ELEC0129 - Introduction to Robotics

Final Report - Team 6

Nandini Chavda Yinan Chen Xiaocheng Sun Matthew Hoult

Table of Contents

I – Introduction

This report is a detailed account of progression through the ELEC0129 module, and a document to the mathematics and programming required for operation of everyday robots. The report follows the building of the robot from scratch and culminates in a final robot, capable of pick and place operations via the use of forward kinematics, inverse kinematics, and trajectory planning. This report covers all the work done by the team (team 6) to derive the coding required for the final robot operation.

II – Forward Kinematics

Figure 1 (self-sourced) - This figure shows how the lengths of the links of the robot were measured

Forward kinematics is the mathematics involved in taking input joint angles of a robot and giving a 3D cartesian co-ordinate for the end effector of the robot, giving an accurate position and orientation of the end effector. This works by utilising the Denavit-Hartenburg parameters to create numerous transformation matrices to work out the final position of the end-effector with respect to the base frame. The Denavit-Hartenburg parameters use the known lengths and angles of the various links of the robot to derive the transformation matrices responsible for calculating the end effector position. To begin with, the lengths of the links must be known. Figure 1 shows how the links were measured.

1. Link Measurement

In figure 1, the red line along the yellow robot link represents the specified distance L_1, which was measured to be 94mm (an average of three measurements for accuracy). The distance indicated by the green line is the length of the second robot link, measured in the same way as the second link. The length of this second link, L_2, was found to be 180mm. Now that the link lengths had been established, the Denavit-Hartenburg parameters could be derived, and the diagrams could be drawn. The diagram formed the start of the Denavit-Hartenburg parameter deviation as it is important to have a visual reference of the robot whilst calculating the parameters. Figures 2.1 through 2.4 show how the diagram was set up.

2. Drawing Steps

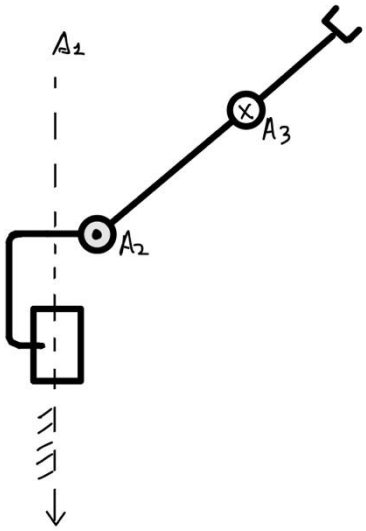


Figure 2.1: The axes and joints of the robot. (step 1)

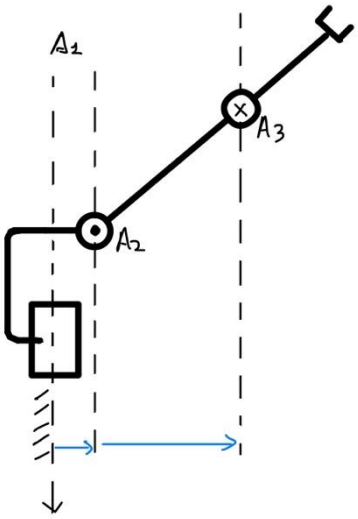


Figure 2.2 The mutual perpendicular lines between axes are drawn. (step 2)

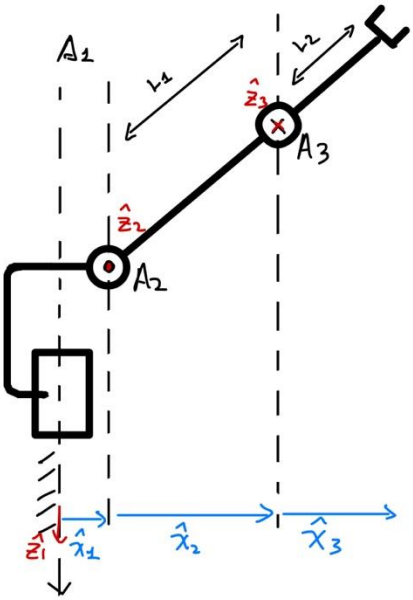


Figure 2.3: Frames are attached to the robot from [1] to [n-1]. (step 3)

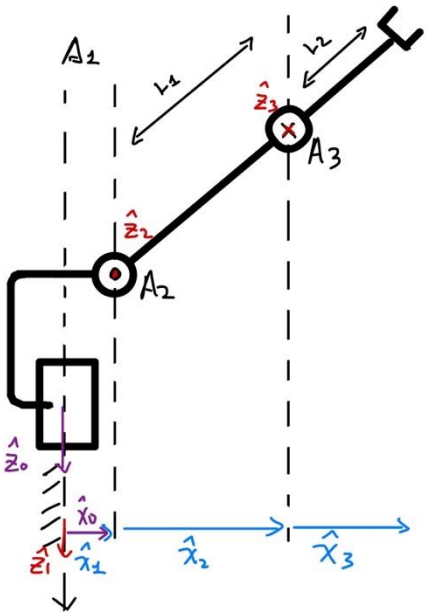


Figure 2.4: The [n] frame and [0] frames are attached. (step 4)

Once the drawing was accurate, they were able to move onto the final Denavit-Hartenburg derivations.

3. DH Parameter Derivation

Firstly, the matrix for link lengths needed to be derived, a_{i-1}. But the formula and definition of a_{i-1} must be evaluated:

Definition of a_{i-1}:

a_{i-1} is the distance from $\overline{Z}_{i-1}$ to $\overline{Z}_i$, along the $\overline{X}_{i-1}$ axis

Therefore, the other link lengths can be determined and evaluated as such:

a_0 is the distance from $\overline{Z}_0$ to $\overline{Z}_1$, along the $\overline{X}_0$ axis $\quad \therefore a_0 = 0$

a_1 is the distance from $\overline{Z}_1$ to $\overline{Z}_2$, along the $\overline{X}_1$ axis $\quad \therefore a_1 = L_1 = 94$ mm

a_2 is the distance from $\overline{Z}_2$ to $\overline{Z}_3$, along the $\overline{X}_2$ axis $\quad \therefore a_2 = L_2 = 180$ mm

The assumption was made that the link length from Joint 1 to Joint 2 is negligible because the distance is too small to measure accurately.

Once the link lengths have been defined, the link twists needed to be derived, α_{i-1}, which can be defined as follows:

Definition of α_{i-1}:

α_{i-1} is the angle between $\overline{Z}_{i-1}$ to $\overline{Z}_i$, about the $\overline{X}_{i-1}$ axis

Therefore, the other link twists can be determined and evaluated as such:

α_0 is the angle between $\overline{Z}_0$ to $\overline{Z}_1$, about the $\overline{X}_0$ axis $\quad \therefore \alpha_0 = 0$

α_1 is the angle between $\overline{Z}_1$ to $\overline{Z}_2$, about the $\overline{X}_1$ axis $\quad \therefore \alpha_1 = -90\ deg$

α_2 is the angle between $\overline{Z}_2$ to $\overline{Z}_3$, about the $\overline{X}_2$ axis $\quad \therefore \alpha_2 = 180\ deg$

After the link twists were defined, the link offsets d_i, needed to be determined next. The definition of the link offsets is as follows:

Definition of d_i:

d_i is the distance from $\overline{X}_{i-1}$ to $\overline{X}_i$, about the $\overline{Z}_i$ axis

Therefore, the other link offsets can be determined and evaluated as such:

d_0 is the distance from $\overline{X}_0$ to $\overline{X}_1$, about the $\overline{Z}_0$ axis $\quad \therefore d_0 = 0$

d_1 is the distance from $\overline{X}_1$ to $\overline{X}_2$, about the $\overline{Z}_1$ axis $\quad \therefore d_1 = 0$

d_2 is the distance from $\overline{X}_2$ to $\overline{X}_3$, about the $\overline{Z}_2$ axis $\quad \therefore d_2 = 0$

Finally, the last Denavit-Hartenburg parameter needing to be defined was the joint angles themselves, which were defined as θ_i:

Definition of θ_i:

θ_i is the distance from $\overline{X}_{i-1}$ to $\overline{X}_i$, about the $\overline{Z}_i$ axis

Therefore, the other angles can be determined and evaluated as such:

θ_0 is the distance from $\overline{X}_0$ to $\overline{X}_1$, about the $\overline{Z}_0$ axis $\quad \therefore \theta_0 =$ variable

θ_1 is the distance from $\overline{X}_1$ to $\overline{X}_2$, about the $\overline{Z}_1$ axis $\quad \therefore \theta_1 =$ variable

θ_2 is the distance from $\overline{X}_2$ to $\overline{X}_3$, about the $\overline{Z}_2$ axis $\quad \therefore \theta_2 =$ variable

Now that all the Denavit-Hartenburg parameters have been derived, they put the results into a DH table to accurately visualise all the parameters at once.

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	-90 deg	0	θ_2
3	L_1	180 deg	0	θ_3

It is important to note that these parameters can be used to kinematically describe any robot and that the parameters a_{i-1} and α_{i-1} are always constant no matter what joints the robot has. d_i is variable for a prismatic joint but a constant for a revolute joint. The θ_i parameter behaves oppositely to the d_i parameter, in that θ_i is constant for a prismatic joint and variable for a revolute joint. Neither d_i nor θ_i are necessarily 0 at any time, but it is always possible. Now that these parameters have been defined, the transformation matrices must be defined.

Each of the transformation matrices can be simplified into 4 sub-problems:

$$^{i-1}{}_{i}T = R_x(\alpha_{i-1}) \cdot D_x(a_{i-1}) \cdot R_z(\theta_i) \cdot D_z(d_i)$$

$$= \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & \cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & 0 \\ 0 & \sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \cos(\alpha_{i-1})\sin(\theta_i) & \cos(\alpha_{i-1})\cos(\theta_i) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\alpha_{i-1})\sin(\theta_i) & \sin(\alpha_{i-1})\cos(\theta_i) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix from frame [0] to frame [3]:

$$^{0}_{1}T = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{1}_{2}T = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_2 & -\cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}_{3}T = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_1 \\ -\sin\theta_3 & -\cos\theta_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix from frame [0] to frame [3]:

$$^{0}_{3}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T$$

Now that the transformation matrix formula has been found, the final transformation matrices with respect to each frame can be deduced.

$$
{}^{0}_{3}T = \begin{bmatrix} \cos(\theta_1)\cos(\theta_2) & -\cos(\theta_1)\sin(\theta_2) & -\sin(\theta_1) & 0 \\ \sin(\theta_1)\cos(\theta_2) & -\sin(\theta_1)\sin(\theta_2) & \cos(\theta_1) & 0 \\ -\sin(\theta_2) & -\cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
\cdot \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_1 \\ -\sin\theta_3 & -\cos\theta_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} \cos(\theta_1)\cos(\theta_2-\theta_3) & \cos(\theta_1)\sin(\theta_2-\theta_3) & \sin(\theta_1) & L_1\cos(\theta_1)\cos(\theta_2) \\ \sin(\theta_1)\cos(\theta_2-\theta_3) & \sin(\theta_1)\sin(\theta_2-\theta_3) & -\cos(\theta_1) & L_1\sin(\theta_1)\cos(\theta_2) \\ \sin(\theta_3-\theta_2) & \cos(\theta_2-\theta_3) & 0 & -L_1\sin(\theta_2) \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

The last step in calculating position of the end effector is multiplying the transformation matrix by the end effector position matrix, defined as:

$$
{}^{3}P = \begin{bmatrix} L_2 \\ 0 \\ 0 \end{bmatrix}
$$

$$
\therefore\ {}^{0}P = {}^{0}_{3}T \cdot \begin{bmatrix} L_2 \\ 0 \\ 0 \\ 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} L_2(cos(\theta_1)\,cos(\theta_2-\theta_3)) + L_1\,cos(\theta_1)\,cos\,(\theta_2) \\ L_2(sin(\theta_1)\,cos(\theta_2-\theta_3)) + L_1\,sin(\theta_1)\,cos\,(\theta_2) \\ L_2\,sin(\theta_3-\theta_2) - L_1 sin\,(\theta_2) \\ 1 \end{bmatrix}
$$

4. Checking the Calculations

To check the accuracy and the reliability of the calculations the calculated values were tested against the actual values.

To do this, the MATLAB code (see Appendix A) was used to enter 5 different angles for each Joint 1 to 3 to obtain the calculated values of the position of the gripper. The team decided to thoroughly investigate each joint individually, therefore, whilst varying the angle of one joint, the other two joints were kept at a constant angle which they chose to be 0 degrees. Therefore, the independent variable was the actual position of the gripper with respect to the base frame that was measured. And, the dependent variable was the joint angles.

The measurements were repeated 3 times and an average was taken. To ensure consistency in data, the same team member took measurements of the gripper with respect to the base frame to omit subjectiveness. A range is calculated from the three values measured for

the joint angles to further investigate the accuracy of the results. All values are given to 2 significant figures which was suitable given the precision of the ruler, ± 0.05 cm.

Results for Joint 1

Table 1: The measured position values of the gripper with respect to the base frame compared to the calculated position values from the MATLAB Code at each joint angle of joint 1.

	Calculated Position Values (m)			**Actual Position Values (m)**		
Joint Angles (°)	**x**	**y**	**z**	**x**	**y**	**z**
5	0.27	0.024	0.00	0.28 ± 0.02	0.065 ± 0.05	0.00 ± 0.01
45	0.19	0.19	0.00	0.18 ± 0.01	0.21 ± 0.02	0.00 ± 0.01
60	0.14	0.24	0.00	0.12 ± 0.02	0.24 ± 0.02	0.00 ± 0.01
90	0.00	0.27	0.00	0.00 ± 0.01	0.27 ± 0.01	0.00 ± 0.01
120	-0.14	0.24	0.00	-0.16 ± 0.01	0.23 ± 0.02	0.00 ± 0.01

Interpretation of results for joint 1

Overall, no anomalies were found in the data collected. And the actual position values were close to the calculated values.

However, some values showed significant variation. For example, the y-coordinate for the Joint angle 5 degrees the average actual position value was 4.1 cm away from the calculated position. This can be caused by human error when reading the measurement from the ruler. Looking at the image, when the robot is positioned at 5 degrees, it is evident that reading the length in the y direction can be difficult. This also can explain the larger range for this measurement, because the team struggled to obtain an accurate measurement with the ruler.

Another reason for smaller variations in the values could be that the joint itself is worn out by use, and therefore, the gripper is not moving to the desired position with the highest accuracy. In fact, the team did find that Joint 1 was not working properly because it was loosely attached to the motor and had to be replaced.

There is also a small range throughout all the z-coordinates which is most likely due to estimation errors.

Otherwise, the results do conclude that the calculations are correct.

Images of Joint 1

Images to document the results and for further analysis to supplement the data that was collected.

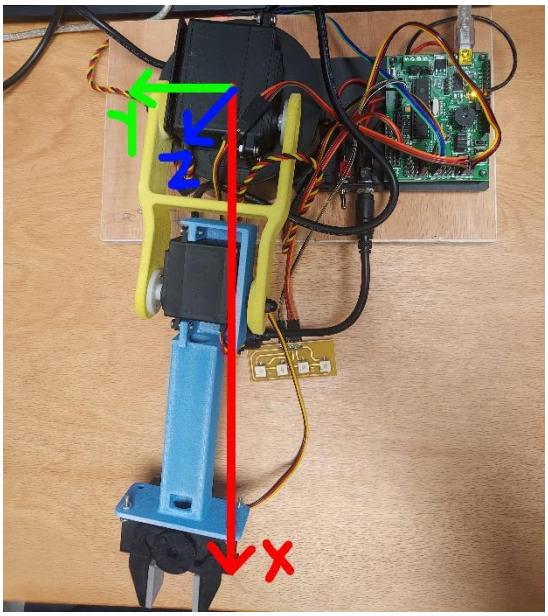


Figure 3.1: Joint 1 set to 5 degrees angle. Joint 2 and 3 are set to 0 degrees. The axes are annotated and remain the same for the rest of the images of Joint 1.

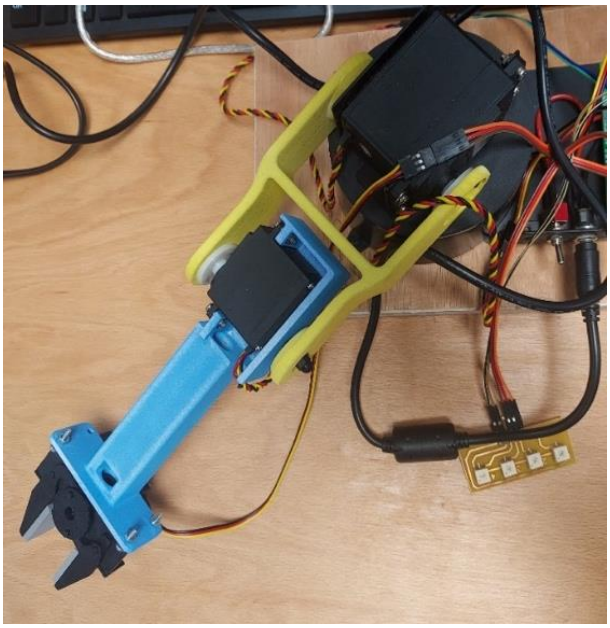


Figure 3.2: Joint 1 set to 45 degrees angle. Joint 2 and 3 are set to 0 degrees.

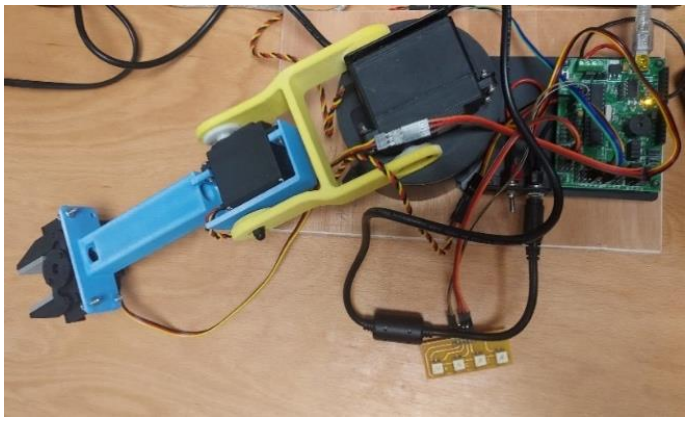


Figure 3.3: Joint 1 set to 60 degrees angle. Joint 2 and 3 are set to 0 degrees.

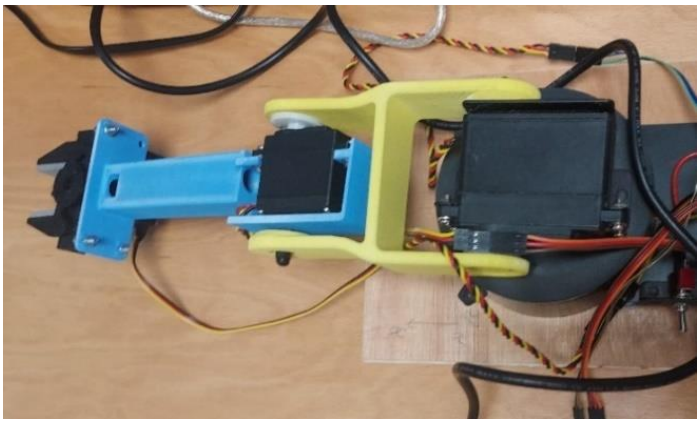


Figure 3.4: Joint 1 set to 90 degrees angle. Joint 2 and 3 are set to 0 degrees.

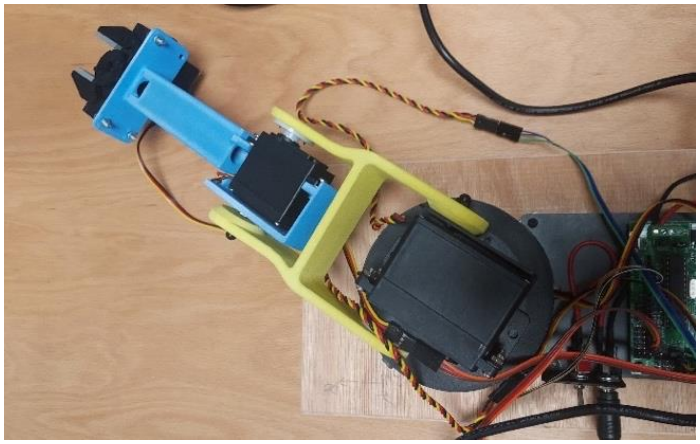


Figure 3.5: Joint 1 set to 120 degrees angle. Joint 2 and 3 are set to 0 degrees.

These figures show the various angular testing positions of joint 1. These pictures confirmed that joint 1 is behaving as predicted.

Results for Joint 2

Table 2: The measured position values of the gripper with respect to the base frame compared to the calculated position values from the MATLAB Code at each joint angle of joint 2.

	Calculated Position Values (m)			Actual Position Values (m)		
Joint Angles (˚)	x	y	z	x	y	z
5	0.27	0.00	-0.024	0.27 ± 0.01	0.00 ± 0.01	-0.0050 ± 0.02
45	0.19	0.00	-0.19	0.22 ± 0.01	0.00 ± 0.01	-0.17 ± 0.02
60	0.14	0.00	-0.24	0.15 ± 0.01	0.00 ± 0.01	-0.21 ± 0.01
90	0.00	0.00	-0.27	0.005 ± 0.02	0.00 ± 0.01	-0.27 ± 0.01
120	-0.14	0.00	-0.24	-0.14 ± 0.01	0.00 ± 0.01	-0.21 ± 0.01

Interpretation of Joint 2 results

At first glance, the calculated values do match the actual position values obtained. There are also smaller ranges in the data collected compared to joint 1 which can be because joint 2 is performing better than joint 1. Again, the team face a small range in the data collected for the y-coordinates due to estimation errors. At joint angles other than 90 degrees, the images of the robot depict that the weight of the arm can reduce the z-coordinate of the gripper. Therefore, often the calculated z-coordinate is an overestimate because the calculations do not take into account the weight of the robot's arm. Which is consistent in the data collected – the actual z-coordinate which is measured is consistently smaller than the calculated value.

Images of Joint 2

Images to document the results and for further analysis to supplement the data that was collected.

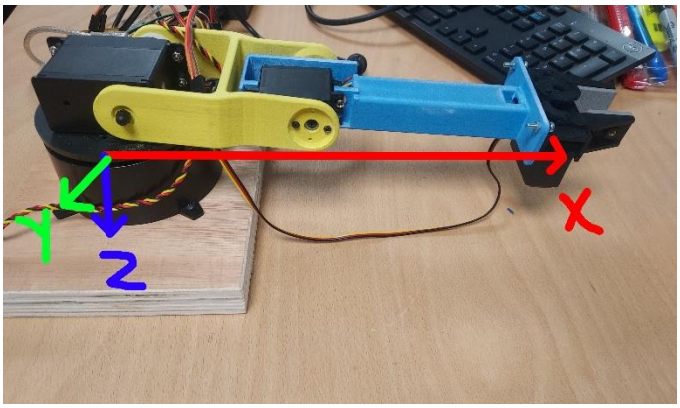


Figure 4.1: Joint 2 set to 5 degrees angle. Joint 1 and 3 are set to 0 degrees. The axes are annotated and remain the same for the rest of the images of Joint 2.

Figure 4.2: Joint 2 set to 45 degrees angle. Joint 1 and 3 are set to 0 degrees.

Figure 4.3: Joint 2 set to 60 degrees angle. Joint 1 and 3 are set to 0 degrees.

Figure 4.4: Joint 2 set to 90 degrees angle. Joint 1 and 3 are set to 0 degrees.

Figure 4.5: Joint 2 set to 120 degrees angle. Joint 1 and 3 are set to 0 degrees.

These figures show the various angular testing positions of joint 2. These pictures confirmed that joint 2 is behaving as predicted.

Results for Joint 3

Table 3: The measured position values of the gripper with respect to the base frame compared to the calculated position values from the MATLAB Code at each joint angle of joint 3.

	Calculated Position Values (m)			Actual Position Values (m)		
Joint Angles (˚)	x	y	z	x	y	z
5	0.24	0.00	0.014	0.24 ± 0.01	0.00 ± 0.01	0.04 ± 0.01
20	0.26	0.00	0.060	0.24 ± 0.02	0.00 ± 0.01	0.08 ± 0.01
45	0.22	0.00	0.13	0.22 ± 0.01	0.00 ± 0.01	0.14 ± 0.01
60	0.18	0.00	0.16	0.20 ± 0.01	0.00 ± 0.01	0.15 ± 0.01
80	0.13	0.00	0.18	0.12 ±0.01	0.00 ± 0.01	0.17 ± 0.01

Interpretation of Joint 3 Results

Finally, for the last joint the results conclude that again, the calculations are correct. However, slight variations in the accuracy occur as indicated by a greater difference in the measured value and the calculated value. Estimation errors are inevitable, as discussed with the joint 1 and joint 2 results, especially with the measuring equipment used (ruler). For example, the pictures of the robot at each position, illustrate the team could have faced trouble measuring

the x, y, z positions due to the piece of wood between the gripper and base frame which can impact the accuracy of the measured values.

Images of Joint 3

Images to document the results and for further analysis to supplement the data that was collected.

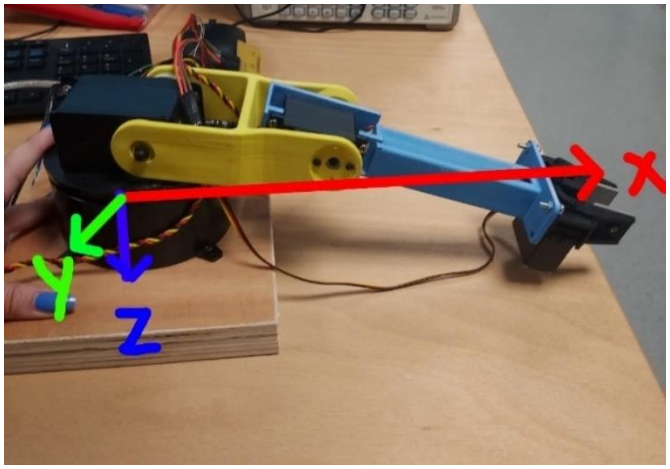


Figure 5.1: Joint 3 set to 5 degrees angle. Joint 1 and 2 are set to 0 degrees. The axes are annotated and remain the same for the rest of the images of Joint 3.

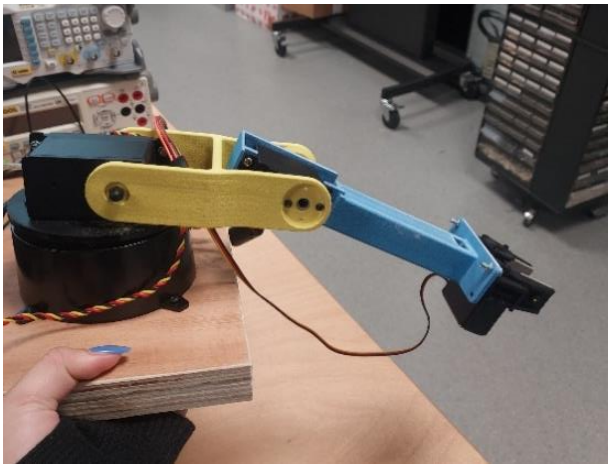


Figure 5.2: Joint 3 set to 20 degrees angle. Joint 1 and 2 are set to 0 degrees.

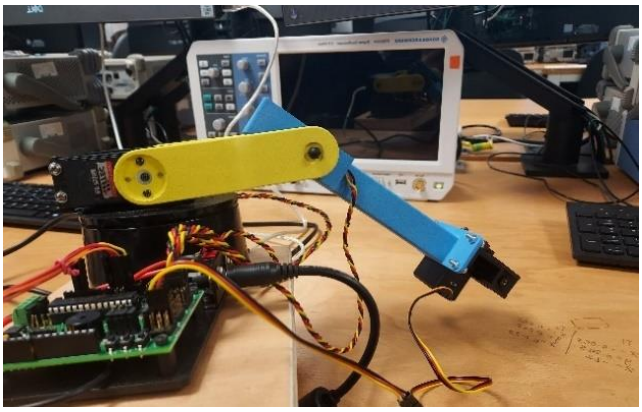


Figure 5.3: Joint 3 set to 45 degrees angle. Joint 1 and 2 are set to 0 degrees.

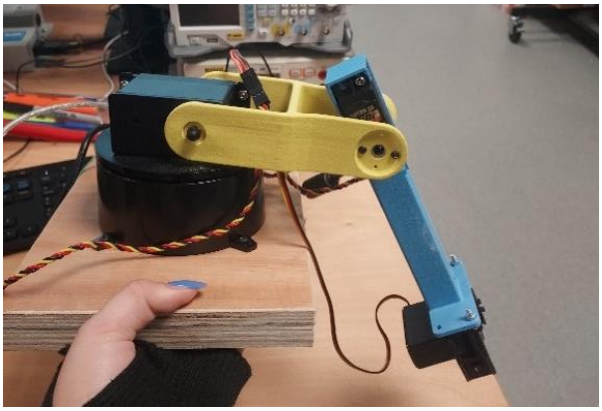


Figure 5.4: Joint 3 set to 60 degrees angle. Joint 1 and 2 are set to 0 degrees.

Figure 5.5: Joint 3 set to 80 degrees angle. Joint 1 and 2 are set to 0 degrees.

These figures show the various angular testing positions of joint 3. These pictures confirmed that joint 3 is behaving as predicted.

Conclusion

Overall, Forward Kinematics was a success. To reflect on the investigation, the team had discovered what can impact their robot's movement. For example, the weight of the robot's arms, or the joint's motors. Fortunately, before moving forward, there was a replacement of the joint's motor unit, as the base (joint 1) had ceased working. Furthermore, to improve the calculations the team can consider the vertical component of the weight of the robot arm when finding the Z co-ordinate.

III – Inverse Kinematics

1. Calculations

The Inverse Kinematics is calculated using a geometric approach. There are two configurations of the robot known as the "Elbow-up" case and the "Elbow-down" case. The following calculations, start with the "Elbow-up" case and then illustrate how the "Elbow-down" case is obtained.

1.1: Finding θ_1 using trigonometry

Firstly, an assumption is made that the gripper is positioned at some coordinate (X, Y, Z), then work backwards to find the angle of each of the joints.

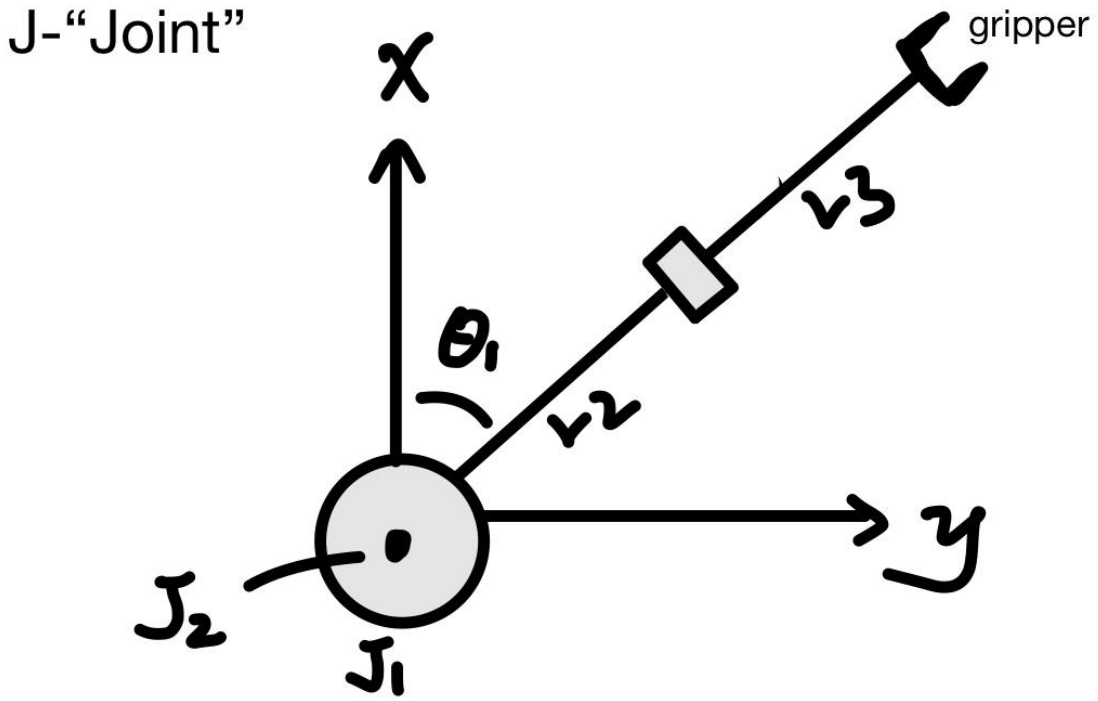


Figure 6.1: a bird’s eye view of the robot

Figure 6.1 to illustrates θ_1. Using trigonometry, the value of θ_1 is obtained by the following formula, given some (X, Y, Z) coordinate:

$$\theta_1 = tan^{-1}(\frac{Y}{X})$$

1.2: Finding θ_2 using the Cosine Rule

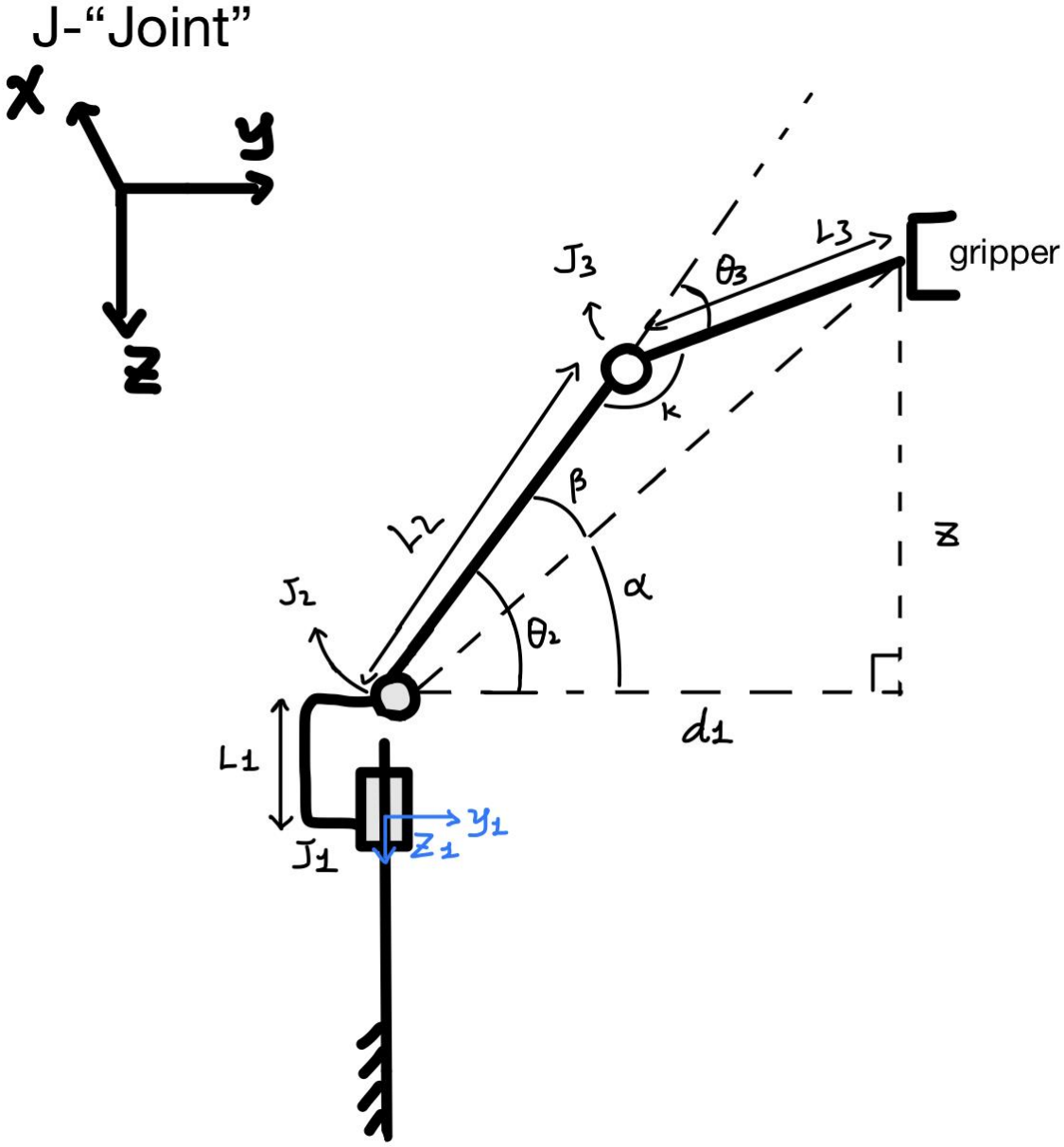


Figure 6.2: A side view of the robot. Angles α and β are intermediate angles that are calculated to obtain the values of the joint 2 and joint 3 angles.

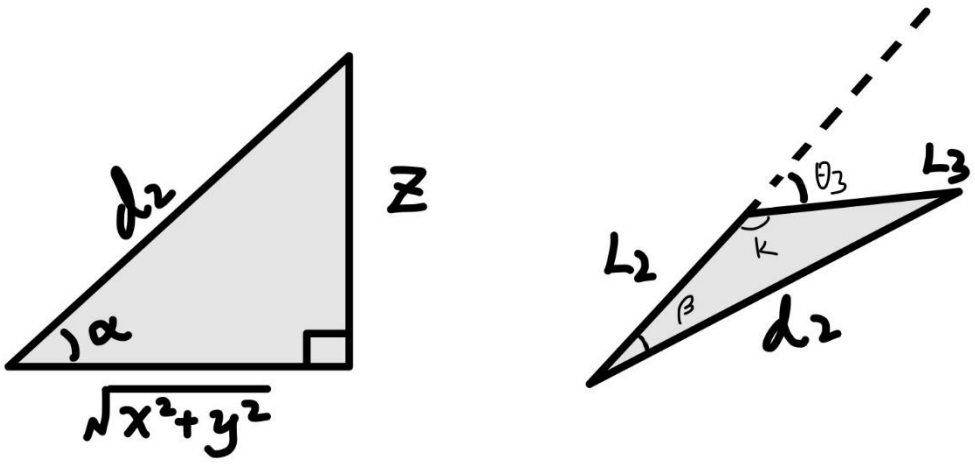
To calculate angles θ_2 and θ_3, Figure 3.2 above is utilised. From the diagram drawn two triangles can be obtained that will be especially helpful for calculating the joint angles.

To allow for the calculations to be more precise, the small length between Joint 1 and Joint 2 should be considered. Therefore, the length Z in the diagram can be written as,

$Z = Z_f - L1$, where Z_f represents gripper final position Z coordinate value.

Firstly, d_1 is calculated using the Pythagoras' theorem and added to the diagrams:

$$d_1 = \sqrt{X^2 + Y^2}$$



Figures 6.3 and 6.4: α is calculated using trigonometric ratios from the right-angled and β can be calculated using the Cosine Rule.

Next, the following equations are derived to calculate the Joint 2 and Joint 3 angles using figures 6.3 and 6.4 (triangles):

$$\theta_2 = \alpha + \beta, \qquad \theta_3 = 180^\circ - k$$

Then use trigonometry to find α°:

$$\alpha^\circ = tan^{-1}\left(\frac{-Z}{\sqrt{X^2 + Y^2}}\right)$$

After, apply the Pythagoras' Theorem again to find d_2:

$$d_2 = \sqrt{\left(\sqrt{X^2 + Y^2}\right)^2 + Z^2}$$

$$= \sqrt{X^2 + Y^2 + Z^2}$$

Following that, find the angle β° using the Cosine Rule:

$$c^2 = a^2 + b^2 - 2bc\ cos(C)$$

Where, $\mathbf{a = L_2}$, $\mathbf{b = d_2 = \sqrt{X^2 + Y^2 + Z'^2}}$, $\mathbf{c = L_3}$ **and,** $\mathbf{\angle C = \beta^\circ}$

$$L_3^2 = L_2^2 + X^2 + Y^2 + Z^2 - 2 * L_2 * \sqrt{X^2 + Y^2 + Z^2}\ cos(\beta^\circ)$$

Rearrange for $cos(\beta^\circ)$:

$$cos(\beta^{\circ}) = \frac{L_2^2 + X^2 + Y^2 + Z^2 - L_3^2}{2 * L_2 * \sqrt{X^2 + Y^2 + Z^2}}$$

And find $sin(\beta^{\circ})$:

$$sin(\beta^{\circ}) = \sqrt{1 - cos^2(\beta)}$$

Finally, β° is obtained as such:

$$\beta^{\circ} = tan^{-1}\left(\frac{sin(\beta^{\circ})}{cos(\beta^{\circ})}\right)$$

Going back to the equation for θ_2 , substitute the values of α° and β°:

$$\theta_2 = \alpha + \beta$$

$$\theta_2 = tan^{-1}\left(\frac{-Z}{\sqrt{X^2 + Y^2}}\right) + tan^{-1}\left(\frac{sin(\beta)}{cos(\beta)}\right)$$

$$\theta_2 = tan^{-1}\left(\frac{-Z}{\sqrt{X^2 + Y^2}}\right) + tan^{-1}\left(\frac{\sqrt{1 - \left(\frac{L_2^2 + X^2 + Y^2 + Z^2 - L_3^2}{2 * L_2 * \sqrt{X^2 + Y^2 + Z^2}}\right)^2}}{\left(\frac{L_2^2 + X^2 + Y^2 + Z^2 - L_3^2}{2 * L_2 * \sqrt{X^2 + Y^2 + Z^2}}\right)}\right)$$

1.3: Finding θ_3 using the Cosine Rule

Similarly, to find the angle k, apply the cosine rule again,

a = d_2 = $\sqrt{X^2 + Y^2 + Z^2}$, b = L_2, c = L_3 and, ∠C = k$^{\circ}$

$$X^2 + Y^2 + Z^2 = L_2^2 + L_3^2 - 2 * L_2 * L_3\, cos(k)$$

$$cos(k) = \frac{L_2^2 + L_3^2 - X^2 - Y^2 - Z^2}{2 * L_2 * L_3}$$

$$sin(k) = \sqrt{1 - cos^2(k)}$$

$$k = tan^{-1}\left(\frac{sin(k)}{cos(k)}\right)$$

Since,

$$\theta_3 = 180^{\circ} - k$$

This implies,

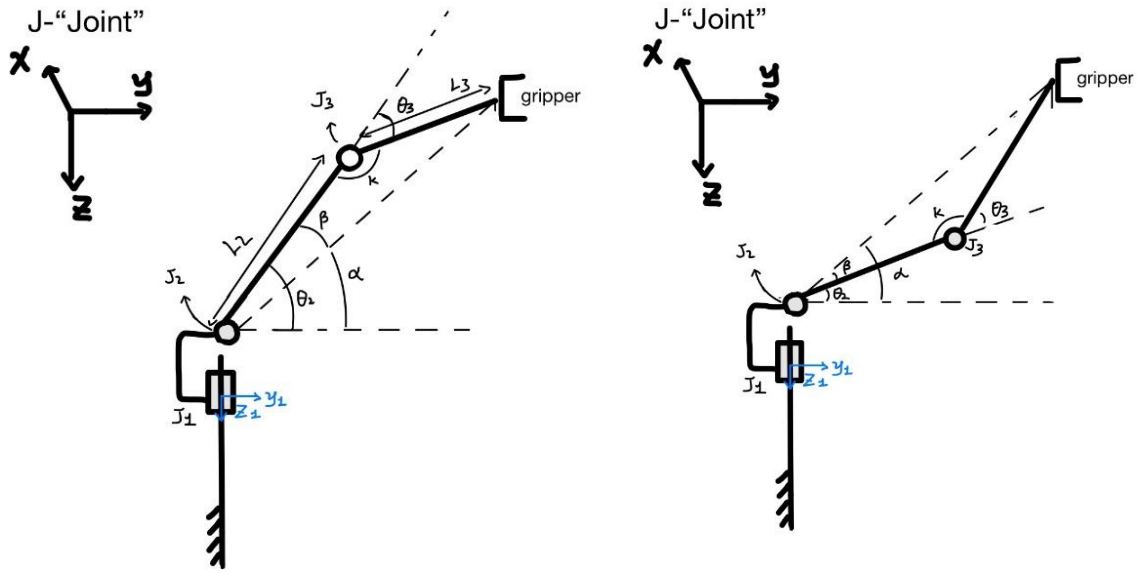
$$\theta_3 = 180^{\circ} - tan^{-1}\left(\frac{sin(k)}{cos(k)}\right)$$

$$\theta_3 = 180^\circ - tan^{-1}\left(\frac{\sqrt{1-\left(\frac{L_2^2+L_3^2-X^2-Y^2-Z^2}{2*L_2*L_3}\right)^2}}{\left(\frac{L_2^2+L_3^2-X^2-Y^2-Z^2}{2*L_2*L_3}\right)}\right)$$

To summarise, the angle of each joint is (Elbow-up case):

$$\begin{cases} \theta_1 = tan^{-1}(\frac{Y}{X}) \\ \theta_2 = tan^{-1}\left(\frac{-Z}{\sqrt{X^2+Y^2}}\right) + tan^{-1}\left(\frac{\sqrt{1-\left(\frac{L_2^2+X^2+Y^2+Z^2-L_3^2}{2*L_2*\sqrt{X^2+Y^2+Z^2}}\right)^2}}{\left(\frac{L_2^2+X^2+Y^2+Z^2-L_3^2}{2*L_2*\sqrt{X^2+Y^2+Z^2}}\right)}\right) \\ \theta_3 = 180^\circ - tan^{-1}\left(\frac{\sqrt{1-\left(\frac{L_2^2+L_3^2-X^2-Y^2-Z^2}{2*L_2*L_3}\right)^2}}{\left(\frac{L_2^2+L_3^2-X^2-Y^2-Z^2}{2*L_2*L_3}\right)}\right) \end{cases}$$

1.4: From Elbow-up case to Elbow-down case



Figures 6.5 and 6.6 Two possible paths for same final position, Elbow up (Left) and Elbow down (Right).

As shown above in figures 6.5 and 6.6, the elbow-up case has already been calculated in the previous step. The key parameters (α, β and k) stay the same for both the elbow-up and elbow-down cases.

Now to consider the elbow-down case, the diagrams are used to help find multiple solutions for the joint angles. Starting with the angle of Joint 1, since there is no rotation on x-y plane, going from figures 6.5 to 6.6, it will stay the same. Next, the angle of Joint 2 in the

elbow-down case will become $\alpha - \beta$ as illustrated by the diagrams, instead of $\alpha + \beta$. Finally, for the angle of Joint 3, the angle of $\pi - k$ stays the same.

So, the angle of each Joint would be as such -

$$\begin{cases} \theta_1 = tan^{-1}(\frac{Y}{X}) \\ \theta_2 = \alpha - \beta \\ \theta_3 = 180^{\circ} - k \end{cases}$$

.

The parameters $(\alpha, \beta \ and \ k)$ are still the same as in the Elbow-up case, so by substituting the parameters in the expression as shown in section 1.3 they obtain the expression for the Elbow-down case.

$$\begin{cases} \theta_1 = tan^{-1}(\frac{Y}{X}) \\ \theta_2 = tan^{-1}\left(\frac{-Z}{\sqrt{X^2+Y^2}}\right) - tan^{-1}\left(\frac{\sqrt{1-\left(\frac{L_2^2+X^2+Y^2+Z^2-L_3^2}{2*L_2*\sqrt{X^2+Y^2+Z^2}}\right)^2}}{\left(\frac{L_2^2+X^2+Y^2+Z^2-L_3^2}{2*L_2*\sqrt{X^2+Y^2+Z^2}}\right)}\right) \\ \theta_3 = 180^{\circ} - tan^{-1}\left(\frac{\sqrt{1-\left(\frac{L_2^2+L_3^2-X^2-Y^2-Z^2}{2*L_2*L_3}\right)^2}}{\left(\frac{L_2^2+L_3^2-X^2-Y^2-Z^2}{2*L_2*L_3}\right)}\right) \end{cases}$$

However, for the real robotic coding, only the elbow-up calculations are utilised, therefore, the tests that follow are only based on the elbow-up case.

2. Testing the inverse kinematics calculations

Next, the accuracy of the results is tested. The independent variable, which they changed, was the position of the end-effector (x, y, z coordinates) and the dependent variable that were measured was the actual joint angles of the robot. The measurements were repeated 3 times and an average was taken. A range is calculated from the three values measured for each position to further investigate the accuracy of the results. All values are given to 3 significant figures which was suitable given the precision of the protractor, ± 0.5 degrees.

Table 4: comparing the calculated joint angles that the MATLAB code (see Appendix B) calculated, and the actual average joint angles measured of the robot at 5 different positions of the gripper.

			Calculated Joint Angles (˚)			Actual Joint Angles (˚)		
x	y	z	θ_1	θ_2	θ_3	θ_1	θ_2	θ_3
0.10	0.10	-0.10	45	114	109	45 ± 2	115 ± 1	110 ± 2
0.15	0.00	-0.10	0	108	105	0 ± 3	110 ± 1	105 ± 1
-0.10	0.10	0.00	135	98	129	138 ± 2	100 ± 2	132 ± 1
0.20	0.10	-0.05	27	60	71	30 ± 1	60 ± 1	70 ± 1
-0.10	0.25	-0.01	-45	105	82	30 ± 1	110 ± 1	85 ± 2
-0.08	0.05	-0.10	148	147	131	145 ± 1	143 ± 1	49 ± 1

3. Interpretation of results:

Looking at the data collected, the difference between the calculated joint angles and the measured joint angles is very little and majority of the time the calculations are accurate. Furthermore, there were no anomalies found in the data collected. However, for some values, for example the angle for Joint 1 at position (-0.1, 0.1, 0), there is a more significant difference in the calculated angles and the measured angles compared to others. It is suspected the reason for this can be random errors.

These random errors may include:

- Human errors when taking measurements using a protractor. It can be difficult reading the angles when the robot is in position that is uncomfortable to reach with a protractor. This is illustrated by larger ranges in some average values presented in the table.
- The voltage from the power supply can impact how much energy the robot may use to move. The more powerful the power supply the easier it is for the robot to move to the desired position with more accuracy.
- Performance of the joints could have been reduced due to aging or having been worn out.

4. Images of the Robot at each position:

Photographs of the robot are taken at each stage of data collection to gain a better understanding of the results and extract more information about the behaviour of the robot's joints.

Figure 7.1: photograph of the end-effector positioned at (0.1, 0.1, -0.1)

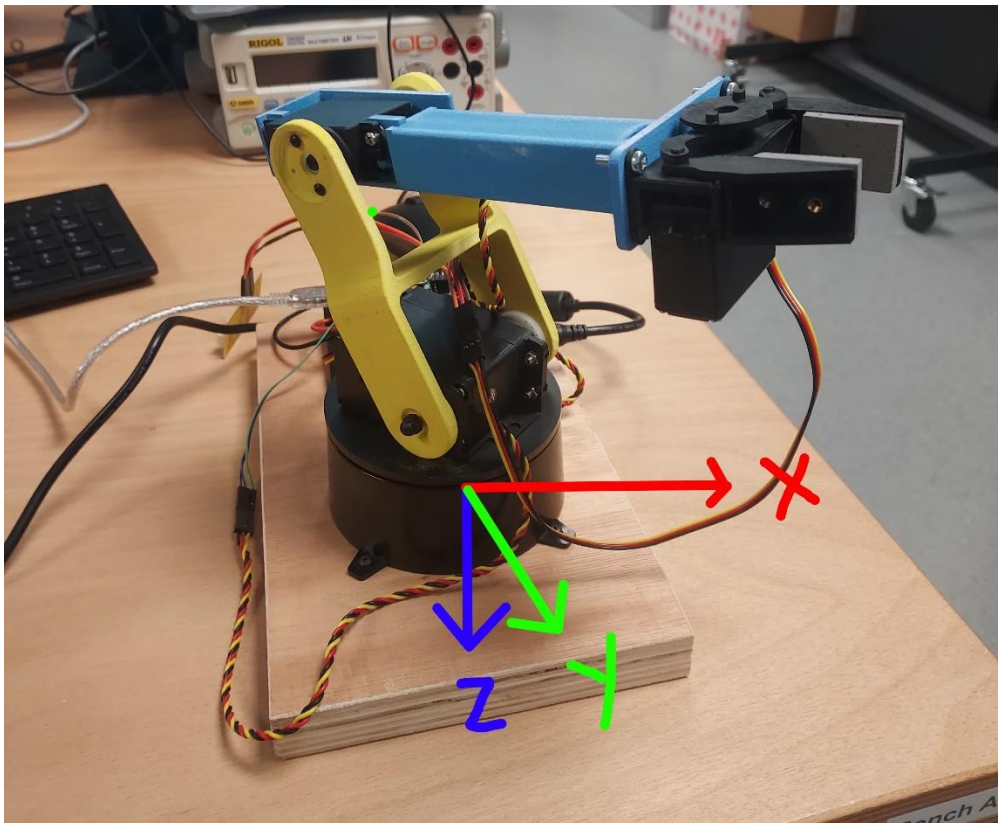


Figure 7.2: photograph of the end-effector positioned at (0.15, 0, -0.1)

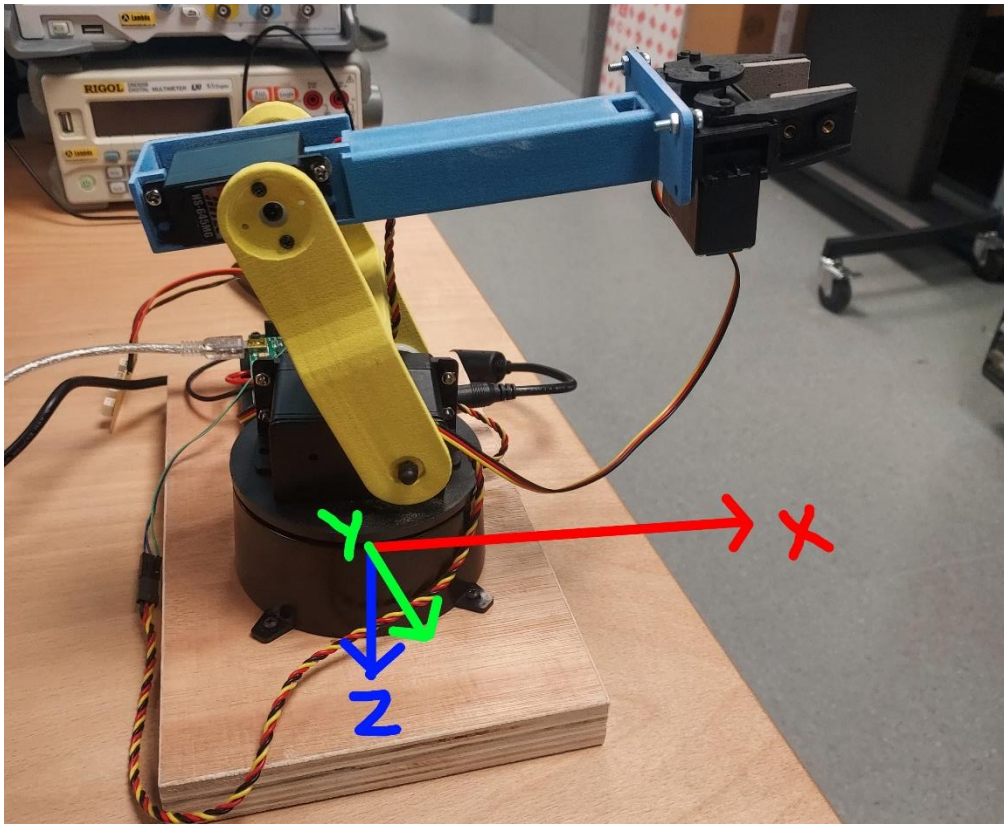


Figure 7.3: photograph of the end-effector positioned at (-0.1, 0.1, 0)

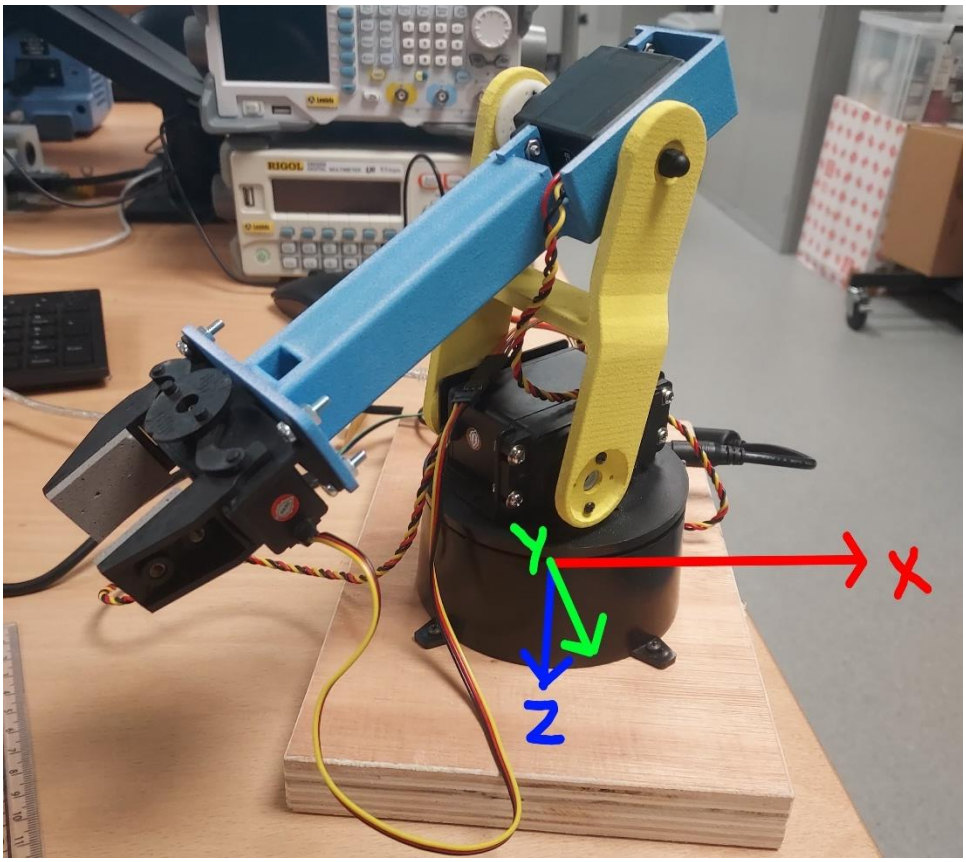


Figure 7.4: photograph of the end-effector positioned at (0.2, 0.1, -0.05)

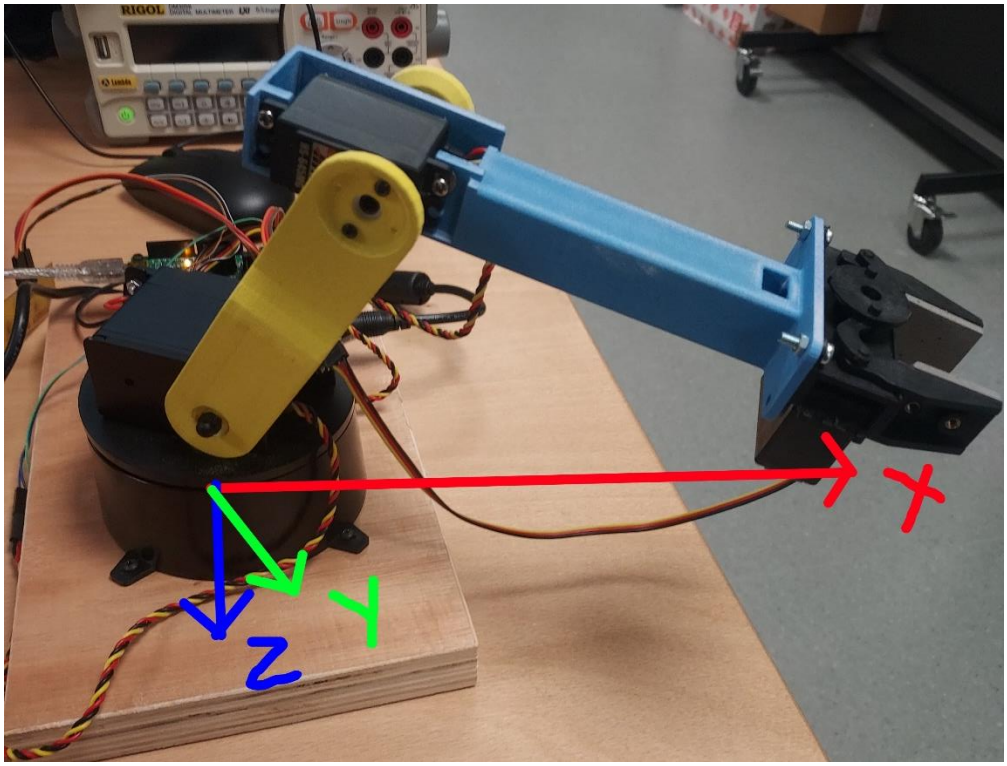
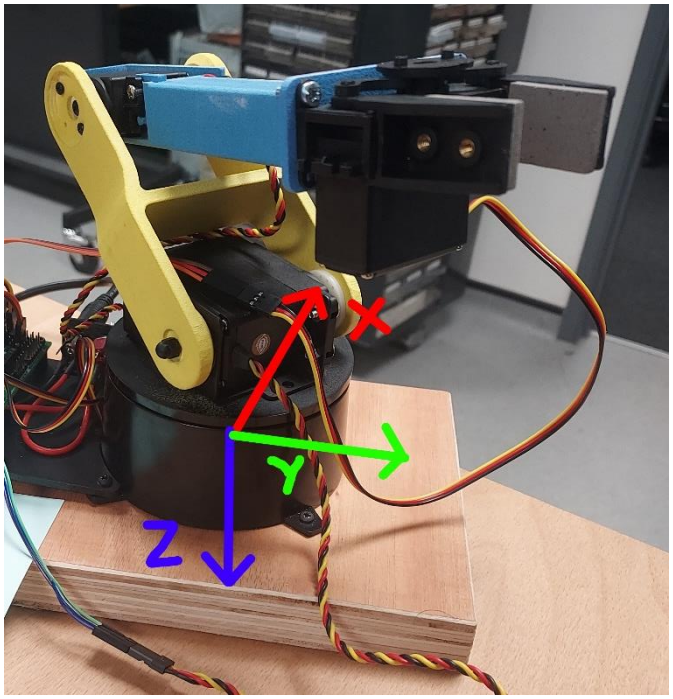


Figure 7.5: photograph of the end-effector positioned at (-0.1, 0.25, -0.1)



5. Coding the robot to move with inverse kinematics

The Arduino code is used to program the robot to move in a linear motion parallel to each of the axes. A potentiometer was used to vary the values of either the x, y or z coordinates whilst keeping the other 2 values constant to achieve this linear motion. The results are shown in the video links below.

Robot moving parallel to the X-axis:

File Name – Videos/InverseKinematics_X_Axis.mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/InverseKinematics_X_Axis.mp4?csf=1&web=1&e=n0SQrZ

Robot moving parallel to the Y-axis:

File Name – Videos/InverseKinematics_Y_Axis.mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/InverseKinematics_Y_Axis.mp4?csf=1&web=1&e=KaomIp

Robot moving parallel to the Z-axis:

File Name – Videos/InverseKinematics_Z_Axis.mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/InverseKinematics_Z_Axis.mp4?csf=1&web=1&e=n0SQrZ

6. Interpretation of results:

Linear motion of the robot's end-effector using inverse kinematics has been successfully derived. The next step in progressing towards the 'pick-and-place' demonstration, would be trajectory planning. Having done the forward and inverse kinematics thoroughly, it should not be too difficult to derive the trajectory plans for the robot. Currently, the Arduino code produces rough and jagged movement of the robot as shown in the videos. Therefore, for trajectory planning more parameters are introduced, with the use of a cubic polynomial to model the path of the robot, to achieve a smoother movement of the robot from a position A to another position, B.

IV – Trajectory Planning

The task required the robot arm to move in a straight line, which can be represented using cartesian space schemes as it enforces a geometrical path.

1. Workflow

To achieve this motion, cubic polynomials were used where the variable in the polynomials represented the distance of the gripper with respect to the base frame. Although, a straight time profile is a simple and easy method for trajectory planning, a disadvantage of this will be that the function of the velocity will not be continuous. This means, the motion of the robot will be rough and jerky. A cubic polynomial allows the use of more parameters meaning more constraints can be set for the motion of the robot to reduce jerky motion.

By using the information of the starting position, ending position, and the time to reach the final position, an equation was created to describe the motion of the robot's end-effector in the x, y, and z axes over time. The Euclidean distance of the end-effector relative to the base frame is expressed using the variable 'u'.

u(t): The function of displacement over time
u_0: The starting position (of the end effector)
u_f: The end position
t_f: The time to reach the final position

The function of a cubic polynomial is:

$$\mathrm{u(t)} = \begin{cases} a_0 + a_1 t + a_2 t^2 + a_3 t^3 & t < t_f \\ u_f & t \geq t_f \end{cases}$$

To find out the parameters $a_0, a_1, a_{2,} a_3$, the following constraints are applied,

u(0) = u_0 when time = 0 s, it is in its start position
u(t_f) = u_f when time equal to t_f, it is in its final position
u'(0) = 0 when time = 0 s, velocity is 0m/s

u’(t_f) = 0 when time = 0 s, velocity is 0m/s,

to the cubic polynomial to obtain the following simultaneous equations:

u(0) = $a_0 + a_1 0 + a_2 0^2 + a_3 0^3 = a_0 = u_0$
u(t_f) = $a_0 + a_1 t_f + a_2 {t_f}^2 + a_3 {t_f}^3 = u_f$

Differentiating u(t) gives, $\dot{u}$(t) = $\frac{d}{dt}(a_0 + a_1 t + a_2 t^2 + a_3 t^3) = a_1 + 2a_2 t + 3a_3 t^2$
$\dot{u}(0) = a_1 + 2a_2 0^2 + 3a_3 0^2 = a_1 = 0$
$\dot{u}(t_f) = a_1 + 2a_2 t_f + 3a_3 {t_f}^2 = 0$

By solving the simultaneous equations above, the parameters in terms of the time taken, the start and end positions, of the cubic polynomial are found:

$a_0 = u_0$
$a_1 = 0$
$a_2 = \frac{3}{{t_f}^2}(u_f - u_0)$
$a_3 = -\frac{2}{{t_f}^3}(u_f - u_0)$

$$\text{u(t)} = \begin{cases} u_0 + \frac{3}{{t_f}^2}(u_f - u_0)t^2 - \frac{2}{{t_f}^3}(u_f - u_0)t^3 & t < t_f \\ u_f & t \geq t_f \end{cases}$$

With expression above, the motion in x-axis can be expressed as:

$$\text{X(t)} = \begin{cases} X_0 + \frac{3}{{t_f}^2}(X_f - X_0)t^2 - \frac{2}{{t_f}^3}(X_f - X_0)t^3 & t < t_f \\ X_f & t \geq t_f \end{cases}$$

For motion in the y-axis:

$$\text{Y(t)} = \begin{cases} Y_0 + \frac{3}{{t_f}^2}(Y_f - Y_0)t^2 - \frac{2}{{t_f}^3}(Y_f - Y_0)t^3 & t < t_f \\ Y_f & t \geq t_f \end{cases}$$

For motion in the z-axis:

$$\text{Z(t)} = \begin{cases} Z_0 + \frac{3}{{t_f}^2}(Z_f - Z_0)t^2 - \frac{2}{{t_f}^3}(Z_f - Z_0)t^3 & t < t_f \\ Z_f & t \geq t_f \end{cases}$$

These equations of motion in X, Y and Z axis were used for simulation of the robot’s motion in MATLAB. (See Appendix C)

With the simulation in the MATLAB:

The following data was used for input,
u_0: [x = 0.00, y = 0.18, z = -0.094]
u_f: [x = 0.10, y = 0.15, z = -0.20].

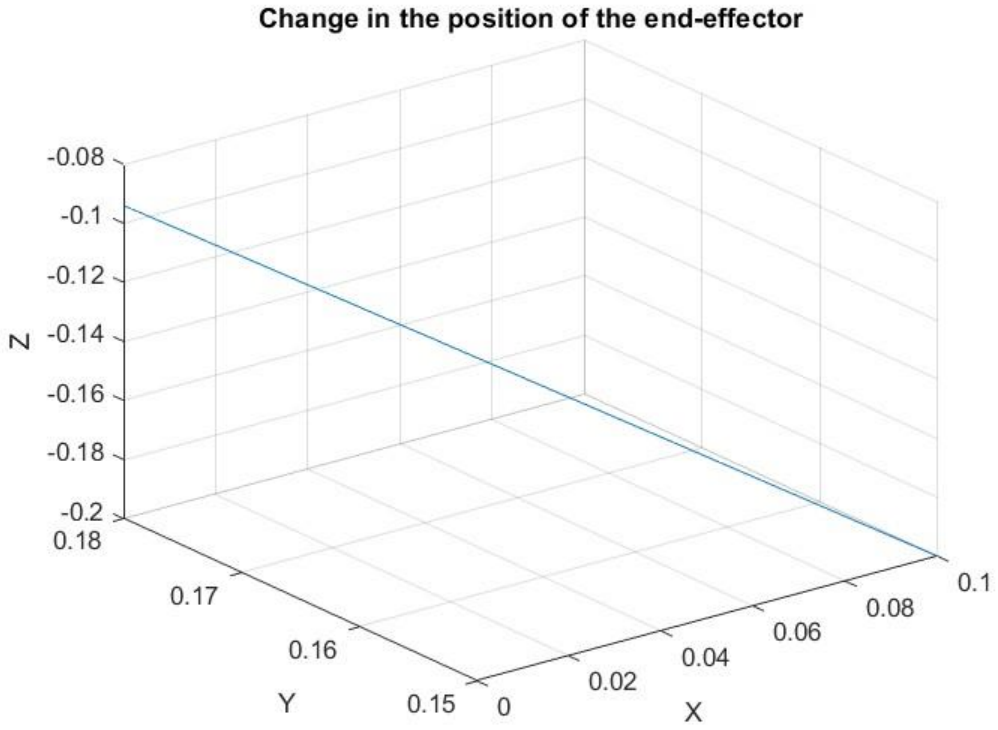


Figure 8.1: A 3D graph showing the change in position of the gripper with respect to the base frame.

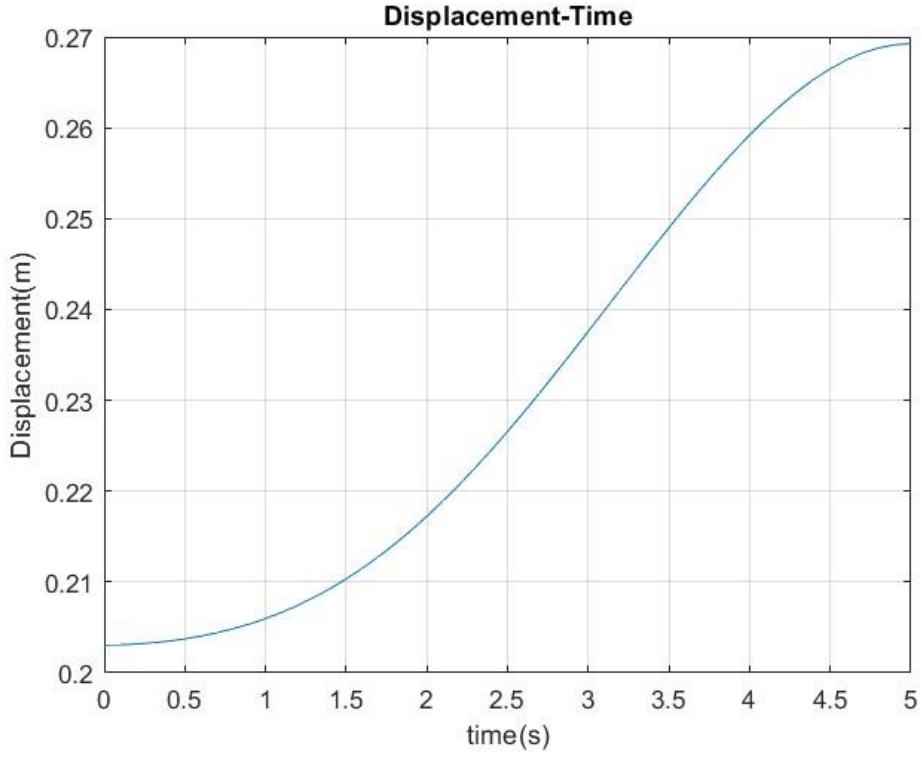


Figure 8.2: Displacement-time graph showing how displacement of the end-effector varies with time.

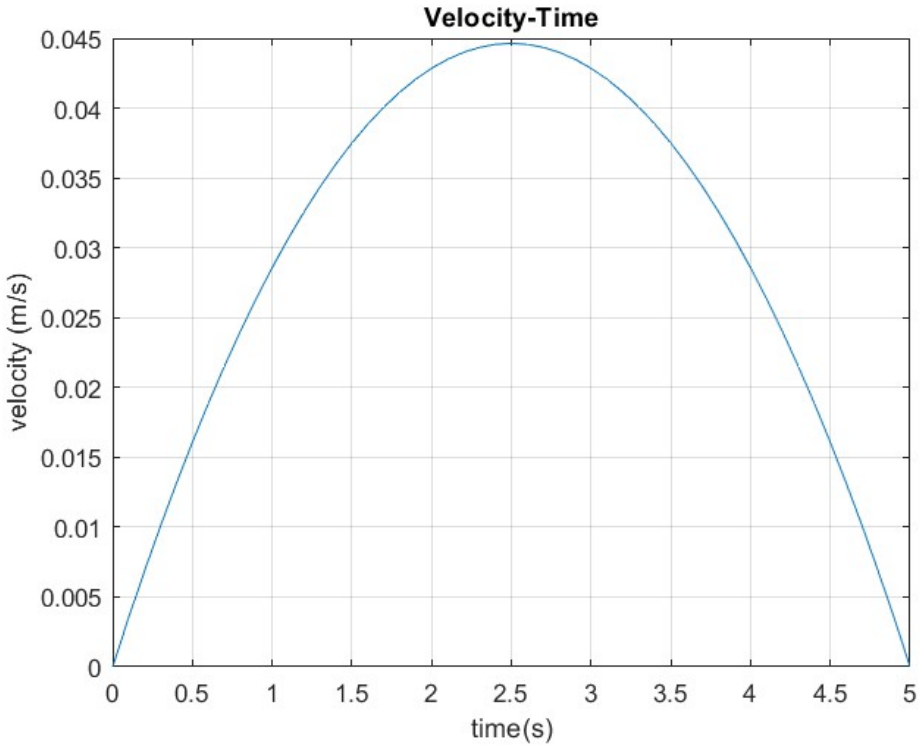


Figure 8.3: Velocity-time graph showing how the velocity of the end-effector (gripper) varies with time.

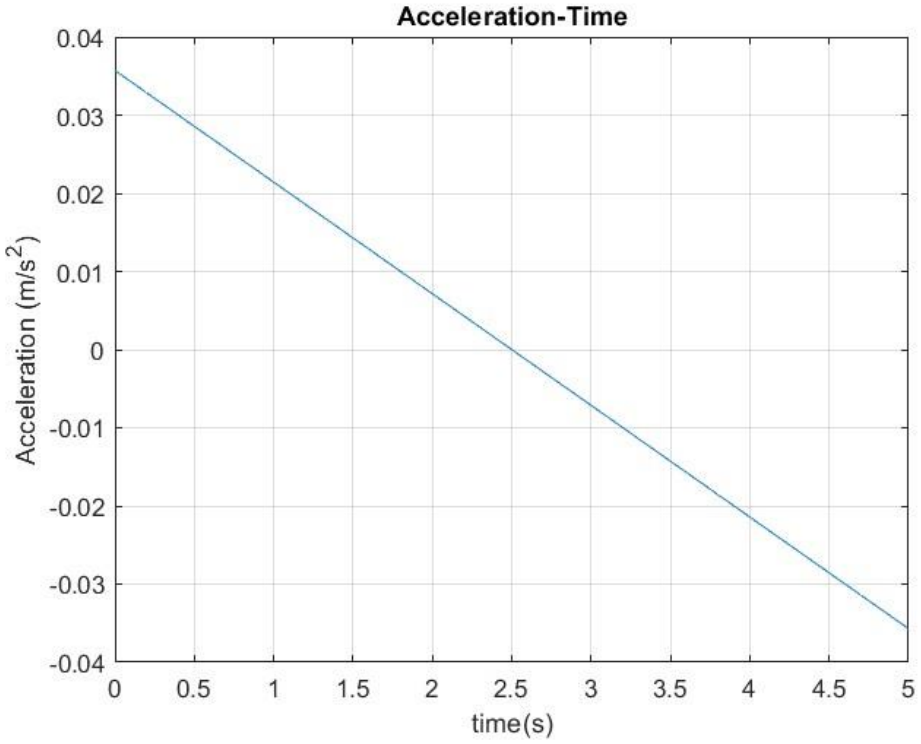


Figure 8.4: Acceleration-time graph showing how acceleration of the end-effector (gripper) varies with time.

These figures show the MATLAB graphs for the trajectory of the robot, plotted as a function of time.

2. Analysis of Time Graphs

The time graphs drawn from the MATLAB Code simulate the motion of the robot. The MATLAB code plots the output of each of the corresponding functions given the input of time t, in the domain $0 \leq t \leq 5$, at regular intervals of 0.1 s.

In Figure 8.1, the graph shows how the x, y and z coordinates vary. A linear line is observed which implies that this trajectory is a linear path from the point (0.05, 0, 0.1) to (0.04, 0.03, 0.01).

In Figure 8.2, the displacement-time graphs implies that there is slow increase of the velocity of the end-effector at the start of motion. The velocity increases until t = 2.5 s (middle of the trajectory) and then the velocity starts to decrease again. Finally, in the second half of the trajectory the robot will slowly move to its end position. Figure 8.3 provides more information on how the velocity of the end-effector changes with time and is obtained by finding the first derivative of the displacement-time graph function. The observation that the velocity increases in the displacement-time graph in the first half is supported by the velocity-time graph. The velocity increases from 0 m/s and then reaches a maximum velocity of 0.045 m/s at t = 2.5 s. From here on the velocity decreases at a slow pace until the end-effector reaches a velocity of 0 m/s, as expected by the calculations. Therefore, there is positive acceleration in the first half of the trajectory and later there is equal but negative acceleration in the second half of the trajectory. This behaviour is demonstrated in the acceleration-time graph shown in Figure 8.4, which is obtained by finding the second derivative of the displacement-time function.

The gradient of an acceleration-time graph is known as the jerk – the rate of change in acceleration. A straight line of negative gradient, -0.0143 (3 s.f.), in Figure 8.4 suggests the change in acceleration is negative but constant. Since the acceleration is not constant, the robot will exhibit some jerky movements. For a smooth, non-jerky movement of the robot, a constant acceleration is required, however, this is not possible to achieve with only just the cubic polynomial because more parameters are required. This is when the quintic polynomial is useful.

These graphs indicate that the theoretical motion of the robot's end-effector is correct given the constraints considered for the calculations. This implies the calculations are correct.

3. Coding Trajectory Planning on Arduino

The formula must now be uploaded to Arduino to cut down the motion of the robot arm into steps with an interval of 0.1 second. This will generate the X, Y, and Z cartesian co-ordinates for each 0.1 second. By using the inverse kinematic method (refer to page 10) to apply these coordinates, the joint angle at each 0.1 second can be calculated.

The initial approach: To ensure the robot motion completes within the given time step, it is necessary to store the calculated results before the motion starts, as the calculation time may slow down the movement. To achieve this, a time list was created with an interval of 0.1s,

ranging from 0s to 5s. By plugging the time list into the cubic polynomial expression, a corresponding list of coordinates was generated. Inverse kinematics were applied to calculate the motor angles, which were also stored in a list. In the loop function, the angle list is processed from head to tail with a delay of 0.1s. After waiting for 10s, the list is processed from tail to head to allow the robot to return to its original position. This Arduino code is located under the relative file path, ‘ Arduino_files\Trajectory_Task1Arduino.ino’.

Problem with this approach:

Only a single trajectory motion is applied, which might dislodge the object that needs to be picked up while the robot arm is moving.

Solution:

By splitting the motion into two trajectory motions, the robot arm would first reach the top of the object, and the second motion would move it down to the final position.

Problem with the solution:

The arrays are occupying too much memory on the board, which causes stability problems, or the Arduino code will not compile.

Solution:

Instead of storing all the calculations in arrays, only the trajectory coordinates are stored. The remaining inverse kinematic calculations will be performed during the robot's movement.

This updated Arduino code can be found in the relative file path: ‘Arduino_files\Trajectory_Task1Arduino1.ino’.

Here are videos showing the arm reaching three different positions selected by the team:

Trajectory planning from position (0, 0.18, -0.094) to (0.1, 0.15, -0.2):

File Name – Videos/Trajectory(0.1, 0.15, -0.2).mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/Trajectory(0.1,%200.15,%20-0.2).mp4?csf=1&web=1&e=5rnl0B

Trajectory planning from position (0, 0.18, -0.094) to (-0.2, 0.1, -0.1):

File Name – Videos/Trajectory(-0.2, 0.1, 0.1).mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/Trajectory(-0.2,%200.1,%20-0.1).mp4?csf=1&web=1&e=Z7DqV7

Trajectory planning from position (0, 0.18, -0.094) to (0.2, 0.1, -0.1):

File Name – Videos/Trajectory(0.2, 0.1, 0.1).mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/Trajectory(0.2,%200.1,%20-0.1).mp4?csf=1&web=1&e=djwrBT

In preparation for the Pick and Place Demo, different positions of the object within the workspace of the robot were selected to practice using the Arduino code for trajectory planning. Here’s a video demoing one of the positions:

File Name – Videos/PickNPlaceDemo.mp4
https://liveuclac.sharepoint.com/:v:/r/sites/ELEC0129276/Shared%20Documents/General/Videos/PickNPlaceDemo.mp4?csf=1&web=1&e=mzbbNI

Interpretation of Results

As it can be seen from these videos, the robot is very capable at picking up objects within its workspace. The robot takes in inputs of where the object is in relation to the base reference frame and performs the pick-up in two linear motions. The first motion is the set up for the grabbing motion, where the robot is the correct X and Y cartesian co-ordinates, but with a difference value for the Z co-ordinate. This is to ensure a smooth grabbing motion as the robot arm will gently drop down onto the object, only having to change one parameter, which would mean increased accuracy and precision.

Overall, using the cubic polynomial to plan the trajectory reduced the abrupt movement of the robot. This is desirable because jagged and instantaneous movement of the robot can cause wear and tear of the joints, reducing its life span or in other words, the robot's economic life. Reducing the delay in the Arduino code, in between writing the angles to each joint, can also help to achieve a more seamless trajectory in between any two positions.

V – Conclusion

This report documents the progress – from the action of building the robot to being capable of performing complex pick and place operations. This involved learning and deriving formulas for forward kinematics, inverse kinematic and trajectory planning. The most difficult aspect of this report was mastering the use of forward kinematics. Whilst this was the first topic taught in the module, there was some difficulty in applying the concept of axes to the robot. This meant that whilst programming in some values for the robot, there was some confusion about which axis pointed in which direction. Once this problem was overcome, there were no major issues in programming the robot for the pick-and-place demonstrations. This course has taught every group member about Arduino programming, a detailed analysis of matrices and the main functions/requirements of a robot.

VI – Appendix

A. MATLAB code for forward kinematics

```
% theta are the joint variables and each theta correspond to joint 1, 2 and
% 3 respectfully IN DEGREES
theta1 = 0;
theta2 = 0;
theta3 = 0;
L2 = 0.094; % link length between joint 2 and joint 3 in meters
L3 = 0.18; % link length between joint 3 and end-effector which is gripper


% The transfer function from base to joint 1
T01 = [cosd(theta1) -sind(theta1) 0 0;
    sind(theta1) cosd(theta1) 0 0;
    0 0 1 0;
    0 0 0  1];


% The transfer function from joint 1 to joint 2
T12 = [cosd(theta2) -sind(theta2) 0 0;
    0 0 1 0;
    -sind(theta2) -cosd(theta2) 0 0;
    0 0 0 1];


% The transfer function from joint 2 to joint 3
T23 = [cosd(theta3) -sind(theta3) 0 L2;
    -sind(theta3) -cosd(theta3) 0 0;
    0 0 -1 0;
    0 0 0 1];


% The position vector with respect to frame{3}
P3 = [L3;
    0;
    0;
    1];


% The transfer function from base to joint3, where the Gripper is attached to
T03 = T01*T12*T23


% The position of the gripper with respect to frame{0} (the base frame)
P0 =T03*P3
```

B. MATLAB code for inverse kinematics

```
% The x, y, z coordinates of the position of the end-effector are inputted.
X = -0.08;
Y = 0.05;
Z = 0.1;


Z = -Z;


% Calculating the intermediate angles required to calculate our joint
% angles.
alpha = atan2(Z, sqrt(X^2+Y^2))


% Link Lengths
L2 = 0.094; % length between joint 2 and joint 3 in meters
L3 = 0.180; % length between joint 3 and end-effector which is gripper




costheta2 = (L2^2+X^2+Y^2+Z^2-L3^2)/(2*L2*sqrt(X^2+Y^2+Z^2)) % Using the
cosine rule.
sintheta2 = sqrt(1-costheta2^2) % Using trig identities.


costheta3 = (L2^2+L3^2-X^2-Y^2-Z^2)/(2*L2*L3)
sintheta3 = sqrt(1-costheta3^2)


% Calculating Joint angles:
theta1 = atan2(Y,X) * (180/ pi) % Joint 1 angle
theta2_1 = (alpha+atan2(sintheta2,costheta2) )* (180 / pi) % Joint 2 angle -
'elbow-up' case
theta2_2 = alpha-atan2(sintheta2,costheta2) % Joint 2 angle - 'elbow-down'
case


theta3 = (pi - atan2(sintheta3,costheta3) )* (180/ pi) % Joint 3 angle
```

C. Trajectory planning MATLAB code

```
X0 = 0.04;
Y0=0.03;
Z0=0.02;

% Final point
ufx=0.05;
ufy=0.0;
ufz=0.1;

% time
tf = 5;
t=0:0.1:tf;
t=t';

% a0,a1,a2,a3 parameters
a0x=X0;
a0y=Y0;
a0z=Z0;

% Calculate the parameters using simultaneous equations.
a2x=3/(tf^2)*(ufx-X0);
a2y=3/(tf^2)*(ufy-Y0);
a2z=3/(tf^2)*(ufz-Z0);
a3x=-2/tf^3*(ufx-X0);
a3y=-2/tf^3*(ufy-Y0);
a3z=-2/tf^3*(ufz-Z0);

% Polynomials for each X, Y, Z component
ux=a0x*ones(length(t),1)+a2x*t.^2+a3x*t.^3;
uy=a0y*ones(length(t),1)+a2y*t.^2+a3y*t.^3;
uz=a0z*ones(length(t),1)+a2z*t.^2+a3z*t.^3;

% Combining all X, Y, Z components of distance to find the Euclidean distance
u = sqrt(uz.^2+uy.^2+ux.^2);

% Plot a graph of the change in X, Y, Z values over time. A linear line is
% observed in 3D axes.
figure
plot3(ux,uy,uz), title('Change in the position of the end-effector')

grid on
xlabel('X')
ylabel('Y')
zlabel('Z')

% Plot a graph of Displacement against time.
figure
plot(t,u),title('Displacement-Time')
grid on
xlabel('time(s)')
ylabel('Displacement(m)')

% Polynomials for the velocity in the X, Y and Z directions respectively.
% The first derivative of displacement with respect to time.
uvelx=2.*a2x*t+3.*a3x*t.^2;
uvely=2.*a2y*t+3.*a3y*t.^2;
uvelz=2.*a2z*t+3.*a3z*t.^2;
```

```
% The total velocity
uvel = sqrt(uvelz.^2+uvely.^2+uvelx.^2)

% Polynomials for the acceleration in the X, Y and Z directions
% respectively.
% The second derivative of displacement with respect to time.
uaccx=2.*a2x+6.*a3x*t;
uaccy=2.*a2y+6.*a3y*t;
uaccz=2.*a2z+6.*a3z*t;

% The total acceleration.
uacc = sqrt(uaccz.^2+uaccy.^2+uaccx.^2)

uacc(27:51) = -uacc(27:51)

% Plot a graph of Acceleration against time.
figure
plot(t,uacc),title('Acceleration-Time')
grid on
xlabel('time(s)')
ylabel('Acceleration (m/s^2)')

% Plot a graph of Velocity against time.
figure
plot(t,uvel),title('Velocity-Time')
grid on
xlabel('time(s)')
ylabel('velocity (m/s)')
```