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 31, 2022

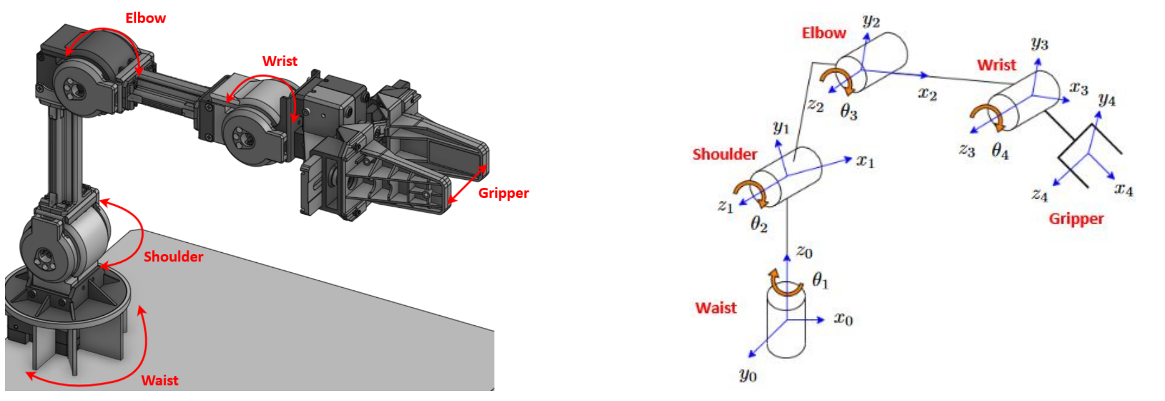
Contents

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 an 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 ($10.6\deg$) is used. This further simplifies the resulting DH table shown in the next section.

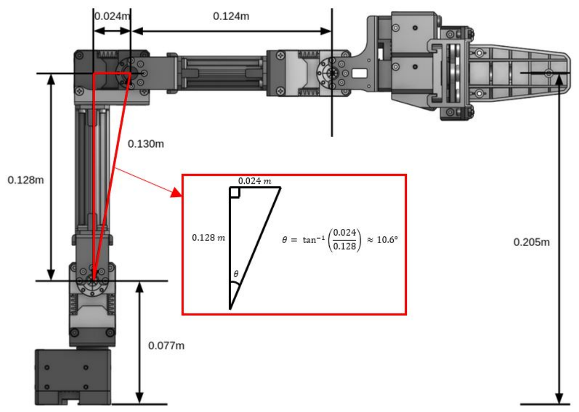

0.024m
0.124m
0.130m
0.128m
0.024 m
0.128 m
$\theta = \tan^{-1}\left(\frac{0.024}{0.128}\right) \approx 10.6°$
θ
0.205m
0.077m

Figure 2

- 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

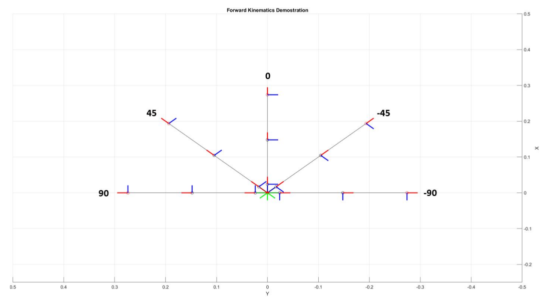
Referring to the *Shoulder* and *Elbow* in the DH table, there are additions of $+90° - 10.6°$ and $-90° + 10.6°$ respectively in order to ensure the standard pose in Figure 2 [2] is achieved with all joint angles set to $0°$. 10.6

1.3 Forward Kinematics

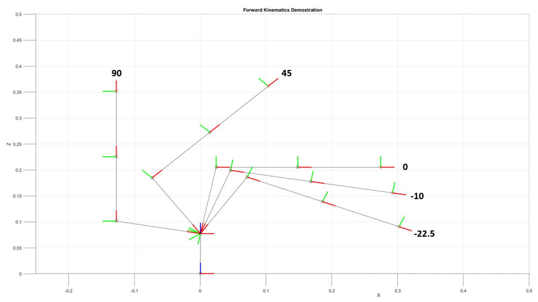
With the DH table completed, the Forward Kinematics is implemented by calculating the transformation matrix for each joint and multiplying them together. This functionality is implemented as a `trajectoryLib` class function that returns x, y, z co-ordinate of the end-effector (*Gripper*) when the angles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

Each frame has it's own transformation matrix that is derived from each row on the specified DH table above. Each transformation matrix is relative to previous frames hence why all transformation matrices are multiplied together. Each consecutive frame is built on the cumulative rotations and transformations of all the frames that came before it. The forward kinematic matrices is shown under `Appendix: Forward Kinematics`

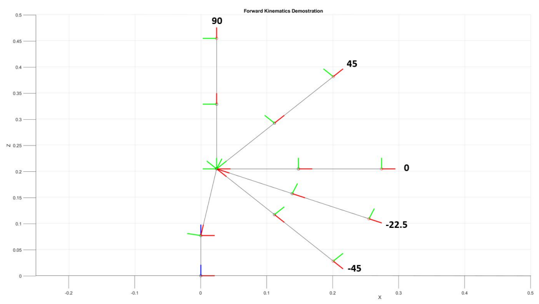
As mentioned previously, this is seen in the following diagrams showing the range of rotations for each joint with all other joints set to 0.



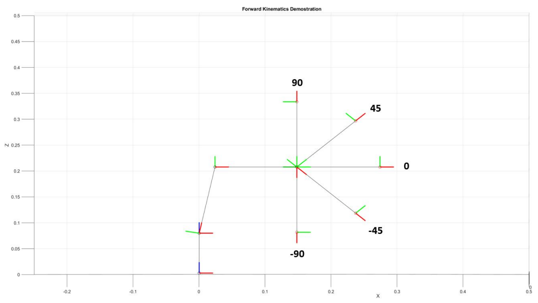
(a) θ_1 rotation: $[90°, 45°, 0°, -45°, -90°]$



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



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



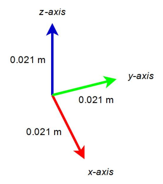
(d) θ_4 rotation: $[90°, 45°, 0°, -45°, -90°]$

1.3.1 Plotting XYZ-axis indicators on Co-ordinate Frames

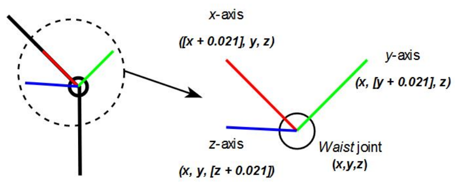
The XYZ-axis indicators of the arm joints are dependent on the current rotation of the associated joint which, itself, is dependent on the world and any previous joints related to it i.e. θ_i where i denotes the specific arm joint. Due to this dependency, the xyz-axis indicators are implemented with 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 the line of the xyz-axis indicator.



(a) Dimensions of co-ordinate frame



(b) Relation of the axis of joint co-ordinate frame to the co-ordinate of the joint

This functionality is implemented as a separate function `FK_coordinate_frames` that returns the x, y, z positions relative to the joint for each of 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) with robotic arm lengths [2]: $a_2 = 0.130$, $a_3 = 0.124$ and $a_4 = 0.126$:

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

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

$$\textbf{Elbow} : \theta_3 = -\arccos\left(\frac{r_2^2 + z_2^2 - a_2^2 + a_3^2}{2 * a_2 * a_3}\right)$$

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

where:

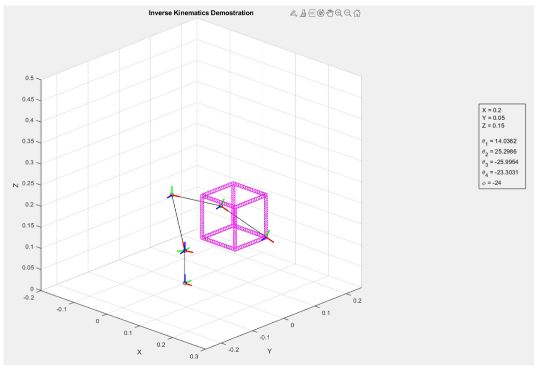
- ϕ is the orientation of the end effect that is specified by the user
- $r_3 = |\sqrt{x^2 + y^2}|$
- $z_3 = z - 0.077$
- $z_2 = z_3 - a_4 \sin(\phi)$
- $r_2 = r_3 - a_4 \cos(\phi)$
- $\sin(\theta_2) = \frac{a_2 + a_3 * \cos(\theta_3) * r_2 - a_3 * \sin(\theta_3) * r_2}{r_2^2 + z_2^2}$
- $\cos(\theta_2) = \frac{a_2 + a_3 * \cos(\theta_3) * z_2 - a_3 * \sin(\theta_3) * z_2}{r_2^2 + z_2^2}$

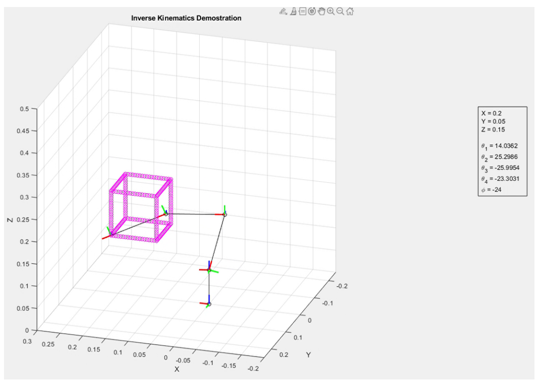
Further conversions are required to convert the angles outputted from inverse kinematics to the angles compatible with the openManipulator-X arm servos as defined in the `Appendix: Task 1: Inverse Kinematics`.

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 achieve the specified co-ordinate





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**

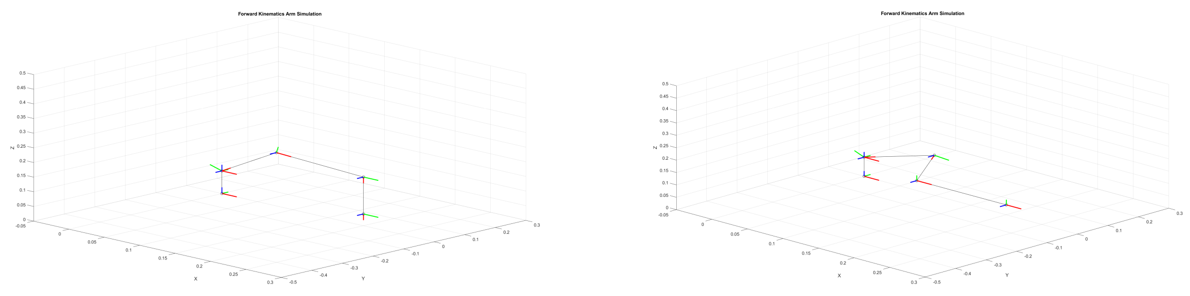
   Given the (x, y, z) co-ordinates of the cubes, the Inverse Kinematics solver will calculate the servo angles required to reach that co-ordinate. In the Inverse Kinematics solver, the team has added multiple checks that ensures only reachable co-ordinates are allowed to be fed into the arm.

   For picking and dropping tasks, the team made the design decision to only pick up and drop cubes from the top ($\phi = -90°$) if it is reachable. Otherwise if the cube is too far then the robot would pick and drop cubes from the middle ($\phi = 0°$) as this is able to extend the range by a tiny amount. However picking up at ($\phi = 0°$) has to be shifted up as picking up from the middle causes the cube holders to be gripped. Some ranges of close range picking up/dropping can't be achieved with ($\phi = 0°$) thus a cube is picked up/dropped at ($\phi = -90°$) for close ranges. However if a cube is picked from far range ($\phi = 0°$) and dropped at close range ($\phi = -90°$) or vice versa the appropriate transformations would have to be taken into account due to the cube being shifted up when being picked by ($\phi = 0°$).

   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 6: 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

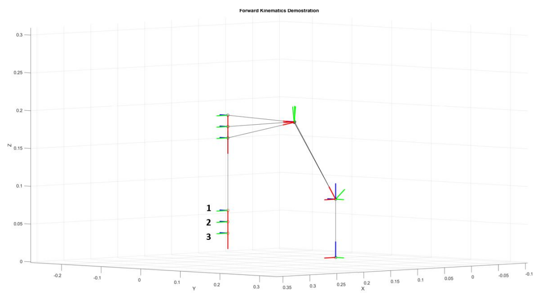


Figure 7: Illustration of arm going down to pick up a cube with linear interpolation

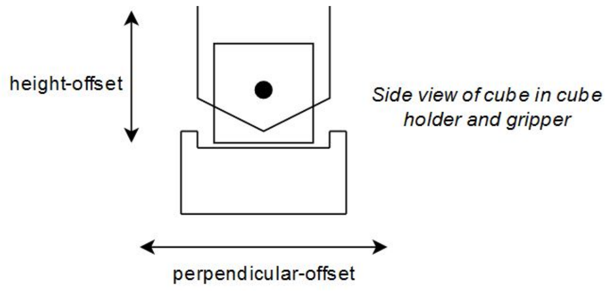
least $+0.030m$. Once there, there are 3 waypoints, each with a decreasing z-offset where the last waypoint is the co-ordinate to pick up/drop off the cube. The reverse happens for the arm moving back up.

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 $+0.030m$. This approach was taken to ensure the smoothest possible movement without implementing velocity control.

To further ensure smooth motion, we investigated the concept of writing the next waypoint to the servo motors when travelled a certain distance between the two waypoints it is currently trying to get to. This is due to the robot having it's own interpolation when in position control mode thus resulting in a stuttery movement when using linear interpolation naively. Provided the next waypoint is not the final one, the current distance travelled between the two waypoints (starting and finishing) is calculated. Using this, a percentage of how much the robot has travelled between the two waypoints is also calculated and when the robot has travelled a certain proportion of the overall distance the next waypoint will be written. This is because the robot's stuttering is due to the robot slowing down before reaching it's waypoint (from it's own interpolation) thus if the robots waypoint is overwritten when it still has not slowed down, this results in a smoother motion.

6. **Calibration mechanisms**

   There is a certain amount of uncertainty with the robotic arm and, with the team working with different arms each time, a system had to be implement to be able to streamline the calibration of the arm. Each function related to manipulation such as `move_cube_coord` has two calibration parameters:



   - `height_offset`: Increases/decreases the z co-ordinate
   - `perpendicular_offset`: Increases and decreases the perpendicular distance e.g. dropping the cube a bit more forward or bit more backward

3 Task 3 - Trajectory Following (Drawing)

Task 3 is split into two components with sub-components:

- Pick up a pen and placing it back
- Drawing a given pattern composed of:
  - Three straight lines in different directions such as horizontal, vertical, and diagonal
  - A circle with a given radius, center point and degree of rotation

The team's approach to solving Task 3 is summarised as follows:

1. A visualiser i.e. plotting function to see the pattern a set of generated co-ordinates would create
2. Modifying the DH table. The modified DH table can be seen in `Appendix: Task 2 and 3 modified DH Table`
3. Designing the pen grippers:

   When designing the pen gripper, two main factors had to be taken into account:

   - Ensure the pen wouldn't slip out as the robot applies pressure on it when drawing
   - Ensure the stability of the pen to obtain the straightest lines possible when drawing

   It was noted that the pen provided had an outline that slightly decreases towards the tip. The pen gripper surface accounts for that decrease in gradient to ensure the tightest grip possible. The CAD model can be accessed under `Appendix: Task 2 and 3 CAD models`
4. Implementing global parameters that would generalise the drawing process allow different patterns to be created with minimal changes
5. Concatenating all the generated co-ordinates into a large array, converting to angles through Inverse Kinematics and feeding all the angles to the robot

For drawing, different approaches were used for drawing lines and circles:

- **Straight lines**: Having specified the start and end co-ordinates, Matlab's built-in colon operator generates evenly spaced vectors of i increments. This allowed the team to control how many way-points are created. With the horizontal line, the x-axis changes while the y-axis remains the same. The reverse occurs for vertical lines. For diagonal lines, it is drawn by considering it as a function $y = x$ where x and y increments simultaneously. The orientation of the diagonal can be chosen by setting accordingly the start and end point as well as modifying the increment to a negative or positive slope depending on the wanted orientation of the diagonal.

  Note that the commonly thought x and y axis are inverted due to our frame choice.

- **Circles**: Circles are drawn using a derivation of the classical circle equation:

$$x = \text{radius} * \cos(\theta) + \text{center point x}$$
$$y = \text{radius} * \sin(\theta) + \text{center point y}$$
$$x^2 + y^2 = r^2$$

  From the inputted degree of the circle, the equivalent radian value is calculated and θ values to reach it from the starting point on the circle are obtained. The `linspace` function was used to increment theta and define how many points are fed to the robot. Using the circle equation and considering the circle center and radius, we calculate the corresponding x and y coordinates for each theta angles. This process ensures a well-defined and precise circle.

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

For the xylophone gripper, a similar approach to the pen holder was taken. As the xylophone's stick radius is uniform across its whole length the designed grippers were slightly loose on some movements. We adapted by wrapping the stick with a rubbery material which fixed our issue. The CAD model can be accessed under `Appendix: Task 2 and 3 CAD models`

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

   To account for the change in gripper design, the original DH table is modified. The modified DH table can be seen in the `Appendix: Task 2 and 3 modified DH Table`

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 and combining hits to produce a recognisable piece of music

   Calibrating the hit action involved 3 tasks: Timing the hit properly and hitting the xylophone with the right power and accuracy.

   - Accuracy can be achieved by simply modifying the DH table to account for the difference in gripper length and also calibrating taking into account the weight of the xylophone stick.
   - Timing can be controlled by 2 parameters, the height in which the robot gripper would hover above the xylophone and the speed of some motors (more specifically motors 11 and 14). Decreasing the height means the robot would need to travel less distance in order to hit a key on the xylophone. Incresing the motor speed means each hit will take less time as the robot will get to it's destination faster. The 2 most important motors to modify is motor 11 (θ_1) and motor 14 (θ_4). Throughout a song, both parameters are modified to either increase or decrease the tempo. A tradeoff of both are important to ensure appropriate power when hitting a key is also satisfied.
   - Power is simply controlled by the motor speed servo motor 14 (θ_4). However power is also linked with timing (when power is decreased, the travel time increases) thus the appropriate height has to be modified accordingly

5 Appendix

5.1 Task 1: Forward Kinematics

Note the dimensions [2]:

- Height offset from base to *Waist* joint: 0.077 m
- Length of hypotenus from *Waist* joint to *Elbow* joint: 0.130 m
- Length from *Elbow* joint to *Wrist* joint: 0.124 m
- Length from *Wrist* joint to *Gripper* end effector: 0.126 m

Forward kinematics matrices:

$$T_{\text{Waist}}^{\text{Base}} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0.077 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{\text{Shoulder}}^{\text{Waist}} = \begin{bmatrix} \cos(\theta_2 + 10.62) & -\sin(\theta_2 + 10.62) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_2 + 10.62) & \cos(\theta_2 + 10.62) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{\text{Elbow}}^{\text{Shoulder}} = \begin{bmatrix} \cos(\theta_3 - 10.62) & -\sin(\theta_3 - 10.62) & 0 & 0.130 \\ \sin(\theta_3 - 10.62) & \cos(\theta_3 - 10.62) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{\text{Wrist}}^{\text{Elbow}} = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & 0.124 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{\text{Gripper (end-effector)}}^{\text{Wrist}} = \begin{bmatrix} 1 & 0 & 0 & 0.126 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2 Task 1: Inverse Kinematics

The process to solve for the Inverse Kinematics equations given a set of co-ordinates (x, y, z) is split into two sections: $x - y$ domain and $r - z$ domain.

Note the dimensions of the robotic arm [2]: $a_2 = 0.130, a_3 = 0.124, a_4 = 0.126$

5.2.1 $x - y$ domain

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

$$\theta_1 = \arctan\left(\frac{y}{x}\right)$$

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.

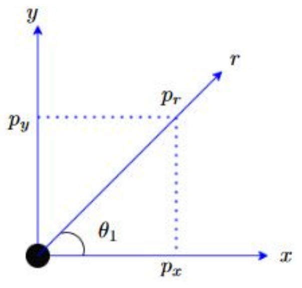
$$\theta_1 = \begin{cases} \arctan(\frac{y}{x}) + 180 & x < 0 \text{ and } y > 0 \\ \arctan(\frac{y}{x}) - 180 & x < 0 \text{ and } y < 0 \\ \arctan(\frac{y}{x}) & \text{otherwise} \end{cases}$$

5.2.2 $r - z$ domain

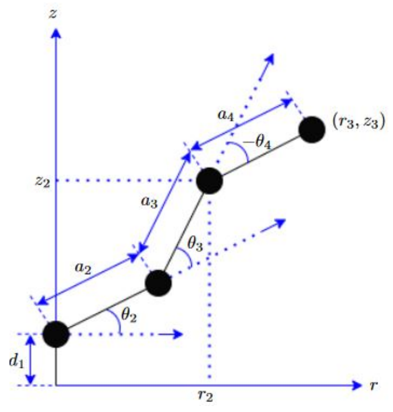
Angles for the *Shoulder* (θ_2), *Elbow* (θ_3) and *Wrist* (θ_4) are solved in the $r - z$ domain. The $r - z$ domain is a simplified view of the 3D space (x, y, z) where the x and y axis are merged using the Pythagorean equation

$$r = \sqrt{x^2 + y^2}$$

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 8: [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):

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

where $d_1 = 0.077$ m

With (r_3, r_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:

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

As mentioned previously, further conversion is before it is fed into the servo motors. The conversion equations are as follows:

$$\begin{aligned}
\theta_{1_{servo}} &= 180^\circ - \theta_{1_{IK}} \\
\theta_{2_{servo}} &= 180^\circ + (\theta_{2_{IK}} - 10.62^\circ) \\
\theta_{3_{servo}} &= 180^\circ + (-\theta_{3_{IK}} - 79.38^\circ) \\
\theta_{4_{servo}} &= 180^\circ - \theta_{4_{IK}}
\end{aligned}$$

5.3 Unity simulation

Due to the limited number of arms and initial uncertainty in lab times, the team wanted to mitigate any negative impacts that could arise. With the simulation tasks in Task 1, one of the team members with game development experience proposed a virtualised environment where we can interact with the arm with accurate dimensions and physics.

As a result, alongside simulating in Matlab, the team created a custom Unity-based simulation that allowed us to further visualise how the arm would look given specific angles and to check whether the arm would grip the cube or not. This allowed us to verify if our forward kinematics and inverse kinematics solvers are working correctly.

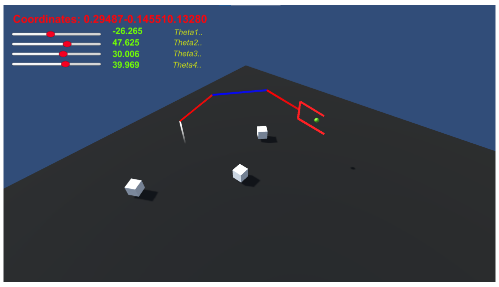
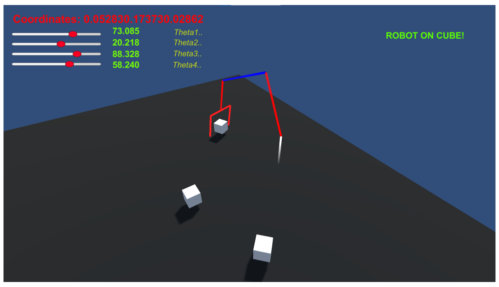
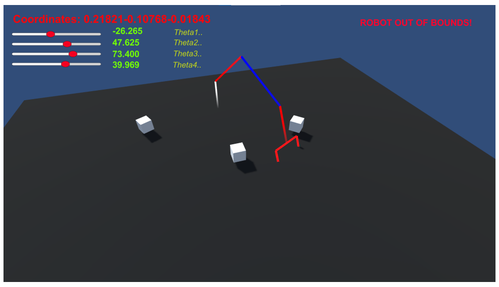


Figure 9: Virtual environment with to-scale robotic arm and cubes. Cubes are place in actual positions for Task 2



(a) Allows user to know if the position of end effector (green dot in Figure 9) is on the cube or not



(b) Checks if arm would violate the range constraints i.e. board dimensions

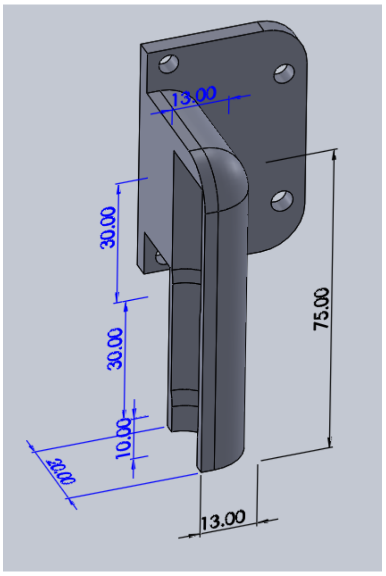
5.4 Task 2 and 3 modified DH Table

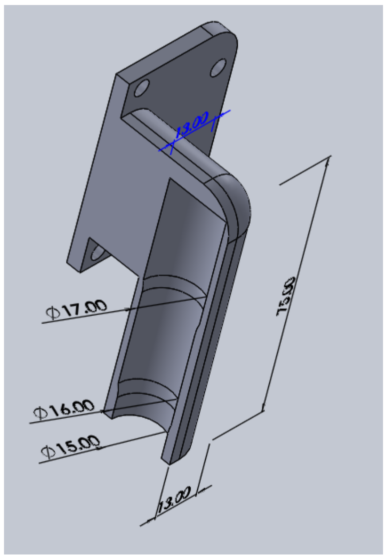
The DH tables are modified for Task 3 and Task 4 as shown below:

	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	$0°$	0.116	0	$0°$

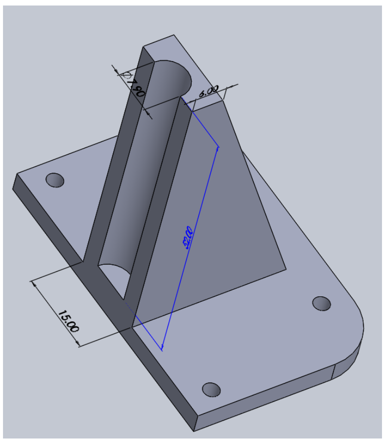
5.5 Task 2 and 3 CAD models

Pen gripper: Pen gripper is composed of a pair, each with the same designs but reflected.





Xylophone stick gripper: Xylophone stick gripper is composed of a pair, each with the same designs but reflected.



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/`.