Robotic Manipulation Report - Group B8

Ashcharya Kela (01841948), Tanmay Lad (01855634) , Eleftheria Safarika (01873841)

Contents

1 Task 1: Modelling the Robot **2**
1.1 Task 1.a. - DH Table and Coordinate Frames 2
1.2 Task 1.b. - Graphical Simulation 3
1.3 Task 1.c. - Inverse Kinematics 3
1.4 Task 1.d. - Tracing a square 4

2 Task 2: Pick and Place **4**
2.1 Cartesian Cubic Interpolation along a Straight Line Path 5
2.2 Task 2.a. - Translation 5
2.3 Task 2.b. - Re-orientation 5
2.4 Task 2.c. - Stacking Several Cubes 6

3 Task 3: Trajectory Following **6**
3.1 Task 3.a. - Holding the Pen 7
3.2 Task 3.b. - Drawing the Pattern 7

4 Task 4: Self-Selected Task **8**
4.1 Idea Selection and Motivation 8
4.2 Movements 8
4.3 3D Printing Gripper and Task Items 8

5 Appendix **9**
5.1 Transformation Matrices Calculations for Forward Kinematics 9
5.2 Servo Step Limits 10
5.3 Calculations for Inverse Kinematics 10
5.4 Calculations for Cartesian Cubic Interpolation on a Line 12
5.5 Calculations for Cubic Interpolation on Arc 14
5.6 3D Printed Parts for Task 4 16

1 Task 1: Modelling the Robot

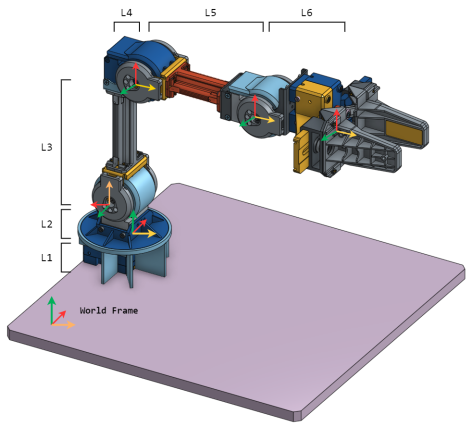
This coursework was concerned with the mathematical modelling of a 5-joint robot arm, its simulation on MATLAB, and the physical implementation of its model on the Robotis OpenManipulator-X robot. To understand how it moves in space, the robot was described in terms of the lengths of its links and of the angles of its joints. To stay consistent, we use the Denavit-Hartenberg (DH) notation.

1.1 Task 1.a. - DH Table and Coordinate Frames

The DH parameters to translate and re-orientate between frames are described first by translation a and rotation α around the x axis, followed by translation d and rotation θ around the z axis. The DH convention demands that at each link the current frame of reference has its x-axis pointing along the link, while at every rotating joint the z-axis is pointing upwards through the joint such that the rotation happens counterclockwise around it. A general transformation between frames is performed by multiplying by the below matrix.

$$T_0^1 = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

To move from frame 0 to frame 5, the product of the transformation matrix between each frame can be taken such that: $T_0^5 = T_0^1 \times T_1^2 \times T_2^3 \times T_3^4 \times T_4^5$. The chosen frames of reference following the DH notation, with annotations to indicate the link lengths and the frame transformations can be seen in Figure 1.1



i	α_{i-1}	a_{i-1}	d$_i$	θ_i
1	0	0	34mm	θ_1
2	$-\pi/2$	0	0	$-\pi/2$
3	0	43mm	0	θ_3
4	0	128mm	0	$-\pi/2$
5	0	24mm	0	θ_5
6	0	124mm	0	θ_6
7	0	126mm	0	0

Figure 1: Robot arm at home position and DH table

The values of the DH parameters can be arranged into a table called the DH table where each row of the table represents one frame transformation. Table 1.1 shows the total number of frames involved.

The first row represents link L1 of the robot. Since in our global frame the z axis is pointing upwards, the only parameter transforming the base frame to the bottom servo frame is a vertical translation along d_1. At that joint, the robot will be able to perform a panning motion, hence we can assign an angle θ_1 for an arbitrary rotation around the z axis at that frame. **The second row** is a transitionary row, where the frame at the first joint is transformed by 90 about the x-axis and then 90 °about the z-axis 90°rotations, in order to align with the desired frame at the second joint.**The third row** represents the link L2, along with an angle θ_3. **The fourth row** translates the frame upwards along L3, and rotates it at the end, to properly align the x-axis with link L4. **The fifth row** represents the small translation representing L4, as well as the angle θ_5, which can be likened to the arm's elbow. **The sixth row** contains the translation down link L5, and the angle θ_6, the angle before the arm's gripper, which can be compared to an arm's wrist. Finally, **the seventh row** translates the frame to the middle of the gripper - L6. From defined our angles θ_1, θ_3, θ_5, and θ_6 as *pan*, *tilt*, *elbow*, and *wrist* respectfully.

The frame assignments were chosen such that the entire mechanical representation of the arm was present in the DH table. It would have been possible to eliminate the short right-angle link in the fifth row of the table and consider the angle of the tilt link as offset however we found that this quickly becomes non-intuitive. Our selection of frames preserves intuition when describing joint angles: a joint angle of zero at any joint results in the next link aligning with the previous. Refer to Appendix 5.1 for transformation matrices between different frames.

1.2 Task 1.b. - Graphical Simulation

The forward kinematics described in the DH table were used to implement functions in code that take joint angles and plot the pose of the robot. Figure 2 below shows the forward kinematics calculated for different joint angles.

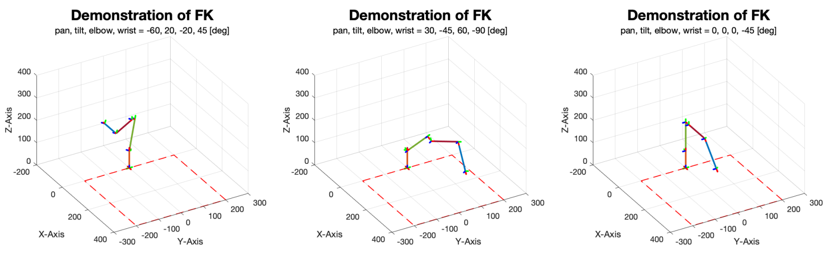


Figure 2: Diagrams demonstrating validity of the forward kinematics approach

1.3 Task 1.c. - Inverse Kinematics

The DH table and forward kinematics assume that the angles needed for the robot to reach an end location are known. However, given the final coordinates only, the joint angles can be found through inverse kinematics. The position of the end effector in 3D space is defined by the four angles, *pan*, *tilt*, *elbow*, and *wrist*. This means that for the three coordinates $\begin{bmatrix} x, y, z \end{bmatrix}^T$ we have four unknown angles. To solve this issue, we can pick a desired angle between the gripper and the base frame x axis. This imposes a constraint on the *wrist*, *elbow*, and *tilt* angles allowing us to solve the system of equations.

Figure 3 shows our modelling of the robot with annotated angles. We note that this model already has coordinate $z_0 = 77$, as it is created from the axis of the tilt joint, so all further calculations imply this z-offset.

Our analytic solution of the inverse kinematics can be found Appendix 5.3. Overall, we can summarize the joint angles of the robot as:

- $\theta_{pan} = \arctan\left(\frac{y}{x}\right)$
- $\theta_{tilt} = -90^\circ + \theta_{offset} + \theta_1 \ -90^\circ + \arctan\left(\frac{24}{128}\right) + \theta_1 = \arctan 2\left(\frac{z_{wrist}}{u_{wrist}}\right) - \arctan 2\left(\frac{k_2}{k_1}\right)$
- $\theta_{elbow} = \theta_2 + 90^\circ - \theta_{offset} = \theta_2 = \arctan 2(\frac{\sin\theta_2}{\cos\theta_2}) + 90^\circ - \theta_{offset}$
- $\theta_{wrist} = \theta_{gripper} - \theta_{tilt} - \theta_{elbow}$

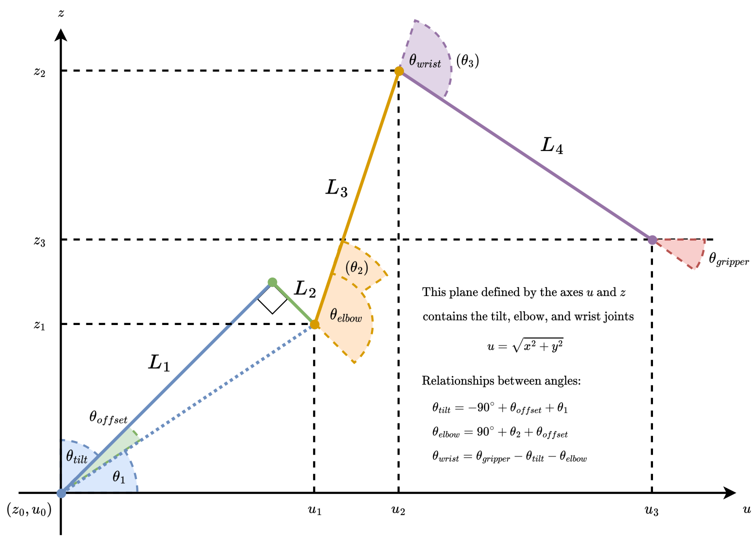
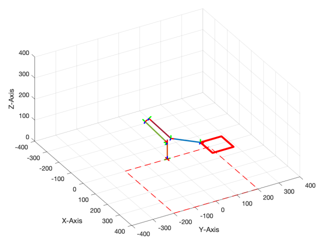

z
z_2
z_3
z_1
θ_{wrist} (θ_3)
L_4
L_3
(θ_2)
L_2
θ_{elbow}
$\theta_{gripper}$
L_1
θ_{offset}
θ_{tilt}
θ_1
(z_0, u_0)
u_1
u_2
u_3
u
This plane defined by the axes u and z
contains the tilt, elbow, and wrist joints
$u = \sqrt{x^2 + y^2}$
Relationships between angles:
$\theta_{tilt} = -90^\circ + \theta_{offset} + \theta_1$
$\theta_{elbow} = 90^\circ + \theta_2 + \theta_{offset}$
$\theta_{wrist} = \theta_{gripper} - \theta_{tilt} - \theta_{elbow}$

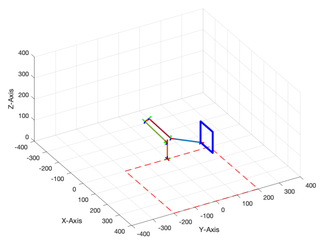
Figure 3: Diagram of our robot on any zu plane with annotated angles for IK

1.4 Task 1.d. - Tracing a square

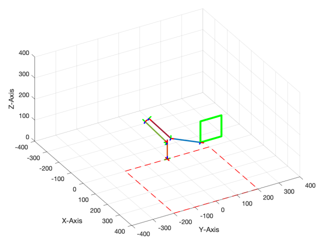
To test that our inverse kinematics calculations were correct, we simulated the robot on MATLAB tracing a square. The function tracing the square takes a starting point, the length of the square's side, the number of points along the square's side which the robot will be tracing, and two orthogonal unit vectors v_1, v_2 defining the plane in which the square exists. We define a separation of the points the robot will be tracing as an integer rounding of $N_{points}/Length$. Then, starting from the given starting point, we move a distance *separation* in the direction of v_1 to get the second point on the trajectory. We perform such a movement from our last point to our new point for a number of steps equal to the number of points on the side. Then, we do the same, but moving along the direction of v_2. Finally, the repeat the task along the direction of first $-v_1$ and then $-v_2$. Inverse kinematics are calculated at each defined point. Figures 4a, 4b and 4c shows the robot having drawn a square on one plane.



(a) Square traced in XY plane



(b) Square traced in XZ plane



(c) Square traced in YZ plane

Figure 4: The simulated robot has traced squares of side length 100mm in all coordinate planes

2 Task 2: Pick and Place

The inverse kinematics finds angles for a given position of the end effector and the in-plane elevation angle of the final link. However we would like to consider how the robot travels between start and end points. Hence, we need to be able to define a trajectory for the robot to follow. We will achieve smooth motion by ensuring the robot accelerates and decelerates as necessary.

2.1 Cartesian Cubic Interpolation along a Straight Line Path

For the pick and place task we require straight-line motion of the end effector to minimise the distance travelled by the cube. One way to achieve smooth motion along a straight line is to perform cartesian cubic interpolation. This involves generating a straight-line trajectory of target points corresponding to equally spaced instants in time but cubically spaced along the trajectory. The cubic fitting of these points is such that the minimum of the cubic lies at the start point and the maximum at the end. Hence, the velocity of the robot will start and end at zero with an intermediate maximum.

We want the distance of the points along the line to be cubically varying. We define equation $ax^3 + bx^2 + cx + d = y$ to describe this. Then, $dy/dx = 3ax^2 + 2bx + c = 0$ at the start and the end point. Following this line of reasoning, which is fully developed in Appendix 5.4, the restrictions on the coefficients are: $a = -\frac{2L}{x^3}, sb = -\frac{3aL}{2}$. Implementing these calculations in code and simulating, we obtain Figure 4.

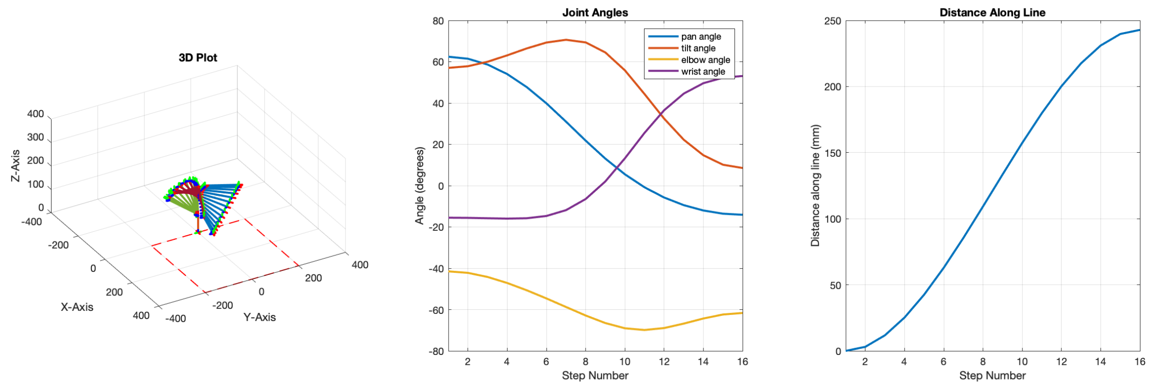


Figure 5: Diagram showing cubic interpolation on a straight line

2.2 Task 2.a. - Translation

We move cube 1 to position 4, cube 2 to position 5, and cube 3 to position 6. Our pick and place routine consisted of five movements: moving the gripper from any location to a height above the target block, moving down, closing the gripper around the block, moving the block up, translating it to a height above the finishing location, then placing it downwards and finally opening the gripper and moving up to be clear of the block. All individual trajectories were made with the Cartesian Cubic Interpolation function described previously. The pick and place routine was packaged in a function called `robot_goto_pick_place()`. The code for this task can be found in Appendix 5.4.

2.3 Task 2.b. - Re-orientation

For this task, the main focus was on the rotation of the gripper angle. To achieve the task, the gripper had to start at $-90°$ and rotate to $0°$ to place the cubes with the correct face up. This was also done with cubically interpolated upwards and downwards paths. For cube 2, it was necessary to do two rotations. Ideally, a single rotation with the gripper angle starting from $-180°$ and going to $0°$ would perform the motion with a single continuous rotation, however the lengths of the robot's links are physically limiting this action. Hence, the robot performs two rotation for the second cube.

2.4 Task 2.c. - Stacking Several Cubes

This task involved a combination of the previous two tasks. We decided to stack the cubes in position 4. We began by picking up block 1 with an angle of $-90°$, and rotating it to $0°$ while performing the cubically interpolated movement of picking and placing, as described in Task 2.a. Next, we pick up block 3 the same way, and perform the pick and place, with a final location now that has a modified height, to account for the height of the block already there. So, while the first movement involved a line of translation parallel to the xy plane, this movement contains a diagonal line. Our function is general enough to be able to handle this, as it can interpolate over any trajectory in the direction of the line joining the start and end points. Finally, we move the arm closer to its base, so as not to hit the existing stack of cubes, and move to cube 2. Once again, the same function is used, however this time we include a rotation of the gripper angle of $-90°$ to $0°$ while the cube is being translated.

3 Task 3: Trajectory Following

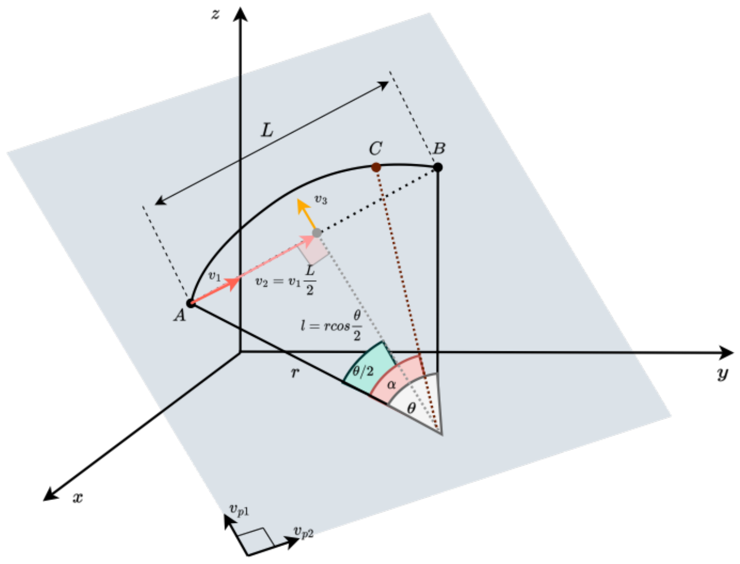


Figure 6: Diagram of arc trajectory traced with cubically spaced angles

After cubically interpolating points of a linear trajectory, we needed to cubically interpolate points on an arc. We wished to write a function to trace an arc of arbitrary start and end points, arbitrary radius (at minimum half the distance between the two points, in which case the arc would be a semicircle), in an arbitrary plane. Figure 6 illustrates this idea.

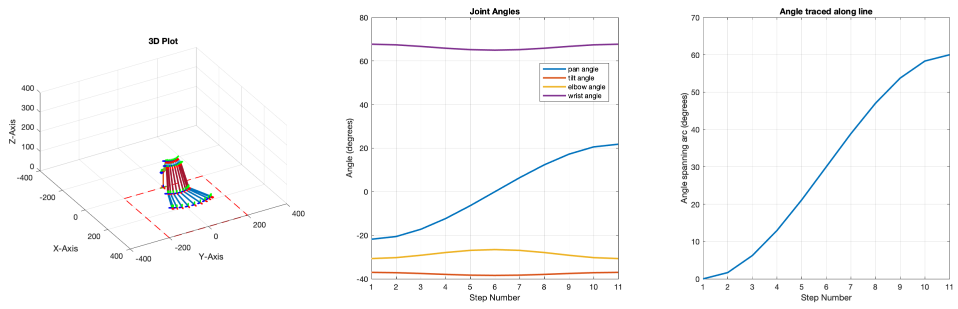


Figure 7: Diagram showing cubic interpolation on an arc path

Our function, hence, has arguments of start and end points, radius, a direction vector for the plane which the arc will be drawn in (the second vector defining the plane can be inferred from the coordinates of the two points), the number of points to interpolate, and a reflex argument, t allow us to select the arc drawn by a reflex angle. The relationship $\Delta d = r \times \Delta\theta$ (where θ is in radians) demonstrates that the relationship between the distance along an arc and the angle spanning the arc is linear, and is what we need to exploit to cubically interpolate the distance.

3.1 Task 3.a. - Holding the Pen

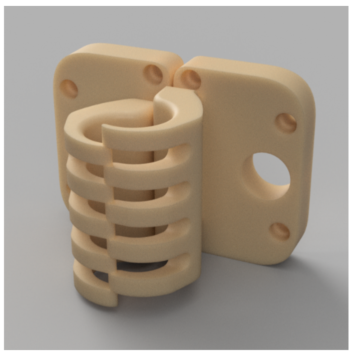
The gripper to hold the pen in task 3.b. was designed such that the 'fingers' interlink at the end of the gripper travel in order to have a caging closure. The diameter of the closed gripper aperture was designed to over-fit the pen with a designed tolerance of a few millimeters intended to leave room for a foam lining. With the addition of foam, the body of the pen in the new gripper is exactly where the forward kinematics including the old gripper puts the end effector frame meaning we do not need to include offsets in any FK calculations with this gripper.

Task 3 also required a pen holder which was designed to hold the pen vertically so no gripper angle change is required between the picking of the pen and the drawing, eliminating any offset or error that would result from picking the pen and drawing with it in different orientations.

3.2 Task 3.b. - Drawing the Pattern

To draw the pattern, we used our functions for cubic interpolation along a linear path, and cubic interpolation along an arc. We had three separate paths for the triangle, starting from point (60, 200) to point (140, 125), then to point (140, 200), and finally back to point (60, 200). Next, the arc is generated.

The function generating the circular path, once again, takes as inputs the coordinates of the start and end points, the number of points to interpolate, and the angle of the gripper while tracing that path. We decided to have the gripper draw all the pattern with its gripper at an angle of 0°. For the arc, we use the same arc trajectory generation function, and we pass it as arguments the start point and the end point, which are points A(60,200), and point C(140, 200), the radius, which is 40mm, which implies that, since it is half the separation of A and C, the arc will be semicircular, as well as the direction of the plane. Setting the direction along x would generate the arc being traced away from the robot. Hence, we instead define the direction as $[-1, 0, 0]^T$, so that the robot traces the arc towards it. The reflex argument can be set to 0.


(a) Task 3 gripper

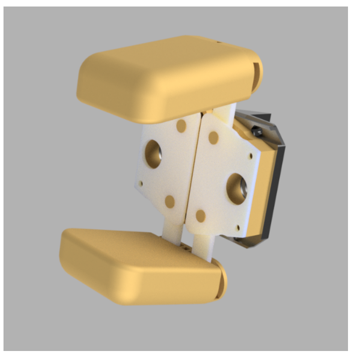

(b) Task 4 gripper

Figure 8: Gripper designs for tasks 3 and 4

4 Task 4: Self-Selected Task

4.1 Idea Selection and Motivation

For this task, we went through a lot of ideas which we would have liked to implement, from a robot that would play the cello, to a robot that would roll sushi, or play chess using computer vision, flip a vinyl, act as a prosthetic arm, or conduct an orchestra. We considered how to incorporate natural and expressive movements on the robot, which would be something different compared to the mechanical tasks it performed in the previous tasks, without loosing functionality. Hence, we arrived to the idea of a robot that would act as a hand puppet; it could be used to, for example, play theater to children at a fair or entertain a general audience.

4.2 Movements

Our aim for this task was to use movements containing, in their majority, arcs, to see the effect of our arc generation function in other planes. Such movements are also closer to human movements, making our robot more expressive. Hence, we decided upon a set of movements performed by the robot, and then applied that principle on their execution.

The general idea is that the robot performs a show. First, it needs to be trained to learn what movements it will be performing during its show. For example, it might perform an expressive monologue, or maybe do a fun dance. During the training process, the torque is turned off, and the user/trainer goes through the movements holding the robot. The learning is done using a function in which the computer continuously reads servo positions and stores them in a matrix.

Once the training is done the robot goes back to its home position. Then, it begins a sequence of predefined movements. It begins by opening a pair of curtains which are mounted on the sides of the robot's "stage". The task of opening the curtains is done using circular arcs in 3D space.

Next, it demands 'tickets' from three different positions, representing the first row of the audience and deposits them in a designated box with three separations while following an arc trajectory. While performing that movement, the robot's gripper also rotates from being $0°$ to being $-90°$. The robot also needs to be precise and mechanically able to correctly hold and release the ticket in the right slot in the box.

It continues by performing a brief dance to engage the audience, where it again moves in piece-wise arc trajectories, on a plane parallel to the defined zy plane.

Next, to perform the learned task that it was trained for, the computer writes to the servos the values read and stored in the matrix during the learning process. Finally, once it finishes its performance, the robot stands up straight, pans to look left and right at the audience, and takes a bow in an arc trajectory. The gripper now is also rotating, first from $0°$ to $-45°$.

4.3 3D Printing Gripper and Task Items

Task 4 required a gripper that could make the robot resemble a hand-puppet so mechanical links were used to transform the horizontal linear motion of the existing gripper chucks into a vertical motion. This involved a trapezoidal linkage with a centering rail to ensure the grippers move parallel to each other. Prop items such as the 'pillars' and the ticket box were designed to slot into the existing base.

5 Appendix

5.1 Transformation Matrices Calculations for Forward Kinematics

These transformation matrices correspond to the frame transformations between any two consecutive rows of the DH table, Table 1.1. The general form of the transformation matrix is:

$$T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1} d_i \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1} d_i \end{bmatrix}$$

Starting from the global frame and rotating it to align the axes such that the rest of the transformations are convenient, while simplifying for the terms that are zero:

$$T_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_1 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} \cos pan & -\sin pan & 0 & 0 \\ \sin pan & \cos pan & 0 & 0 \\ 0 & 0 & 1 & 77 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_4 = \begin{bmatrix} \cos tilt & -\sin tilt & 0 & 0 \\ \sin tilt & \cos tilt & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_5 = \begin{bmatrix} 0 & 1 & 0 & 128 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6 = \begin{bmatrix} \cos elbow & -\sin elbow & 0 & 24 \\ \sin elbow & \cos elbow & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_7 = \begin{bmatrix} \cos wrist & -\sin wrist & 0 & 124 \\ \sin wrist & \cos wrist & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_8 = \begin{bmatrix} 1 & 0 & 0 & 126 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We obtained the following vector which holds the x, y, and z coordinates of the middle of the gripper in terms of all the angles:

$$\mathrm{T}_{position} = \begin{bmatrix} 24c(p)c(t) - 128c(p)s(t) + 126c(w)[c(e)c(p)c(t) - c(p)c(e)s(t)] - 126s(w)[c(e)c(p)s(t) + c(p)c(t)s(e)] + 124c(p)[c(e)c(t) - s(e)s(t)] \\ 24c(t)s(p) - 128s(p)s(t) - 126c(w)[s(e)s(p)s(t) - c(e)c(t)s(p)] - 126s(w)[c(e)s(p)s(t) + c(t)s(e)s(p)] - 124s(p)[s(e)s(t) - c(e)c(t)] \\ 128c(t) + 24s(t) + 124[c(e)s(t) + c(t)s(e)] + 126c(w)[c(e)s(t) + c(t)s(e)] + 126s(w)[c(e)c(t) - s(t)s(e)] + 77 \end{bmatrix} \tag{2}$$

The forward kinematics calculations were implemented in MATLAB as the function `FK_calc` that returns the pose matrices for every frame used to define the DH table. The function code can be found below:

```
function [poses] = FK_calc(pan, tilt, elbow, wrist, gripper)

    % this function returns a cell array of pose matrices corresponding to
    % the pose of each of the frames defined by our DH table.
    % the inputs are all the joint angles pan, tilt, elbow, and wrist. the
    % gripper input argument is for use if another rigid link was added to
    % the gripper servo, but is currently unused here and elsewhere

    % convert all to radians
    pan = pan * pi/180;
    tilt = tilt * pi/180;
    elbow = elbow * pi/180;
    wrist = wrist * pi/180;
    gripper = gripper * pi/180; % not used atm

    % *** DH TABLE DEFINITION ***
    thetas = [pi, pan,  -pi/2,  tilt,   -pi/2,  elbow,  wrist,  0];
    alphas = [0, 0,    -pi/2,  0,      0,      0,      0,      0];
    as =     [0, 0,    0,      0,      128,    24,     124,    126];
    ds =     [0, 77,   0,      0,      0,      0,      0,      0];

    % frame positions
    pose0 = eye(4); % identity matrix
    poses = {pose0};
    dh_transforms = {pose0};

    % perform each of the transformations to the base frame to generate all
    % the frame positions.
    for i = 1:8
        dh_transforms{i} = [ cos(thetas(i)),                     -sin(thetas(i)),
             0,                     as(i) ;
                             sin(thetas(i))*cos(alphas(i)),    cos(thetas(i))*cos(alphas
    (i)),    -sin(alphas(i)),       -ds(i)*sin(alphas(i));
                             sin(thetas(i))*sin(alphas(i)),    cos(thetas(i))*sin(alphas
    (i)),    cos(alphas(i)),        ds(i)*cos(alphas(i));
                             0,                                 0,
             0,                     1 ];

        poses{i+1} =  poses{i} * dh_transforms{i};
    end
end
```

5.2 Servo Step Limits

In order to ensure the servos are not instructed to move to positions where the robot will collide with itself or over-rotate, we defined the following servo limits on each of the joints by writing the limits to each Dynamixel servo. This adds a degree of safety whilst allowing the full range of motion from resting position to just before the joint mechanical end-stops.

	Pan	**Tilt**	**Elbow**	**Wrist**	**Gripper**
Max	3073 (right)	3073 (down)	3070 (down)	3420 (down)	2550 (open)
Min	1023 (left)	700 (up)	1023 (up)	1023 (up)	1023 (closed)

5.3 Calculations for Inverse Kinematics

The analytic solution process to the inverse kinematics approach we took can be found below.

We can consider the coordinates of each joint in terms of the angles and the lengths of the links used to reach it. Specifically for *pan*, we know that it does not affect the z coordinate of the end effector. Hence, we can consider the arm first existing in a plane in the global reference frame, with a horizontal axis $u = \sqrt{x^2 + y^2}$, and the same vertical axis as global z. A projection of the arm onto the x-y plane would show that the arm is simply a straight line on that plane, while $pan = \left(\frac{y}{x}\right)$.

L_2 represents the small distance between the end of the servo generating the *elbow* angle, and the center of its joint. Given that this distance is fixed, and that there will always be a 90° angle between the link L_1 and the end of L_2, we can ignore this small offset, and instead consider the hypotenuse of the triangle formed by L_1 and L_2, $L_{hyp} = \sqrt{24^2 + 128^2}$. This also allows us to define $\theta_{offset} = \arctan\left(\frac{24}{128}\right)$.

Given we considered the "home" position of the robot to be the one shown in Figure 1 (a) for our calculations of forward kinematics, the angles we defined for inverse kinematics are with respect to that pose. We considered the robot as two subsystems, one containing links L_1, L_2, and L_3, and one containing link L_4, separated at the *wrist*. In Figure 3 we see all the angles that the robot arm creates on any plane perpendicular to the xy plane. We can see that $\theta_{tilt} = -90^\circ + \theta_1 + \theta_{offset}$. We define $\theta_2 = \theta_{tilt} - 90^\circ + \theta_{offset}$. This will help us get the coordinates of the point C. Finally, we define $\theta_{gripper} = \theta_{tilt} + \theta_{elbow} + \theta_{wrist}$, which we will relate to the position of the end effector. Given that we have three equations with four unknown angles, we can choose the angle $\theta_{gripper}$ ourselves. For example, we might desire that for a certain task that angle remains 0° or 90°. This imposes a constraint on the sum $\theta_{tilt} + \theta_{elbow} + \theta_{wrist}$. Working backwards, the coordinates of the *wrist* joint are:

$$\begin{bmatrix} u_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} u_3 \\ z_3 \end{bmatrix} - \begin{bmatrix} L_4 \cos\theta_{gripper} \\ L_4 \sin\theta_{gripper} \end{bmatrix}$$

This means that when defining $\theta_{gripper}$, we immediately know the coordinates of the *wrist* joint. This leaves us two equations for *tilt* and *elbow*, with those two angles as the unknowns. This is a simple subsystem with two links. Hence, defining the coordinates of the *wrist* joint as:

$$u_{wrist} = L_{hyp}\cos\theta_1 + L_3\cos\theta_1 + \theta_2, z_{wrist} = L_{hyp}\sin\theta_1 + L_3\sin\theta_1 + \theta_2$$

$$\Rightarrow \cos\theta_2 = \frac{u_{wrist}^2 + z_{wrist}^2 - L_{hyp}^2 - L_3^2}{2L_{hyp} \times L_3}$$

and hence $\sin\theta_2 = \pm\sqrt{1 - \cos\theta_2^2}$. From this we define: $k_1 = L_{hyp} + L_3\cos\theta_2, k_2 = L_3\sin\theta_2$. Using both of these angles in our calculation of the arctangent, to calculate the angle in the correct quadrant: $\theta_2 = \arctan 2(\frac{\sin\theta_2}{\cos\theta_2})$. Now, using the intermediate values k_1, k_2, we get:

$$\theta_1 = \arctan 2\left(\frac{z_{wrist}}{u_{wrist}}\right) - \arctan 2\left(\frac{k_2}{k_1}\right)$$

. Overall, we can summarize the joint angles of the robot as:

- $\theta_{pan} = \arctan\left(\frac{y}{x}\right)$
- $\theta_{tilt} = -90^\circ + \theta_{offset} + \theta_1 \; -90^\circ + \arctan\left(\frac{24}{128}\right) + \theta_1 = \arctan 2\left(\frac{z_{wrist}}{u_{wrist}}\right) - \arctan 2\left(\frac{k_2}{k_1}\right)$
- $\theta_{elbow} = \theta_2 + 90^\circ - \theta_{offset} = \theta_2 = \arctan 2(\frac{\sin\theta_2}{\cos\theta_2}) + 90^\circ - \theta_{offset}$
- $\theta_{wrist} = \theta_{gripper} - \theta_{tilt} - \theta_{elbow}$

The function `IK()` was implemented to perform the inverse kinematics calculation when given the cartesian coordinates for the gripper and the final link angle relative to the ground.

```matlab
function [pan, tilt, elbow, wrist] = IK(x, y, z, gripper_angle, elbow_up_down)

    % calculated all joint angles required for the end effector (point
    % between the gripper jaws) to be at specified coordinates AND for the
    % link between the elbow and gripper joints to be at a specified
    % 'gripper_angle' to the ground. 'elbow_up_down' should be set to -1 to
    % obtain valid joint angles.

    % Get pan angle
    pan = atan(y/x) * 180/pi;

    % Get the wrist coordinates
    L4 = 126;
    u_wrist = sqrt(x^2 + y^2) - L4 * cos(gripper_angle*pi/180);
    z_wrist = z - L4 * sin(gripper_angle*pi/180) - 77;

    % get length of the hypotenuse
    L3 = 124;
    Lhyp = sqrt(24^2 + 128^2);
    theta_offset = atan(24/128);

    % find inplane angles (using different angle convention)

    c2 = ( u_wrist^2 + z_wrist^2 - Lhyp^2 - L3^2 ) / ( 2 * Lhyp * L3 );
    s2 = elbow_up_down*sqrt( 1 - (c2)^2 ); % elbow up/down

    k1 = Lhyp + L3 * c2;
    k2 = L3 * s2;

    theta2 = atan2(s2, c2);
    theta1 = atan2(z_wrist,u_wrist) - atan2(k2, k1);

    % convert to our angles (i.e. pan, tilt, elbow, wrist
    tilt = ( - pi/2 + theta_offset + theta1 ) * 180/pi;
    elbow = ( theta2 + pi/2 - theta_offset ) * 180/pi;

    % find wrist angle
    wrist = gripper_angle - tilt - elbow;
end
```

5.4 Calculations for Cartesian Cubic Interpolation on a Line

We want the distance of the points along the line from the start point to the end point to be cubically varying. We define a cubic equation $ax^3+bx^2+cx+d = y$ to describe this. Then, $dy/dx = 3ax^2+2bx+c$. This quadratic must have solution $3ax^2 + 2bx + c = 0$ at the start and the end point.

The general solution of the quadratic is $(x - x_1)(x - x_2)$, and needs to be zero for $\frac{-2b\pm\sqrt{4b^2-12ac}}{6a}$, which has two solutions, $\frac{-b+\sqrt{b^2-3ac}}{3a}$, $\frac{-b-\sqrt{b^2-3ac}}{3a}$. If we consider the distance along the line being cubically interpolated, rather than the coordinates of the points themselves, then one of those solutions will be zero, corresponding to the start point, and the other will be equal to the length of the line joining the points. The solution $-b+\sqrt{b^2-3ac} = 0$ corresponds to the start point, since, rearranging: $b = \sqrt{b^2-3ac}$, yielding either $a = 0$ or $c = 0$. We discard the first solution, since it exists in the denominator of a fraction, and since setting this would mean the trajectory would not be cubic anymore. Given that we want the plot of the cubic to have no offset, we can also set $d = 0$. This means we just need to find a relation between a and b. Given $c = 0$, the positive solution becomes: $L = \frac{-2b}{3a}$. This also means that the new form of the cubic is: $y = ax^3+bx^2 \Rightarrow L = ax^3 - \frac{3x}{2}ax^2 \Rightarrow a = -\frac{2L}{x^3}, and b = -\frac{3aL}{2}$. Now we have the equation of the cubic that dictates the spacing of the points along the line. It is important to note that here, x does not represent a spatial coordinate, rather the index of the point that we are considering every time.

In code, we create a function which takes as inputs the start and end points, the number of points to interpolate, and the starting and ending gripper angle. The function symbolically calculates the parameters a and b of the cubic, evaluates the cubic for every one of the number of points required to interpolate, then adds the result of the distance calculation to the starting point, by multiplying with a unit vector in the direction of the line joining the start and end points. The function returns the angles of the robot, which can then be written to the servos. This is a function we extensively used in the implementation of most our tasks.

```matlab

function [joint_angles] = cart_cubic_IK(start_point, end_point, npoints,
    gripper_angle_end, gripper_angle_start)

    % given a start point and end point this function cubically
    % interpolates npoints + 1 along the straight line between the two and
    % returns 'joint angles' - a 4*(npoints+1) where each column
    % corresponds to the pan, tilt, wrist, and elbow joint angles required
    % to reach that point along the trajectory.
    %
    % the gripper angle may also have a start and finish value which is
    % linearly interpolated from 'gripper_angle_start' to
    % 'gripper_angle_end' during the movement.
    %
    % the gripper angle is also an input which is  linearly interpolated
    % when solving inverse kinematics along the line

    if (~isequal(start_point, end_point)) % only interpolate if start!=end

        % unit vector along linear trajetory
        v = (end_point - start_point) / norm(start_point - end_point);

        % distance from start to end (mm)
        d = norm(start_point - end_point);

        % find the cubic that fits the start and end points as we desire
        % i.e. min and max of the cubic lie at start_point and end_point
        a = -2 * d / ( npoints ^ 3 );
        b = -3/2 * npoints * a;

        syms y x;
        y = a*x^3 + b*x^2; % define the cubic symbolically

        x = 0:npoints;

        cartesian_points_for_ik = [];

        for i=1:length(x)
            dd = eval(subs(y, x(i))); % distance along the line, delta d
            cartesian_points_for_ik = [cartesian_points_for_ik, start_point + v * dd];
        end

    else % if start=end, this is just a rotation; don't cubically interpolate

        cartesian_points_for_ik = start_point .* ones(3,npoints+1);
        disp('Note: only theta_gripper rotation can occur in this movement');

    end

    joint_angles = [];

    % vary gripper_angle linearly for each defined point - this is smooth
    % enough as the wrist servo does not have a large intertial burden
    gripper_angles = linspace(gripper_angle_start, gripper_angle_end, length(
    cartesian_points_for_ik));

    % run IK for each point
    for i=1:length(cartesian_points_for_ik)
        [joint_angles(1,i), joint_angles(2,i), joint_angles(3,i), joint_angles(4,i)] =
    ...
            IK(cartesian_points_for_ik(1,i), cartesian_points_for_ik(2,i),
    cartesian_points_for_ik(3,i), gripper_angles(i), -1);
    end

end
```

5.5 Calculations for Cubic Interpolation on Arc

To find and increment this angle, we need to find the center point of the circle. For this, we first define a unit vector between the start and points, $\mathbf{v_1}$, and we find the distance L between them. Next, we define the midpoint of the line joining them. We can find the angle that the arc will be spanning by using the cosine rule for the triangle created by the two radii connecting the start and end points to the center of the circle (which still has unknown coordinates), and the length of the line joining the points: $L^2 = 2r^2 - 2r^2\cos\theta \Rightarrow \theta = \cos^{-1}\left(1 - \frac{L^2}{2r^2}\right)$. Taking the cross product between $\mathbf{v_1}$ and the vector defining the plane, $\mathbf{v_{p1}}$, we obtain a perpendicular vector to the plane of the arc. Crossing that vector with $\mathbf{v_1}$ now gives the vector perpendicular to the midpoint, $\mathbf{n}$ of the line joining start and end, $\mathbf{v_3}$. The centre of the circle is a length l away from the midpoint, taken in the direction of $\mathbf{v_3}$, where: $l = r\cos\theta/2$, since it creates a right-angled triangle with $\mathbf{v_2}$, and the radius. Hence, the coordinates of the centre of the circle are given by:

$$\begin{bmatrix} x_{centre} \\ y_{centre} \\ z_{centre} \end{bmatrix} = \begin{bmatrix} x_{L/2} \\ y_{L/2} \\ z_{L/2} \end{bmatrix} + \mathbf{v_2} \times l$$

Now we need to find the coordinates of the cubically spaced points along the arc trajectory. We define unit vector $\mathbf{v_4}$ in the direction of the line connecting the center point and the starting point. That vector will be get rotated by an arbitrary angle α, around the perpendicular vector to the plane, $\mathbf{n}$, placed on the center point. We define an intermediate matrix to aid this calculation, $\mathbf{C}$, which we will use to calculate the rotation matrix by angle θ, $\mathbf{R}_\alpha(\theta)$:

$$\mathbf{C} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \Rightarrow \mathbf{R}(\alpha) = \mathbf{I} + \mathbf{C}\sin\alpha + \mathbf{C}^2(1-\cos\alpha)$$

Now, we can define a new unit vector in the direction of the point at the trajectory C generated by the arbitrary angle α:

$$\mathbf{v_5} = \frac{\mathbf{R}(\alpha) \times \mathbf{v_4}}{|\mathbf{R}(\alpha) \times \mathbf{v_4}|}$$

Finally, the points in the trajectory are generated by taking the coordinates of the center point and adding a distance given by $\mathbf{v_5} \times r$. We note that the value of the angle step every time is calculated using the cubic interpolation function described in Task 2, by converting the distance along the arc into an angle in radians, which then is converted to degrees, and finally into servo steps.

```

function [arc_points] = generate_arc_trajectory(start_point, end_point, radius,
    plane_direction, reflex, npoints)
    % given a start point, end point, radius, and directional vector this
    % function finds npoints+1 points along the circular arc lying in a
    % place containing all given points and defined by 'plane_direction'.
    %
    % points are cubically spaced along the distance of the arc! setting
    % the 'reflex' argument to 1 choses the reflex arc to be drawn instead
    % of the smaller arc. the points are cartesian coordinates.

    % normalise plane direction vector in case it isn't normalised by bad people
    plane_direction = plane_direction / norm(plane_direction);

    % find theta (arc angle)
    L = norm(start_point - end_point);
    theta = acos(1 - (L^2 / (2*radius^2)) );

    if (reflex)
        theta = pi*2 - theta;
    end

    % find length of the perpendicular bisector of the line from the start
    % point to the end point
    l = (radius * cos(theta / 2));

    % v1 is the unit vector between start and end
    v1 = (end_point - start_point) / norm(end_point - start_point);

    % obtain plane normal vector
    plane_normal = cross(plane_direction, v1);
    plane_normal = plane_normal / norm (plane_normal);

    % obtain other plane direction vector (not used anywhere)
    plane_direction_two = cross(plane_direction, plane_normal);
    plane_direction_two = plane_direction_two / norm (plane_direction_two);

    % v2 is vector to midpoint from start point
    v2 = v1 * L / 2;

    % find v3 - it is the normal to v1 AND the plane_normal vector
    v3 = cross(v1, plane_normal);
    v3 = v3/norm(v3); % normalised

    % find circle centre point
    centre_point = start_point + v2 + (-v3) * l;

    % v4 is a normal vector pointing from the centre point to the start point
    v4 = (start_point - centre_point) / norm((start_point - centre_point));

    % alphas = linspace(0,theta,npoints); % linear interpolation OLD

    % cubically interpolate angle 'alpha' from 0 to theta
    % this is equivalent to cubically interpolating distance along the arc
    % because d = r
    d = theta;
    a = -2 * d / ( npoints ^ 3 );
    b = -3/2 * npoints * a;
    syms y x;
    y = a*x^3 + b*x^2;
    x = 0:npoints;
    for i=1:length(x)
        alphas(i) = eval(subs(y, x(i)));
    end

    % find the cartesian coordinates of the end effector for each value of alpha
    for i=1:length(alphas)

        % define a rotation of the vector from the centre point to end
        % effector in matrix R
        C = [ 0, - plane_normal(3), plane_normal(2) ;
        plane_normal(3), 0, - plane_normal(1) ;
        -plane_normal(2), plane_normal(1), 0];
        gamma = alphas(i);
        R = eye(3) + C*sin(gamma) + C^2 * (1 - cos(gamma));

        % v5 is the unit vector pointing from the centre point to the
        % desired point along the trajectory
        v5 = R * v4;
        v5 = v5 / norm(v5);

        % get the cartesian coordinate of the desired point
        arc_points(:,i) = centre_point + v5 * radius;

    end
end
```

5.6 3D Printed Parts for Task 4

The grippers and props used in tasks 3.b. and 4 were designed in SOLIDWORKS and Fusion 360 respectively and the parts were 3D printed on FDM printers in PLA and PLA+.

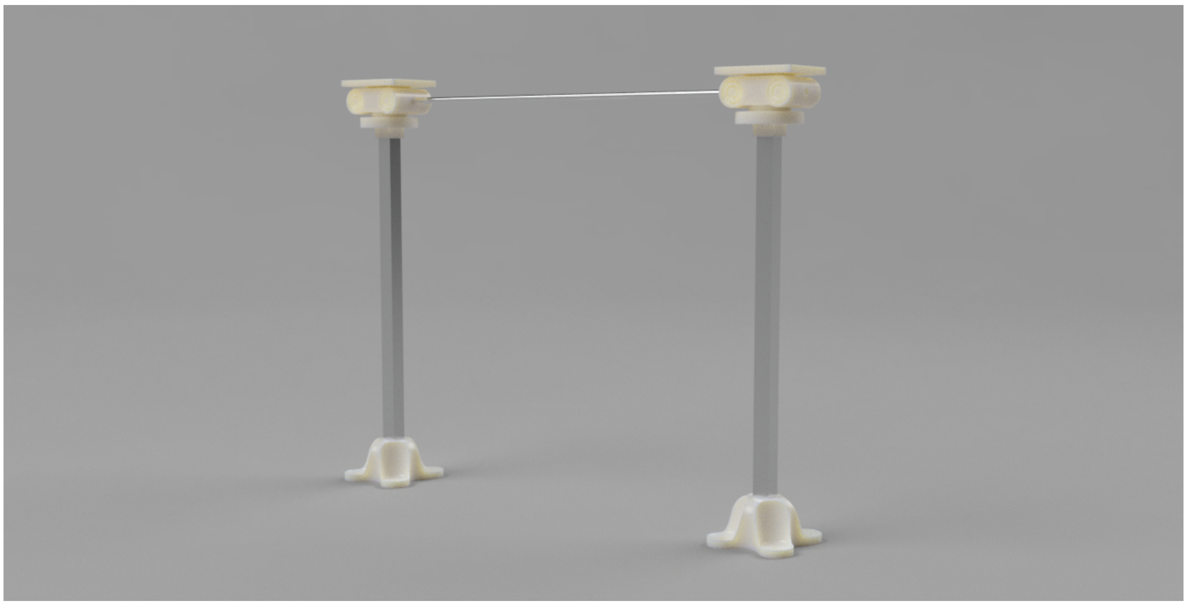
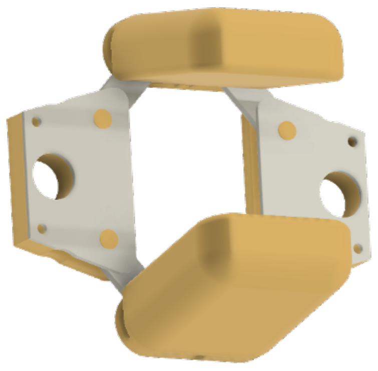
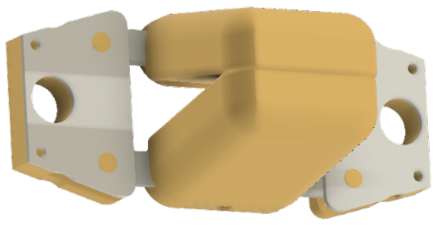


Figure 9: Curtain assembly for task 4, with 3D printed pillar holders and bases


(a) Task 4 gripper vertically open position


(b) Task 4 gripper vertically closed position