflow, we created and trained a custom object detection models, we trained 4 in Roboflow 3.0 Object Detection architecture and 1 in Yolo v11, since both are relatively light models to run.

To prepare the dataset, we manually annotated around **550** images of different keyboards. We defined 41 classes we deemed necessary for typing sentences on a keyboard, which were

- All letters in the English alphabet
- Numbers (0-9)
- Space bars, commas, full stops and backspace

Figure 6.5: The annotation proccess

The images were reprocessed to 640x640 px since this is the standard for most models (including Roboflow and Yolo). For our Roboflow v4 and Yolo model, we used 66% images for training, 20% for validation and 14% for testing.

The training set is used to train the model through iterative learning, during this training process the validation images are used to evaluate and adjust the model. Then, the testing set is used to access the model's performance.

The model(s) was then deployed and can be accessed for our program through an API. The Yolo model has a 97.3% mAP (mean average precision), and the Roboflow 4th version has a 97%. mAP is used to analyze the accuracy of an object detection model.

Figure 6.6: The YOLO model used on a keyboard, with the percentages being how confident it is that what it detected belongs to that class

Transforming camera coordinates to real coordinates

To fix the scaling problem and give the keyboard "plane" a point of origin, we implemented AruCo codes. AruCo codes are simplified QR codes and are used for pose estimation in computer vision.

Using the OpenCV library, we detect the AruCo markers, their corresponding IDs and their corner location in the image plane. Since we know the physical size of our Aruco marker (20 mm in this case), we use the top left corner of AruCo 0 as an origin point (0,0,0) for the real-world coordinate system. By treating the keyboard as a plane, all key coordinates are 0 on the plane.

We can then calculate the scaling factor (pixels per mm) based on the AruCo corners in the image by comparing the length between the corners in pixels in the image plane to our known length of 20 mm. This scaling factor allows us to map image pixel coordinates to the real world with the proper scale.

To estimate the camera's pose (which also is the robot's tip), we treat the AruCo marker corners as known points in the coordinate system, the more points the better accuracy, since we need at least four points for a reliable estimation.

Finally, we transform the world coordinate system (defined by the AruCo marker and keyboard plane) into the robot coordinate system. This transition lets us know what coordinates the robot needs to move to, to press a given key.

We were able to do all this except estimating the camera pose and transforming the world coordinate system into the robot coordinate system.

7 Individual contributions

Every team member made equal contributions to the project, fostering a collaborative environment where all development tasks were coordinated collectively. As a result, the work produced by the team as a whole is seamless and lacks any discernible distinctions in individual contributions. The main focus of each of the group's members is described below in percentages:

		Problem										
		1	2	3	4	5	6	7	8	9	10	11
Ava	s242184	16.6			33.3	50.0			33.3	30.0		16.0
Caroline	s242182	16.6			33.3	50.0			33.3	30.0		16.0
Louise	s173937	16.6	33.3		33.3		30.0		33.3	18.0		20.0
Moritz	s223293	16.6		50.0			30.0	50.0		22.0		16.0
Simon	s193962	16.6	33.3	50.0			20.0	50.0				16.0
Sophie	s242180	16.6	33.3				20.0				100.0	16.0

We Group 3, declare that this submission is our own work and that we have adhered to the academic integrity standards outlined by DTU. Additionally, we agree with the individual contributions stated in the table above.

Ava Bennett (s242184): _____

Caroline Lange (s242182): _____

Louise Sternholdt-Sørensen (s173937): _____

Moritz Genzwürker (s223293): _____

Simon Therkildsen (s193962): _____

Sophie Rücker (s242180): _____