Imperial College London

Department of Electrical and Electronic Engineering

ELEC60030: Robotic Manipulation

Team Machina: OpenMANIPULATOR-X Coursework Report

Authors

Khayle Torres
CID: 01753211
kt1719@ic.ac.uk

Xin Wang
CID: 01735253
xw2519@ic.ac.uk

Yuna Valade
CID: 01765409
yv19@ic.ac.uk

March 30, 2022

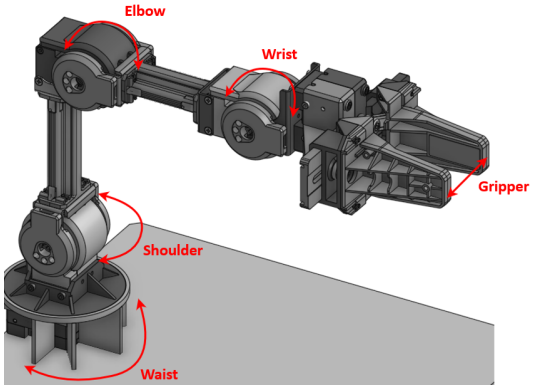
Contents

1 Task 1 - Modelling

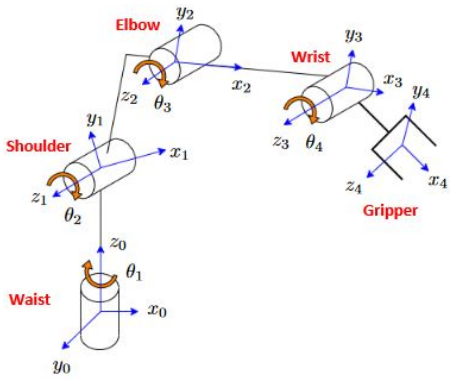
1.1 Assigning Co-ordinate Frames

The process of assigning the co-ordinate frames and the design decisions taken are summarised into the following points:

- Based on the CAD models [2] of the robotic arm, the team assigned labels to each of the joints as shown in Figure 1(a).
- Where the arm has a joint, the team assigned an inital co-ordinate frame to it as shown in Figure 1(b).



(a) Location of robotic arm joints and labels

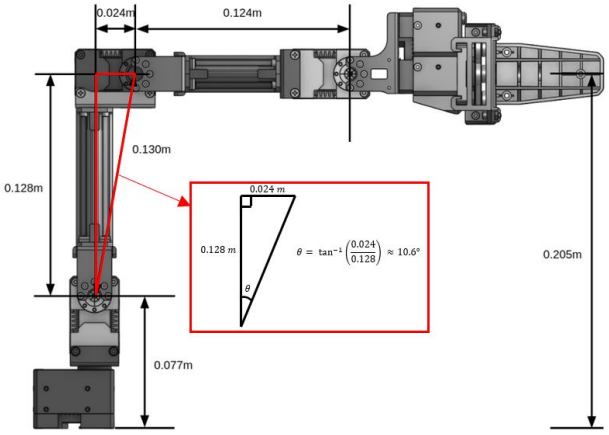


(b) Initial assigment of co-ordinate frames [1]

Figure 1

The actual model from which the DH table is generated is modified with the following design decisions:

- The co-ordinate frame for the *Waist* and *Shoulder* both have the same height z. By setting it to the same height, it simplifies the final DH table since we do not have to account for the different heights.
- There is a offset in the horizontal x axis between the *Shoulder* and *Waist* co-ordinate frames as shown on the right. Instead of accounting for the z-axis offset and x-axis offset, the length of the hypotenus ($0.130m$) and the angle subtended ($10.6\,\text{deg}$) is used. This further simplifies the resulting DH table shown in the next section.



- The robotic arm's end effector location is designed to be in the center by default but its final position can be changed depending on the task e.g. the end effector location for drawing will be different to the end effector location for interacting with cubes.

1.2 DH Table

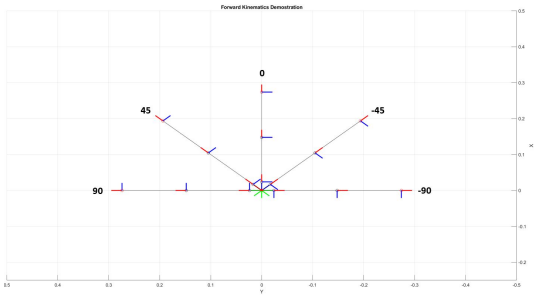
The DH table is created based on the standard robotic arm position as show in Figure 1(a), and the co-ordinate frames for *Waist* and *Shoulder* are in the same location with different orientations.

	α_{i-1}	a_{i-1}	d_i	θ_i
Waist	0	0	0.077	θ_1
Shoulder	90	0	0	$\theta_2 + 90 - 10.6$
Elbow	0	0.130	0	$\theta_3 - 90 + 10.6$
Wrist	0	0.124	0	θ_4
Gripper	0	0.126	0	0

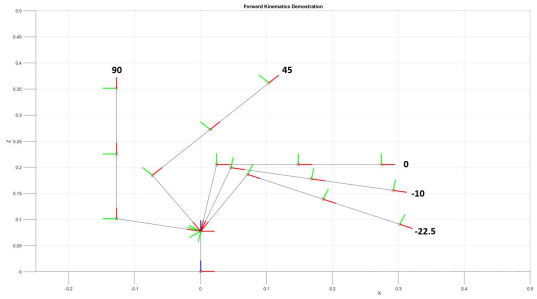
1.3 Forward Kinematics

With the DH table completed, the Forward Kinematics is implemented by calculating the transformation matrix for each joint and multiplying them together. This functionality is implemented as a `trajectoryLib` class function that returns x, y, z co-ordinate of the end-effector (*Gripper*) when the angles for each joint is specfied.

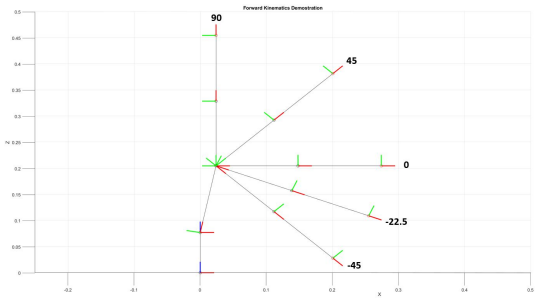
Referring to the *Shoulder* and *Elbow* in the DH table, there are additions of $+90°$ and $-90°$ respectively in order to ensure the standard pose [2] is achieved with all joint angles set to $0°$. This is seen in the following diagrams showing the range of rotations for each joint with all other joints set to 0.



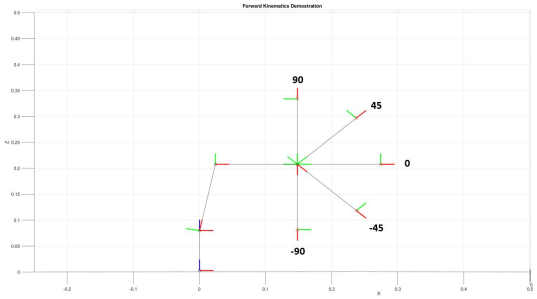
(a) θ_1 rotation: $[90°, 45°, 0°, -45°, -90°]$



(b) θ_2 rotation: $[90°, 45°, 0°, -10°, -22.5°]$



(c) θ_3 rotation: $[90°, 45°, 0°, -22.5°, -45°]$



(d) θ_4 rotation: $[90°, 45°, 0°, -45°, -90°]$

1.3.1 Implementing Joint Co-ordinate Frames

The co-ordinate frames of the arm joints move along with the joint so the positions of the x, y, z axis is dependent on the rotation of the joints i.e. θ_i where i denotes the specific arm joint. Due to this dependency, the co-ordinate frames are implemented with a variation of Forward Kinematics where the x, y, z axis are calculated based on the θ_i value of the associated joint. For example, the co-ordinate frame of the *Waist* with rotation θ_1 is calculated using Forward Kinematics as follows:

$$Waist_x = \begin{bmatrix} c(\theta_1) & -s(\theta_1) & 0 & 0.021 \\ s(\theta_1) & c(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} Waist_y = \begin{bmatrix} c(\theta_1) & -s(\theta_1) & 0 & 0 \\ s(\theta_1) & c(\theta_1) & 0 & 0.021 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} Waist_z = \begin{bmatrix} c(\theta_1) & -s(\theta_1) & 0 & 0 \\ s(\theta_1) & c(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0.021 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the value 0.021 represents the length of an axis of the co-ordinate frame.

This functionality is implemented as a separate function `FK_coordinate_frames` that returns the x, y, z positions relative to the joint for the 5 joints of the arm.

1.4 Inverse Kinematics

The inverse kinematics problem for the OpenManipulator-X arm is solved through a geometric approach that is covered in depth in the Appendix. The following equations are the solutions to the inverse kinematics problem given co-ordinates (x, y, z):

$$\textbf{Waist} : \theta_1 = \begin{cases} \arctan(\frac{y}{x}) + 180 & x < 0 \text{ and } y > 0 \\ \arctan(\frac{y}{x}) - 180 & x < 0 \text{ and } y < 0 \\ \arctan(\frac{y}{x}) & \text{otherwise} \end{cases}$$

$$\textbf{Shoulder} : \theta_2 = \arctan\left(\frac{\frac{L_2 + L_3 * c(\theta_3) * R_2 - L_3 * s(\theta_3) * R_2}{R_2^2 + Z_2^2}}{\frac{L_2 + L_3 * c(\theta_3) * Z_2 - L_3 * s(\theta_3) * Z_2}{R_2^2 + Z_2^2}}\right)$$

$$\textbf{Elbow} : \theta_3 = -\arccos\left(\frac{R_2^2 + Z_2^2 - L_2^2 + L_3^2}{2 * L_2 * L_3}\right)$$

$$\textbf{Wrist} : \theta_4 = \phi - \theta_3 - 90 + \theta_2$$

where:

- ϕ is the orientation of the end effect that is specified by the user
- Robotic arm lengths [2]: $L_2 = 0.130$, $L_3 = 0.124$ and $L_4 = 0.126$
- $R_2 = |\sqrt{x^2 + y^2}| - L_4 * \cos(\phi)$
- $Z_2 = z - 0.077$

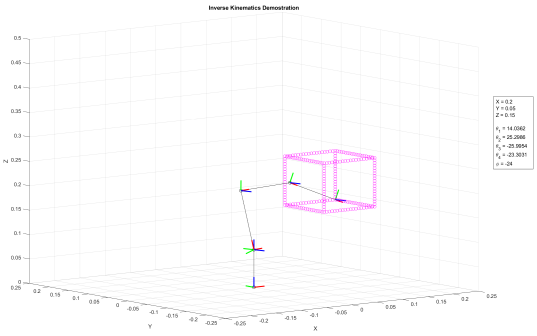
Further conversions are required to convert the angles outputted from inverse kinematics to the angles compatible with the openManipulator-X arm servos as defined in the Appendix.

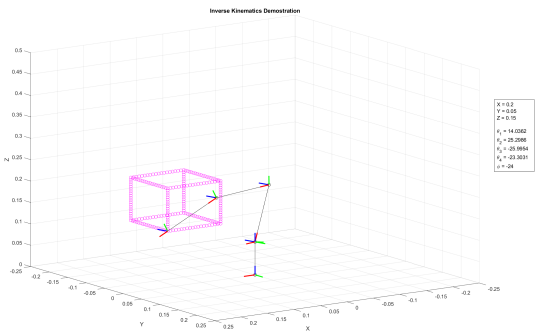
1.4.1 Inverse Kinematics Simulation: Tracing a square on each cartesian plane

To test the functionality of the inverse kinematic solver, the robotic arm draws a cube in 3D space. The process is summarised as follows:

- Start and end co-ordinate of each line of the cube is specified
- Using matlab `linspace`, the co-ordinates of the points between the start and end points are generated
- These sets of co-ordinates are concatenated together in the order the arm traverses over them
- The combined set is iterated in a FOR loop, each co-ordinate fed into the `IK` function that specifies the joint angles required to acheive the specified co-ordinate

In the simulation, the plot also displays the joint angle values of θ_1, θ_2, θ_3 and θ_4.





2 Task 2 - Pick and Place Cubes

Task 2 was broken down into several modular components that enabled the team to divide the work and tackle it separately.

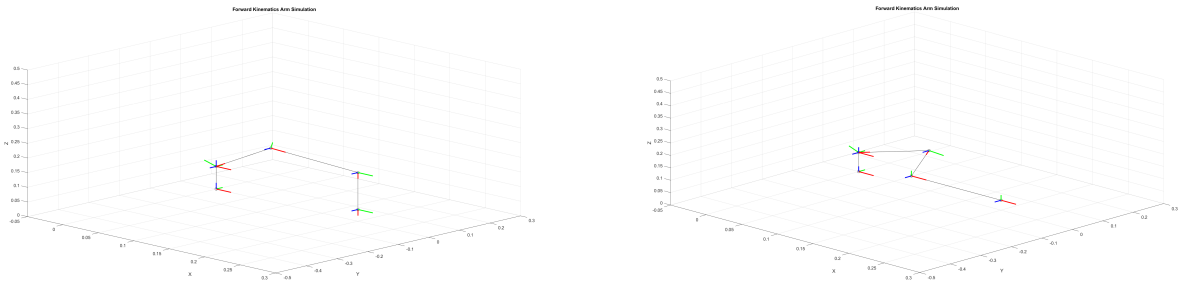
1. **Picking up and dropping function**

   Given the (x, y, z) co-ordinates of the cubes, the Inverse Kinematics solver will calculate the servo angles required to reach that co-ordinate. In the Inverse Kinematics solver, the team has added multiple checks that ensures only reachable co-ordinates are allowed to be fed into the arm.

   For picking and dropping tasks, the team made the design decision to only pick up and drop cubes from the top ($\phi = -90$). This was due to the fact that picking/dropping from horizontal direction ($\phi = 0$) would cause the gripper extrusions to hit the cube holders and increasing the height at which the gripper grips the cube would result in a very loose grip.

2. **Rotate cube function**

   Rotating cubes are divided into two types: Rotating forward and rotating backward. The key to completing this task was understanding that ϕ determines the final orientation of the end-effector (*Gripper*). With that, a cube can be rotated by picking it up with $\phi = 0$ and dropping it in the same place with $\phi = -90$, and vice versa for the opposite direction.



(a) Picking up cube: $\phi = -90°$ (b) Dropping cube: $\phi = 0°$

Figure 4: Depiction of backward rotation

3. **Stacking cube function**

   The action of stacking is a combination of rotating and picking up/dropping cubes. Therefore, the arm is designed to first rotate all the cubes then proceed to picking up and drop the cubes at a height increasingly $0.025m$ higher to account for the height of the already stacked cubes.

4. **Co-ordinates of sockets on the acrylic board**

   With the OOP approach, the co-ordinates of every socket on the acrylic board could be stored as three matrices representing the x, y and z values of a specific socket. To obtain the co-ordinates of a socket, the row and column of the desired socket is specified and the function returns the (x, y, z) co-ordinates for the associated row and column.

5. **Linear interpolation**

   Basic linear interpolation is implemented to prevent the arm moving in an undesired way. This is important for picking/dropping and rotating cubes where, without linear interpolation, could approach the cube holders from the sides and hit the protruded edges of the cube holders. For stacking, the behaviour of the arm for a particular action would be unpredictable and acidentally hit the stacked cubes.

   Linear interpolation affects the action of moving down and moving up from cube co-ordinates. For any particular cube co-ordinate, the arm would first move to directly that co-ordinate with a z-offset of at least $+0.030m$. Once there, there are 3 waypoints, each with a decreasing z-offset where the last waypoint is the co-ordinate to pick up/drop off the cube. The reverse happens for the arm moving back up.

   For transit i.e. when the arm picks up a cube and moving to another co-ordinate to drop off the cube, no linear interpolation is required. The arm automatically determines the best way to reach the drop-off co-ordinate with a z-offset of at least $+0.030m$.

6. **Calibration mechanisms**

3 Task 3 - Trajectory Following (Drawing)

3.1 Picking up the pen

3.2 Drawing

4 Task 4 - Musical instruments

For Task 4, the team chose to perform a musical piece using drums and a xylophone. Music is complex to the robotic arm in the sense that the timings at which the notes are hit must be approximately correct. This meant that the speeds of the arm servos are all different and changes continuously throughout the musical performance. If any servos was wrongly calibrated or the arm does not hit the instruments fast enough, the musical piece would immediately sound wrong. This challenge was interesting to the team and we felt that it was a suitable task to show the skills we learnt throughout the course of this module and through working with the robotic arm.

This task was broken down into the following steps:

1. Designing the gripper to hold the xylophone stick
2. Placing musical instruments in reachable range and using forward kinematics to note down the coordinates of drums and xylophone tone bars

   The DH tables are modified to account for the increased length of the end effector as shown below:

	α_{i-1}	a_{i-1}	d_i	θ_i
Waist	0	0	0.077	θ_1
Shoulder	90	0	0	$\theta_2 + 90 - 10.6$
Elbow	0	0.130	0	$\theta_3 - 90 + 10.6$
Wrist	0	0.124	0	θ_4
Gripper	0	0.126	0	0

3. Creating functions to hit drums and specified tone bars

   Functions were created that handled the actions of hitting drums and tone bars. With the OOP approach taken by the team, many pieces of code could be re-used and slightly adjusted for this particular task. This modular approach code allowed the team to easily identify and finely adjust factors on the fly such as velocity and acceleration.
4. Calibrating the hit action
5. Combining hits to produce a musical piece

5 Appendix

5.1 Task 1: Inverse Kinematics

The process to solve for the Inverse Kinematics equations is split into two sections: $x - y$ domain and $r - z$ domain.

5.1.1 $x - y$ domain

The angle for the *Waist* joint, given co-ordinates (x, y, z) can be calculated as:

$$\theta_1 = \arctan\left(\frac{p_y}{p_x}\right)$$

Note that the *Waist* joint rotation range is between $-180°$ to $180°$ with the middle point being $0°$. Therefore it is important to ensure that the angles are within that range for any combination of p_x and p_y.

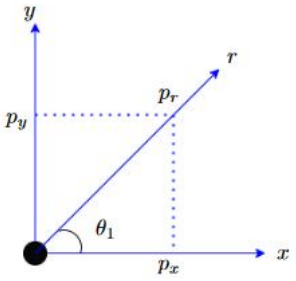
$$\theta_1 = \begin{cases} \arctan(\frac{y}{x}) + 180 & x < 0 \text{ and } y > 0 \\ \arctan(\frac{y}{x}) - 180 & x < 0 \text{ and } y < 0 \\ \arctan(\frac{y}{x}) & \text{otherwise} \end{cases}$$

5.1.2 $r - z$ domain

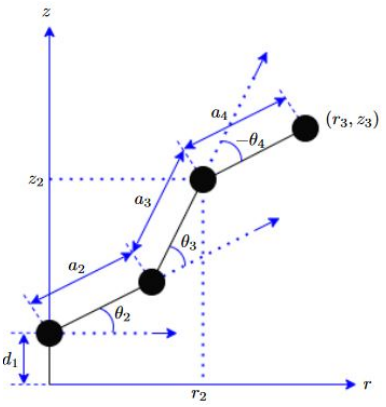
Angles for the *Shoulder* (θ_2), *Elbow* (θ_3) and *Wrist* (θ_4) are solved in the $r - z$ domain. The $r - z$ domain is a simplified view of the 3D space (x, y, z) where the x and y axis are merged using the Pythagorean equation

$$p_r = \sqrt{p_x^2 + p_y^2}$$

to define the r axis as shown in the image Figure 4(a).



(a) r-axis a result of merging the x and y axis



(b) Robotic arm projected onto the $r - z$ domain

Figure 5: [1]

In order to solve the inverse kinematic equations, the variable term ϕ has to be defined which, in this case, is defined as the sum of the angles

$$\phi = \theta_2 + \theta_3 + \theta_4$$

The co-ordinates (r_3, z_3) could easily be calculated given co-ordinates (x, y, z):

$$r_3 = \sqrt{x^2 + y^2}$$
$$z_3 = z - d_1$$

With (r_3, z_3) and ϕ, the co-ordinates (r_2, z_2) can be found:

$$r_2 = r_3 - a_4 \cos(\phi)$$
$$z_2 = z_3 - a_4 \sin(\phi)$$

The *Elbow* joint angle can therefore be defined as:

$$\theta_3 = \pm \arccos\left(\frac{r_2^2 + z_2^2 - (a_2^2 + a_3^2)}{2a_2a_3}\right)$$

With θ_3, co-ordinates (r_2, z_2) can be alternatively defined using θ_2 and θ_3:

$$r_2 = a_2\cos(\theta_2) + a_3\cos(\theta_2 + \theta_3)$$
$$z_2 = a_2\sin(\theta_2) + a_3\sin(\theta_2 + \theta_3)$$

Expanding with Ptolemy's identities:

$$r_2 = \cos(\theta_2)\left(a_2 + a_3\cos(\theta_3)\right) - a_3\sin(\theta_2)\sin(\theta_3)$$
$$z_2 = a_3\cos(\theta_2)sin(\theta_3) + \sin(\theta_2)(a_2 + a_3\cos(\theta_3))$$

By recognising the hypotenus, adjacent and opposite of the triangle subtended by θ_2:

$$\cos(\theta_2) = \frac{r_2(a_2 + a_3\cos(\theta_3)) + z_2(a_3\sin(\theta_3))}{r_2^2 + z_2^2}$$
$$\sin(\theta_2) = \frac{z_2(a_2 + a_3\cos(\theta_3)) + r_2(a_3\sin(\theta_3))}{r_2^2 + z_2^2}$$

Thus the angle θ_2 can be found:

$$\theta_2 = \arctan\left(\frac{\sin(\theta_2)}{\cos(\theta_2)}\right)$$

By using the relation of ϕ, θ_2, θ_3, θ_4 can be found:

$$\theta_4 = \phi - \theta_2 - \theta_3$$

6 References

References

[1] Mohd Hairi Mohd Zaman. “Kinematic Modeling of A Low Cost 4 DOF Robot Arm System”. In: (Nov. 2020). DOI: 10.30534/ijeter/2020/328102020.

[2] Robotis OpenMANIPULATOR-X. *Robotis e-Manual: Specification*. DOI: https://emanual.robotis.com/docs/en/platform/openmanipulator_x/specification/.