Imperial College London

ELEC60030: Robotics Manipulation

Tenaci Coursework Report

Tenaci Members:

Nicholas Pfaff	Jianing Li	Shifan Chu
01709264	01704428	01700000
nep19@ic.ac.uk	jl1719@ic.ac.uk	sc209@ic.ac.uk

Department of Electrical and Electronic Engineering
Imperial College London
London, SW7 2AZ

Table of Contents

Section 1: Modelling the Robot (Task 1)

Assigning the co-ordinate frames and creating the DH table

The robotic manipulator used in this coursework is the OpenMANIPULAOR-X, manufactured by Robotis. Figure 1 illustrate the robot arm configuration along with necessary link dimensions. The robot arm has 4 DOF and 1 DOF for the gripper.

The modeling of the robot's arm was divided into three steps. Firstly, coordinate frames were assigned, using the D-H convention as guidance. Secondly, the D-H parameters were determined according to the frame assignment. Thirdly, homogenous transformation matrices were obtained using the D-H parameters and multiplies to obtain the gripper pose. The position vector of the end-effector (gripper) could then be obtained in the last column of the combined transformation matrix. The orientation of the end-effector (angle between the horizontal plane) could be obtained by summing the joint angles θ_2, θ_3 and θ_4. The DH-table was set up as shown in Table 1, where transformations were separated into seven distinctive steps. Each step corresponds to one of the numbered frames in Figure 1. More frames than necessary were chosen as this simplified splitting the kinematics into two subsystems for determining the inverse kinematics (see appendix).

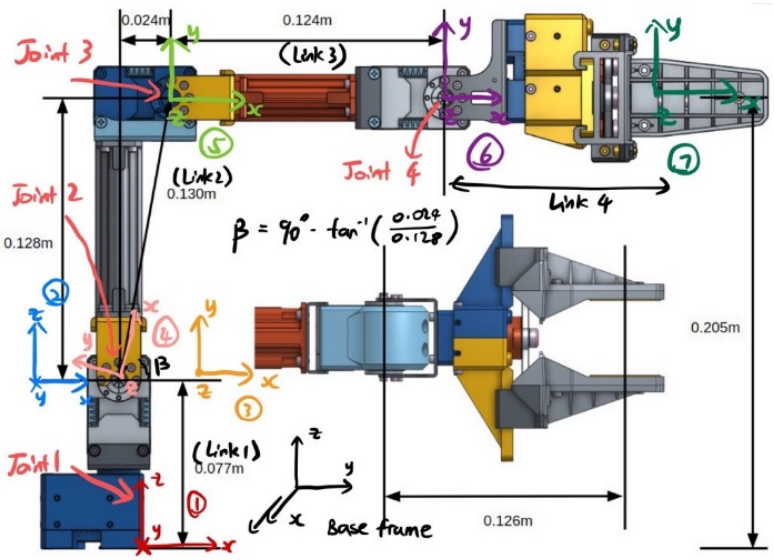


Figure 1: Robot arm configuration

Transformation step i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	$\theta_1 + \pi/2$
2	0	0	0.077	0
3	$\pi/2$	0	0	0
4	0	0	0	$\theta_2 + \beta$
5	0	0.13	0	$\theta_3 - \beta$
6	0	0.124	0	θ_4
7	0	0.126	0	0

Table 1: D-H table (where $\beta = \frac{\pi}{2} - atan\left(\frac{0.024}{0.128}\right)$)

All joint angles (θ_i) are defined as increasing when rotating anticlockwise around the Z-axis.

The world frame is set such that when all the Dynamixel joint angles were set to zero, the OpenManipulator arm is pointing toward the positive Y-axis. The X-axis is positive to the right of the arm and the Z-axis is positive upwards. This world frame made it easy to determine the cartesian coordinates required for completing the tasks.

Before assigning the rest of the frames, it is noteworthy that the rotation of Joint 1 only affects the position of the end-effector on the horizontal (X-Y) plane, and the rotation angles of Joint 2, Joint 3, and Joint 4 only affect the position of the gripper on the vertical plane. Hence, it is beneficial to split the system into two subsystems for computing the inverse kinematics. In the D-H table, the steps 1-3 represent subsystem one and steps 4-7 represent subsystem two.

In transformation step 1, the frame is rotated anticlockwise by 90 degrees around the Z-axis such that the positive Y-axis points into the plane. This is done because the rotation later will make Z point out of the plane which make it easy to model the Joint 2, 3 and

4 rotations. In step 2, the frame was lifted by 0.077m, this value is assigned to d_i since the transformation is along the Z-axis and represent the distance between 2 links. The third step is the most important step since this involves the transformation from subsystem one to subsystem two. The angle between the Z-axis before and after the rotation became 90 degrees. After this step, the Z-axis is pointing out of the frame and all the latter joint angle rotations (Joint 2, 3 and 4) can be measured as θ about the Z-axis.

Transformation step 4 represent the rotation of Joint 2, in this step, an offset angle β is also introduced, which makes it easier to translate along Link 2 (0.13m) in later steps. After the transformation, the X-axis aligns with Link 2 and θ_2 represent the Joint 2 rotation. Step 5 preforms a similar transformation; the frame is translated along the X-axis by 0.13m and the offset angle β is introduced to align the X-axis with Link 3. θ_3 represent the Joint 3 rotation.

Transformation step 6 and 7 were easy to obtain, using similar logic, since they only involve translation along Link 4 and the end-effector as well as rotation around Joint 4, where no link or angle offsets had to be introduced. The complete D-H table, describing these transformations, could then be obtained, as shown in Table 1.

Determining the forward kinematics

Computing the forward kinematics involves the calculation of the transformation matrix from the world frame to the tool (gripper) frame. After the parameters in the D-H table were obtained, these parameters were then inserted into the D-H transformation matrix and seven distinct matrices were obtained. Then, the seven matrices were multiplied together in sequence and a final transformation was obtained in the below format (homogenous transformation matrix):

$$T_0^N = \begin{Bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

The $p(p_x, p_y, p_z)$ in the matrix represent the coordinates of the end-effector in the world frame, while the other three vectors n, o and a are the orthogonal unit vectors that represent the orientation of the end-effector in the world frame. This transformation matrix represents a mapping from the four joint angles to the tool (gripper) pose.

In addition to the final end-effector, the poses of the other joints can also be obtained by matrix multiplications. For example, to obtain the coordinate of Joint 3, the transformation matrices up to T_6 would be multiplied.

Determine the inverse kinematics

Inverse kinematics is a process to determine the individual joint angles when the pose of the end-effector is given. In calculating the inverse kinematics of the OpenMANIPULATOR-X robot arm, the overall system is broken down into two subsystems.

The reason for splitting the system into subsystem is that different joint angles of the robot arm contribute to movement on different orientations. The magnitude and direction of the angle at Joint 1 (θ_1) only affects the movement of the robot arm on the horizontal X-Y plane, hence Joint 1 and link 1 are grouped together to form subsystem one. While the angles at joints 2, 3, and 4 contribute to the arm position on the vertical plane, hence the rest of the parts on the robot arm form subsystem two. The details of the inverse kinematics calculations can be found in the appendix.

Note that the inverse kinematics may return more than one solution due to redundancy. Also, some or all of these solutions might be invalid due to joint angle limit violations. Hence, all solutions are validated, and the first valid solution is chosen when wanting to obtain one inverse kinematic solution, given a cartesian gripper pose.

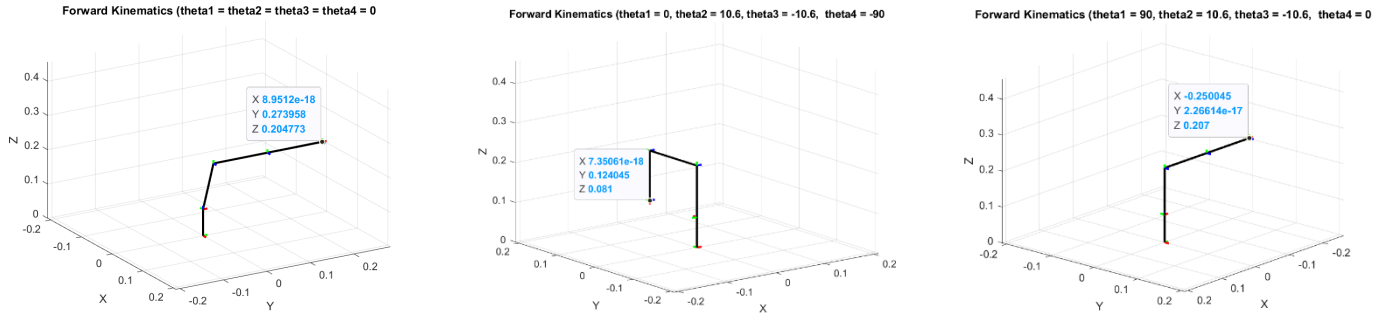
Graphical simulation for FK and IK

Forward kinematics

The graphical simulation for the forward kinematics was constructed by computing the transformation matrix from the world frame to different joint frames. By multiplying individual D-H transformation matrices, the transformation from the world frame to the joint frame, represented by the last transformation matrix, could be obtained. The coordinates for each joint could then be obtained by extracting the last column of the matrix representing the transformation from the world frame to that joint.

In graphical simulation, a link was illustrated by connecting the coordinates for each joint with a line, using MATLAB's "plot3()" function. To plot the frame for each joint, the first column in the transformation matrix from the world frame to that joint frame

was extracted to form the unit vector in the X-direction. Similarly, the unit vectors in the Y and Z directions were extracted from the 2nd and 3rd column respectively. Then, these vectors were scaled to a suitable length and plotted on the joint's coordinate using MATLAB's "plot3()" function. The result is a full graphical simulation of the forward kinematics.



Inverse kinematics (drawing a square in each cartesian plane)

The IK simulation uses a similar process: For a given pose of the end-effector (x, y, z, theta), the IK method explained in previous sections is applied to obtain the corresponding joint angles. Of all the possible joint angle combinations sets, we pick the first set that does not violate any joint angle limits. Then, the joints and links of the gripper are plotted using the FK graphical simulation.
To plot a square, the four corner coordinates of the square are provided. Between each of these coordinates, a sequence of intermediate waypoints is generated using cubic task space interpolation (see Section 2: System infrastructure). Each of these intermediate waypoints is converted into joint angles using the inverse kinematics (automatically done as part of the trajectory planning). The angles are then processed by the forward kinematics matrix multiplication to generate back the individual joint poses. The animation is achieved by plotting the new frames and deleting the old frames continuously in a loop. The final squares are shown in Figure 2, where each dot represents an individual waypoint.

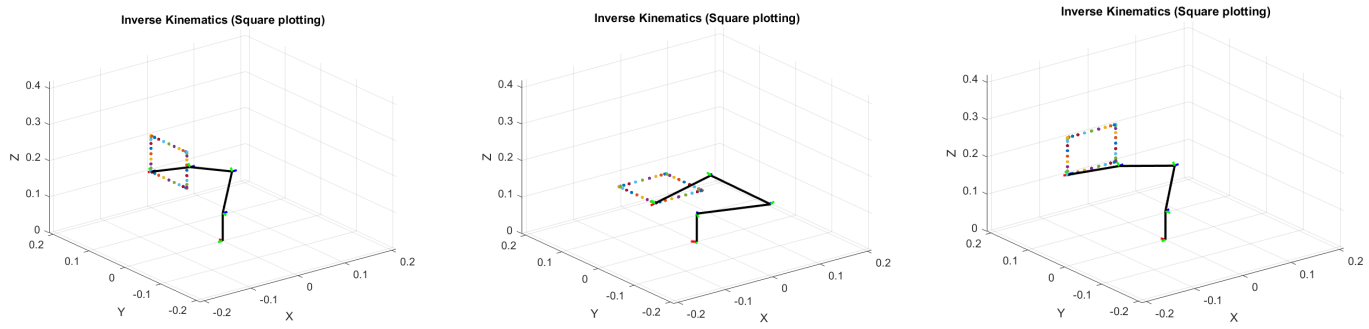


Figure 2: Left = Y-Z plane, middle = X-Y plane, right = X-Z plane

Section 2: System infrastructure

System entities

The system consists of three main entities: waypoints, set-points, and stages. Waypoints are the most fundamental entities that are produced by the task-specific algorithms. A waypoint specifies the tool's pose (x, y, z, theta) in the world frame. Moreover, it specifies whether this waypoint should form a continues trajectory with previous waypoints ("groupToPrevious"), the time that the gripper should take for this trajectory ("timeForTrajectory"), and the gripper opening after this trajectory. Cubic interpolation is used to convert waypoints into set-points. Single cubic interpolation is used if a

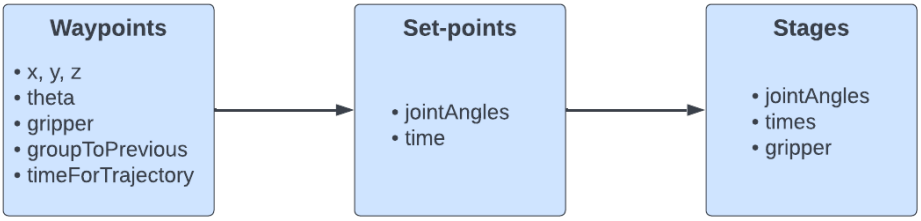


Figure 3: System entities

waypoint's "groupToPrevious" attribute is false and cubic spline interpolation otherwise. A set-point represents low-level information of a point in a particular trajectory. This includes the joint angles, for the four servos, and the time in the trajectory at which the gripper should reach these angles. A stage wraps a sequence of set-points forming a trajectory, to include additional information such as the gripper opening after this trajectory. During execution, stages are looked at iteratively. A stage's trajectory is sent to the servos and the gripper opening is sent once this trajectory has been completed. Stages are the only entities that the control script looks at while the robot is moving. Both waypoints and set-points only exist before any motion starts.

Trajectory planning

To avoid unexpected end effector trajectories, we want all joints to start moving and stop moving at the same time. Moreover, to avoid rigid motion, we want the joint velocity to peak in the middle of the movement rather than being constant throughout. We can achieve these using cubic interpolation, by ensuring that velocity starts and ends at zero.

Single cubic vs cubic spline interpolation

A single cubic interpolation allows us to plan a direct trajectory between two waypoints. This is often desirable when for example we have one waypoint above a cube and another at the position where we want to grab the cube. We want to stop at both waypoints to increase the grab's accuracy. However, in other scenarios it is desirable to move through multiple waypoints without slowing down or stopping at each waypoint. An example would be multiple waypoints between two cube locations that form an obstacle avoidance path. The waypoint's "groupToPrevious" attribute allows us to group a sequence of waypoints into a single spline trajectory (see appendix).

Joint-space vs task-space interpolation

We decided to default to task-space interpolation. The main reason is that it makes avoiding obstacles, such as the floor, easier. Moreover, it does not add significant computational overhead compared to joint-space interpolation due to our analytical inverse kinematics.
However, there are certain scenarios when a task-space trajectory is not realizable due to joint limit violations. In these scenarios, we use joint-space interpolation for that trajectory as joint-space trajectories are always realizable if the end points are realizable. Testing revealed that this does not usually occur in object-avoidance critical situations, such as when rotating a cube close to the floor. Consequently, we do not need to perform additional checks for such joint-space trajectories.

Time-based control and sending commands to the servos

Trajectories are sent to the servos using a combination of position control mode and a time-based profile. Robotis Dynamixel's time-based profile allows one to specify both the time span and the acceleration time of a trapezoidal velocity profile (Robotis, n.d.). The time span is set to the sampling period, used to sample the trajectories, and acceleration time to a quarter of that sampling period. New position demands are sent to the servos after half the sampling period, to avoid deacceleration between set-points. Half the sampling period was chosen as this allows velocity override (maintaining velocity continuity while updating position demand) to occur when velocity is at its maximum. This is necessary as sending trajectories to the servos in time-based mode represents a custom control loop on top of the individual servo control-loops.

Section 3: Pick and Place the Wooden Blocks (Task 2)

LEGO idea behind pick and place algorithms

A LEGO-style algorithm was developed for converting an arbitrary set of start and finish cube locations into a sequence of stages. This works by having one high-level function for each subtask of task 2, which specifies a sequence of actions required for achieving that subtask, using the given cube locations. An example of an action is picking up a cube at a given position with the gripper facing down and placing it back down at another position with the gripper facing straight. Each of these actions is then converted into a

sequence of waypoints using an action's corresponding low-level function.

Cube and cube-stand collision avoidance

Cube or cube-stand collisions can occur when low-height trajectories go through cube locations. This can occur with the waypoints produced by the LEGO-style algorithm. The individual action building blocks do not have information about their surrounding actions and hence the environment's state. These collisions are avoided by parsing the waypoint sequence and inserting cube avoidance waypoints if two consecutive waypoints are too far from each other (x-y L2-norm). This is based on the idea that a cube location can only be in between two waypoints if the distance between them is big enough. For example, we do not want to insert additional waypoints for movement along the z-direction.
It was found that a reliable way of producing a cube avoidance waypoint is to find the waypoint equidistant from the two waypoints. This waypoint is then modified in the joint-space by increasing joint angle 2 (counting from the base). This causes the main robot arm to bend backwards, leading to the gripper moving upwards. This modified version is then converted back into the task-space and grouped into a spline trajectory with the two surrounding waypoints. The resulting motion looks like an upward bounce.

Task 2 a

Task 2 a requires simple movement of cubes from start to finish locations. The default action for this is moving a cube between a start and a finish location using a gripper orientation of -90 degrees. If infeasible (joint limit violations), this action is replaced with a similar one where the gripper orientation is zero degrees.
The initial approach involved approaching cubes from the top. However, this does not work for some cubes outside of the dexterous workspace. Hence, a check was added that replaces from-the-top approaches with from-the-side approaches if they are infeasible. Approaching cubes from the side is achieved by computing an offset using Pythagoras.
Some cube locations cannot be reached with gripper orientations of zero or -90 degrees. To increase the robot's range, multiple orientations close to the desired one are tried until one is feasible. This is achieved by slowly decrementing orientations of zero degrees or slowly incrementing orientations of -90 degrees.

Task 2 b

Task 2 b requires rotating cubes so that the cubes' red faces end up on the top. The actions used to achieve this involve picking up the cube with the gripper facing one way (0 or -90 degrees) and placing it down again with the gripper facing another way (-90 or 0 degrees). Depending on the amount of rotation, it might be necessary to chain multiple such actions for rotating one cube. The starting gripper orientation is chosen based on the original cube orientation, to minimize the number of required rotations.
Picking up a cube with one gripper orientation and placing it back down with another one requires an offset. This is the case because the frame positioned inside the gripper's fingers cannot reach the cube's center, due to the cube stand walls. A vertical offset when grabbing the cube requires a corresponding sideways offset when placing it back down and via versa. This offset is obtained using Pythagoras.
Rotating a cube by changing the gripper orientation by 90 degrees can lead to floor collisions. Floor collisions are easier to avoid with task-space than with joint-space interpolation. Task-space trajectories between two waypoints above the floor will not go through the floor. However, gripper rotations close to the floor might still lead to collisions when using task-space trajectories. This is the case because only the frame between the gripper's fingers is considered during interpolation. The other parts of the gripper might still collide with the floor. It was found that a reliable solution to this is to do the rotation during a trajectory at increased height. This is achieved by creating a spline trajectory with an additional bounce waypoint which is computed similarly to a cube collision avoidance waypoint.

Task 2 c

Task 2 c requires rotating and stacking cubes so that all stacked cubes have their red face facing away from the robot. This is achieved using the same actions used in task 2a and 2b. Cubes that start with the correct orientation can use the actions from task 2 a while cubes that require rotation can use the actions from task 2 b. The finish location's z-coordinate is increased incrementally as cubes

are stacked. This is done by the high-level task 2 c function before calling the actions for stacking the next cube.

Section 4: Trajectory Following (Task 3)

Designing a method for securely holding the pen

A horizontal cubic holder was designed to allow the gripper to grab the pen. To fit the pen insider the holder, a cylindrical cut extrude feature was added with tilted angle to fit the pen with increasing radius toward the end. A picture of the CAD design is shown in the appendix section Task 4 tool adaptor CAD drawings. It was decided to design a simple pen holder "rectangle" rather than a more complicated gripper as this did not compromise accuracy due to the default gripper's high-friction rubber. It was not allowed to modify 3D printed grippers with similar high-friction material. Moreover, simpler is better when accuracy is not compromised due to the increased speed and decreased cost of manufacturing. Simpler is also easier to understand and less likely to break.

Drawing straight lines

A line is defined using a start and an end waypoint. The waypoints' z-coordinates and orientations are fixed to drawing specific ones that are based on how the gripper holds the pen holder while drawing. These are then connected using direct cubic interpolation (setting "groupToPrevious" to false).

Drawing arcs

An arc is defined using a start point, a center point, an angle, and a direction (clockwise or anticlockwise). Such an arc is converted into waypoints along this arc using Pythagoras. This is achieved by incrementally increasing the current angle until the current angle is equal to the specified angle. Each current angle is used to obtain a 2D point on the arc by adding the side lengths of a right-angled triangle to the arc's center point. Noting that the long side of the triangle is equal to the radius which is the distance from the start to the center point. The waypoints' z-coordinate and orientation are fixed as for drawing straight lines.
This results in an arc being defined by many waypoints. To achieve fluid motion with smooth velocity, these waypoints are connected using cubic spline interpolation (setting "groupToPrevious" to true).

Accuracy-time tradeoff

Initial accuracy improvements were achieved by decreasing the trajectory sampling period from 0.1s to 0.05s. Accuracy could be further improved by increasing all trajectory times. However, this revealed a tradeoff as both accuracy and speed are valued equally. Hence, it was decided to run a couple of experiments with different trajectory times and select the settings that produced a decently accurate drawing in the shortest time. The left picture in Figure 4 shows the drawing completed in 7s while the right picture shows it completed in 27s. The 27s version shows almost perfect accuracy. After the experiments, it was decided to use settings that produced a drawing in 11s as can be observed in the video.

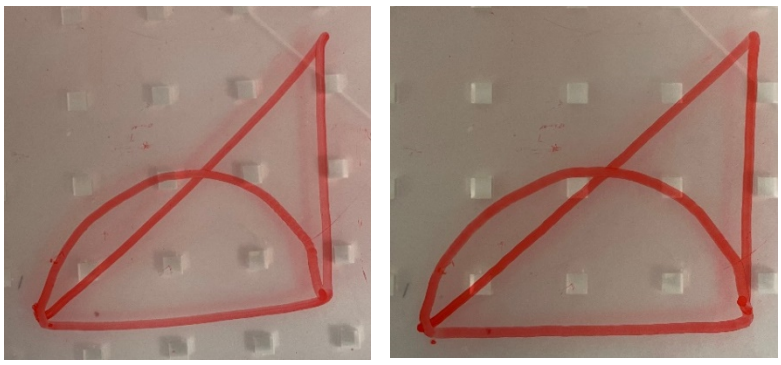


Figure 4: Accuracy-time tradeoff

Section 5: Own Task – Making Breakfast (Task 4)

Motivation and task description

Studies show that one in four adults do not have time to make breakfast in the morning (Melore, 2021). People are unwilling to wake up early to make breakfast before leaving for work. At the same time, many consider breakfast to be the most important meal

of the day, which is supported by numerous studies (Spence, 2017).

What about allowing people to sleep in while waking up to a freshly made breakfast? Startups like Moley Robotics are working on fully autonomous robotic kitchens which can solve the breakfast problem (Moley Robotics, 2022). However, the products produced by these companies are insanely expensive and unaffordable for most households. Therefore, it was decided to build a relatively low-cost solution to the breakfast problem using an OpenMANIPULATOR-X robotics arm which can be placed in a designated kitchen area.

The designed proof-of-concept implementation can serve coffee, tea, and toast. This should be sufficient for most quick breakfasts. Moreover, a positive good morning message such as "ENJOY!" can brighten one's day. Future extension ideas would include capabilities of preparing and serving healthier breakfasts.

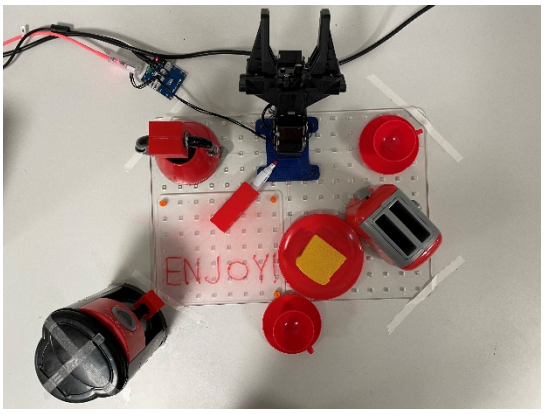


Figure 5: Task 4 layout

Uniform adaptors for tools

The 3D parts designed for picking up the cookware adopts a method which reduce complexity while increasing efficiency. It was decided not to change the gripper design so that the robot arm could use the same gripper to grasp the tea pot, coffee jug, and the pen for the "ENJOY!" message. Hence, specific adaptors for each cookware (tool) were designed to allow the gripper to use different tools. The robot arm demonstrates high reliability in grabbing cubic objects and since both the coffee jug and the tea pot have handles which allow the adaptor to fit on, cubic adaptors were designed for both tools with cut extrude features determined by the shape of the handles. The pictures for the 3D parts are shown in Figure 6. Each adaptor was split in halve during printing and then glued together to achieve a tight fit around the tool handles.

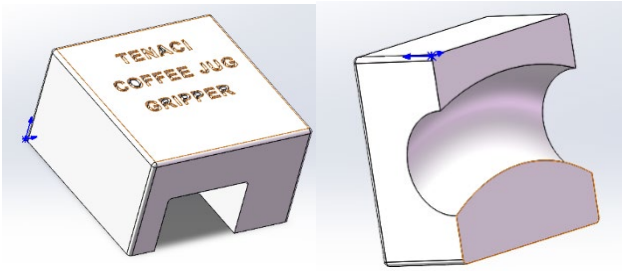


Figure 6: Uniform tool adaptors

Ensuring repeatability

The robot does not contain any exteroceptive sensors. Hence, the environment's state must be perfectly known which requires all items to be at exact locations. Some static items such as the toaster could be positioned on the plastic board by gluing old cube stands to their base. However, this would limit the movement of some items such as the tea pot. These items' boundaries were aligned with a specified set of the small holes in the plastic board.

Waypoint generation

Preparing breakfast

The waypoints for preparing breakfast were created in two steps. Firstly, rough waypoints were generated by moving the gripper by hand with its torque disabled and recording waypoints using forward kinematics. Secondly, the rough waypoints were finetuned by trial and error. If needed, more waypoints were added using step one.

The final waypoint sequence was then grouped into multiple cubic splines where consecutive splines were separated by gripper opening changes. Finally, the trajectory times were tuned by trial and error to achieve smoothness while maintaining good speed.

Writing message

Waypoints for the "ENJOY!" message were created by writing the message on the drawing board, splitting it into lines and arcs, and determining relative coordinates for the lines and arcs. The relative coordinates were then translated onto the desired part of the drawing board. The "J" can be split into one line and one half-circle, the "O" can be split into a full circle, and all other letters can be fully specified using lines. These lines and arcs were then drawn at the determined coordinates as described for task 3.

Appendix

Inverse Kinematics

The inverse kinematics can be determined by splitting the system into two subsystems. The first subsystem includes the first joint (counting from the robot base) while the second subsystem contains the remaining three joints (see Figure 7). The input to our inverse kinematic algorithm is the gripper pose in the form (x, y, z, ∅). The output is four unique joint angle combinations that correspond to the input gripper pose. Noting that only a subset of these four solutions are valid solutions due to the robot's joint limits.

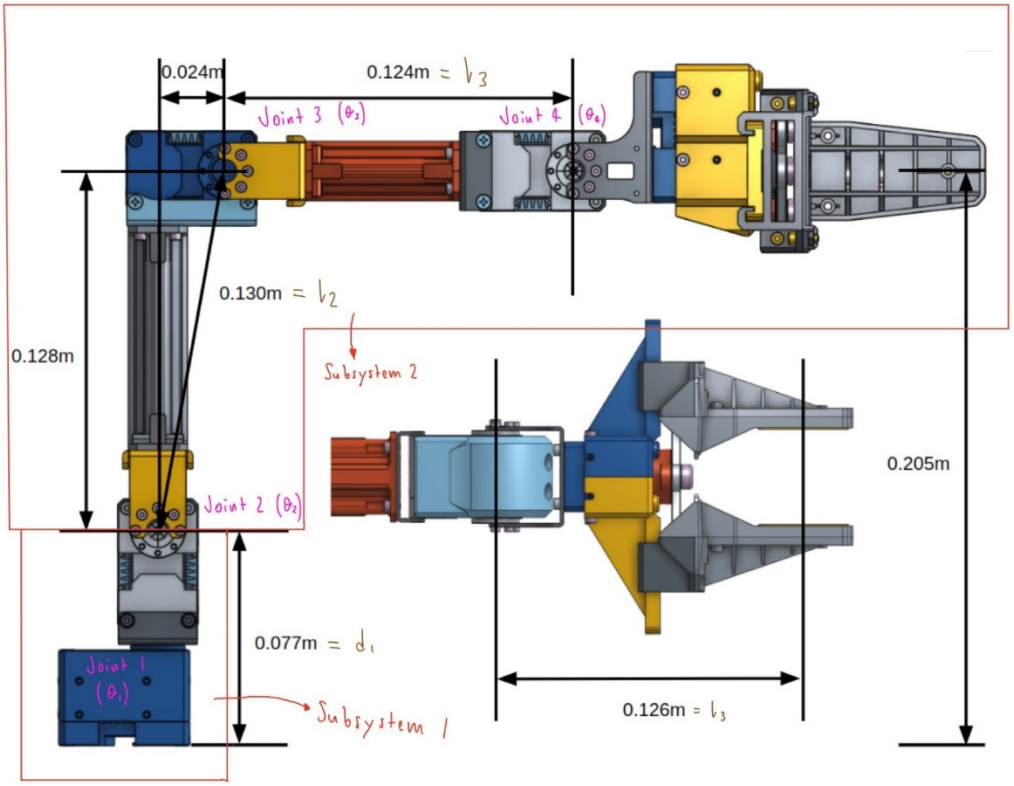


Figure 7: Subsystems

Subsystem 1

Subsystem one solely determines the position on the $x - y$ plane. Consequently, joint angle one can be determined using

$$\theta_1 = atan2(y, x)$$

Where x and y are gripper pose coordinates. $Atan2$ is used to ensure that the joint angle is between -180 and 180 degrees.

Subsystem 2

To solve subsystem two, we can collapse the $x - y$ plane onto a combined axis that we call r-axis. This is possible as all the information contained in the $x - y$ plane is only relevant for subsystem one. A diagram showing subsystem two is shown in Figure 8.

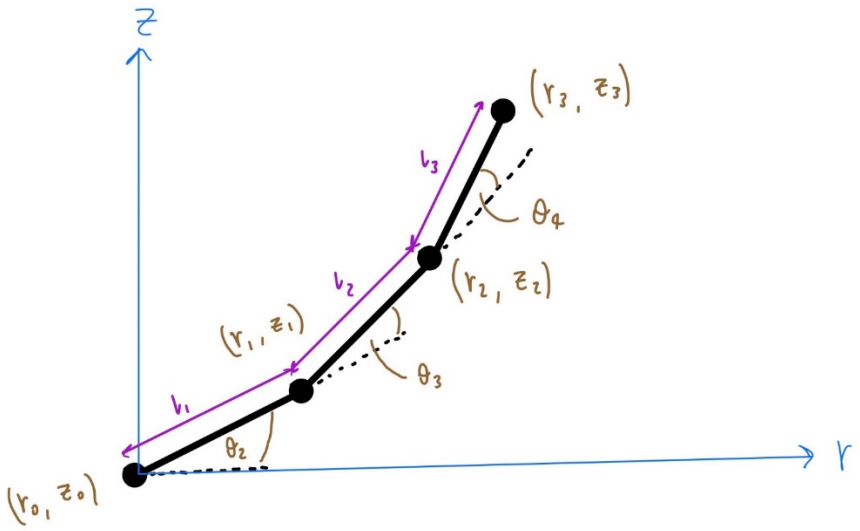


Figure 8: Subsystem 2

Note that subsystem two is translated by distance d_1 along the world frame's z-axis. Hence, $z_3 = z_{tool} - d_1$.
Collapsing the $x - y$ plane onto an r-axis: $r_3 = \sqrt{{x_{tool}}^2 + {y_{tool}}^2}$.

To solve subsystem two, recall the DH-table (simplified using variables):

Transformation step i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	0	d_1	0
3	$\pi/2$	0	0	0
4	0	0	0	θ_2
5	0	l_1	0	θ_3
6	0	l_2	0	θ_4
7	0	l_3	0	0

Table 2: DH-table

Column one represents subsystem one, columns two and three represent the connection between the two systems, and columns four to seven represent subsystem two.
Writing out the homogenous transformation matrices for the columns representing subsystem two, where c_{23} is a short form for $\cos(\theta_2 + \theta_3)$:

$$T_3^4 = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^5 = \begin{bmatrix} c_3 & -s_3 & 0 & l_1 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^6 = \begin{bmatrix} c_4 & -s_4 & 0 & l_2 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^7 = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finding the gripper pose relative to the origin frame of subsystem 2:

$$T_3^7 = T_3^4 T_4^5 T_5^6 T_6^7 = \begin{bmatrix} a_1 & b_1 & d_1 & p_x \\ a_2 & b_3 & d_2 & p_y \\ a_3 & b_3 & d_3 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where a, b, d, p can be written out as follows:

$$a_1 = c_2 c_3 c_4 - c_4 s_2 s_3 - c_2 s_3 s_4 - c_3 s_2 s_4$$

Using trigonometric addition of angle formulae, we can simplify this to $a_1 = c_{234}$.

Similarly, we can obtain

$a_2 = s_{234},\ a_3 = 0,\ b_1 = -s_{234},\ b_2 = c_{234},\ b_3 = 0,\ d_1 = 0,\ d_2 = 0,\ d_3 = 1,$

$p_x = l_1 c_2 + l_3 c_{234} + l_2 c_{23},\ p_y = l_1 s_2 + l_3 s_{234} + l_2 s_{23},\ p_z = 0$

Resulting in:

$$T_3^7 = \begin{bmatrix} c_{234} & -s_{234} & 0 & l_1 c_2 + l_3 c_{234} + l_2 c_{23} \\ s_{234} & c_{234} & 0 & l_1 s_2 + l_3 s_{234} + l_2 s_{23} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rotation part of this transformation matrix is a 2D rotation matrix with angle $\theta_2 + \theta_3 + \theta_4$. The meaning of this is that the gripper orientation in subsystem two is $\emptyset = \theta_2 + \theta_3 + \theta_4$. This corresponds to the gripper orientation in the world frame as subsystem one does not affect the gripper orientation.

The translation part of this transformation matrix can be written in the $r - z$ frame as follows, where $z_3 = z_{tool} - d_1$ as shows previously:

$$r_3 = l_1 c_2 + l_3 c_{234} + l_2 c_{23}$$
$$z_3 = l_1 s_2 + l_3 s_{234} + l_2 s_{23}$$

Using $\emptyset = \theta_2 + \theta_3 + \theta_4$, all already known values can be moved to the left-hand side:

$$r_3 - l_3 c_\emptyset = l_1 c_2 + l_2 c_{23}$$
$$z_3 - l_3 s_\emptyset = l_1 s_2 + l_2 s_{23}$$

Squaring both sides:

$$(r_3 - l_3 c_\emptyset)^2 = {l_1}^2 {c_2}^2 + {l_2}^2 {c_{23}}^2 + 2 l_1 c_2 l_2 c_{23}$$
$$(z_3 - l_3 s_\emptyset)^2 = {l_1}^2 {s_2}^2 + {l_2}^2 {s_{23}}^2 + 2 l_1 s_2 l_2 s_{23}$$

Adding the squared terms and simplifying:

$$(r_3 - l_3 c_\emptyset)^2 + (z_3 - l_3 s_\emptyset)^2 = {l_1}^2 ({c_2}^2 + {s_2}^2) + {l_2}^2 ({c_2}^2 + {s_2}^2) + 2 l_1 l_2 (c_2 c_{23} + s_2 s_{23})$$
$$(r_3 - l_3 c_\emptyset)^2 + (z_3 - l_3 s_\emptyset)^2 = {l_1}^2 + {l_2}^2 + 2 l_1 l_2 c_3$$

Expressing c_3 in terms of already known values:

$$c_3 = \frac{(r_3 - l_3 c_\emptyset)^2 + (z_3 - l_3 s_\emptyset)^2 - {l_1}^2 - {l_2}^2}{2 l_1 l_2}$$

Note that a solution is not valid if $|c_3| > 1$.

Solving for θ_3:

$$s_3 = \pm\sqrt{1 - {c_3}^2}$$
$$\theta_3 = atan2(s_3, c_3)$$

The two options for θ_3 represent the elbow-up and elbow-down solutions.

Using the gripper orientation $\emptyset$, (r_2, z_2) can be obtained from (r_3, z_3) as shown in Figure 9.

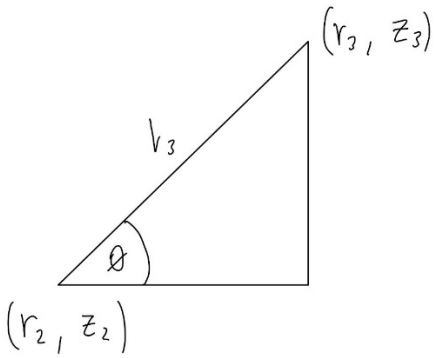


Figure 9: Subsystem 2 triangle

$$c_{\emptyset} = \frac{r_3 - r_2}{l_3}$$

$$s_{\emptyset} = \frac{z_3 - z_2}{l_3}$$

$$r_2 = r_3 - l_3 c_{\emptyset}$$

$$z_2 = z_3 - l_3 s_{\emptyset}$$

The remaining part of subsystem two can now be considered as classical 2R system. Solving such a 2R system has been demonstrated in the lectures.

Using Figure 8 and basic geometry:

$$r_2 = r_3 - l_3 c_{\emptyset} = l_1 c_2 + l_2 c_{23} = k_1 c_2 - k_2 s_2$$
$$z_2 = z_3 - l_3 s_{\emptyset} = l_1 s_2 + l_2 s_{23} = k_1 s_2 - k_2 c_2$$

Where $k_1 = l_1 + l_2 c_3$ and $k_2 = l_2 s_3$

Consequently, using Figure 8:

$$\theta_2 = atan2(z_2, r_2) - atan2(k_2, k_1)$$

Using $\emptyset = \theta_2 + \theta_3 + \theta_4$:

$$\theta_4 = \emptyset - \theta_2 - \theta_3$$

This completes the joint angle calculations for the first two inverse kinematics solutions. These two solutions represent the elbow-up and elbow-down solutions for the gripper reaching forward. Another two solutions can be found that represent the elbow-up and elbow-down solutions for the gripper reaching backwards. Reaching backwards can be achieved by rotating joint angle one by an additional 180 degrees. The calculations are basically the same with the only difference being that we need to add 180 degrees to joint angle one and use $-r_2$ instead of r_2 in the calculations for subsystem two. Noting that the sign reversal also applies to the calculation of θ_3 where $r_3 - l_3 c_{\emptyset} = r_2$. This gives a total of four inverse kinematics solutions.

Note, that the obtained angles do not yet represent the actual joint angles used in the robot modelling due to the DH table simplifications. However, these angles can easily be converted into the actual joint angles by equating the theta columns of the two DH tables (Table 1 and Table 2).

Cubic Spline Interpolation

Cubic spline interpolation is an extension of the cubic interpolation between two points shown in the lectures. It allows us to connect n waypoints into a single trajectory with velocity and acceleration continuity. This is achieved by concatenating n-1 cubic polynomials so that they pass through all N waypoints (control points) and are continuous in velocity and acceleration (Hlaváč, 2021). The resulting function has the following structure when interpolating (x, y) points:

$$f(x) = \begin{cases} a_1 x^3 + b_1 x^2 + c_1 x + d_1 & \text{if } x \in [x_1, x_2] \\ a_2 x^3 + b_2 x^2 + c_2 x + d_2 & \text{if } x \in (x_2, x_3] \\ \dots \\ a_n x^3 + b_n x^2 + c_n x + d_n & \text{if } x \in (x_n, x_{n+1}] \end{cases}$$

The $4n$ coefficients can be determined using the following equations (Denk, 2017).

Ensuring position continuity:

$$\begin{aligned} a_1 x_1^3 + b_1 x_1^2 + c_1 x_1 + d_1 &= y_1 \\ a_1 x_2^3 + b_1 x_2^2 + c_1 x_2 + d_1 &= y_2 \\ a_2 x_2^3 + b_2 x_2^2 + c_2 x_2 + d_2 &= y_2 \\ a_2 x_3^3 + b_2 x_3^2 + c_2 x_3 + d_2 &= y_3 \\ &\dots \\ a_n x_n^3 + b_n x_n^2 + c_n x_n + d_n &= y_n \\ a_n x_{n+1}^3 + b_n x_{n+1}^2 + c_n x_{n+1} + d_n &= y_{n+1} \end{aligned}$$

Ensuring velocity continuity (first derivative):

$$\begin{aligned} 3a_1 x_2^2 + 2b_1 x_2 + c_1 &= 3a_2 x_2^2 + 2b_2 x_2 + c_2 \\ 3a_2 x_3^2 + 2b_2 x_3 + c_2 &= 3a_3 x_3^2 + 2b_3 x_3 + c_3 \\ &\dots \\ 3a_{n-1} x_n^2 + 2b_{n-1} x_n + c_{n-1} &= 3a_n x_n^2 + 2b_n x_n + c_n \end{aligned}$$

Ensuring acceleration continuity (second derivative):

$$\begin{aligned} 6a_1 x_2 + 2b_1 &= 6a_2 x_2 + 2b_2 \\ 6a_2 x_3 + 2b_2 &= 6a_3 x_3 + 2b_3 \\ &\dots \\ 6a_{n-1} x_n + 2b_{n-1} &= 6a_n x_n + 2b_n \end{aligned}$$

Together these give $4n - 2$ equations. Two more equations are required for solving the system of equations. These can be obtained using so called boundary conditions (Denk, 2017). An option for these boundary conditions is to set the second derivate of the first and the last polynomial equal to zero (known as natural spine):

$$\begin{aligned} 6a_1 x_1 + 2b_1 &= 0 \\ 6a_n x_{n+1} + 2b_n &= 0 \end{aligned}$$

These equations form a linear system of equations that can be combined into a matrix and solved using Gaussian elimination.
To apply this technique to waypoints, we need to apply them to all variables describing that waypoint. The variable represents the y-coordinate while time represents the x-coordinate.

Note on the role of simulation

The initial coursework specifications specified that each team would only have six hours with their robot. It was clear that this would not be sufficient time for testing all the required features. Hence, it was determined that having a realistic simulation would be the most crucial factor for success in this coursework. This would then enable using the time on the actual robot efficiently by solely fine-tuning features that were already extensively tested in simulation.
To achieve realistic simulation, it was decided to maintain up-to-date versions of the codebase in both MATLAB (for the real robot) and Python (for a Gazebo simulation). Robotis provides a Gazebo simulation of the OpenMANIPULATOR-X (Robotis, n.d.) that includes numerous ROS topics and services for sending commands to the simulated robot's servos. A custom controller was written,

as a ROS node, for controlling the robot through the provided endpoints. A screenshot of the Gazebo simulation is shown in Figure 10.

This effort of maintaining two versions of the codebase for simulation purposes was eventually given up. The reasons were time constraints, better than expected MATLAB simulations, difficulties with obtaining appropriate friction values in Gazebo to enable picking up objects, and increased robot availability. The increased physical robot availability, from six hours to almost unlimited, almost completely removed the need for simulation. Everything could simply be tested on the robot directly without worries about time being wasted. Moreover, the developed MATLAB simulation allowed one to visualize all robot trajectories in real-time (by plotting every set-point), which is not much worse than a Gazebo simulation with non-working friction values. A screenshot of the MATLAB simulation for task 4 is shown in Figure 11.

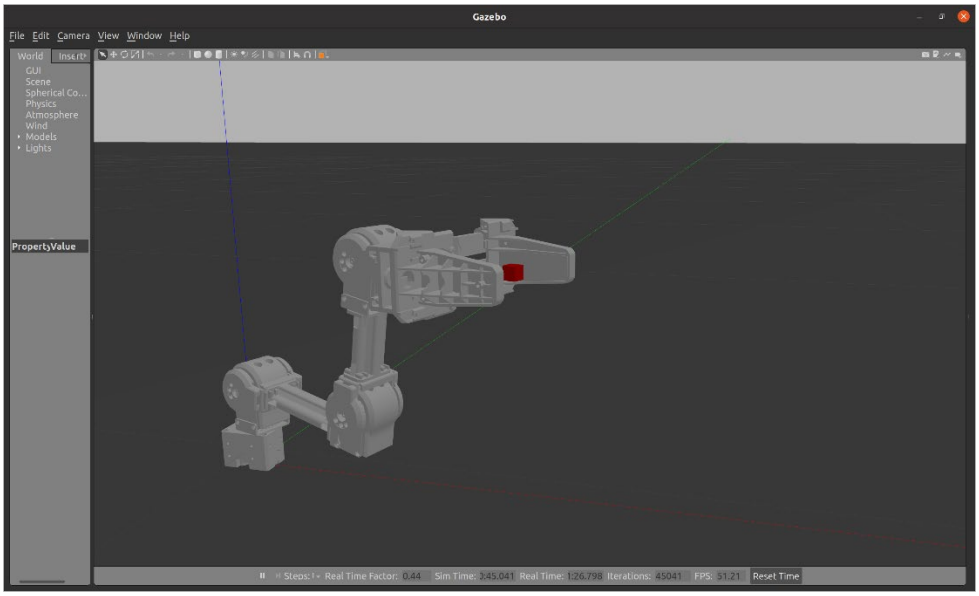


Figure 10: Simulating the OpenMANIPULATOR-X in Gazebo

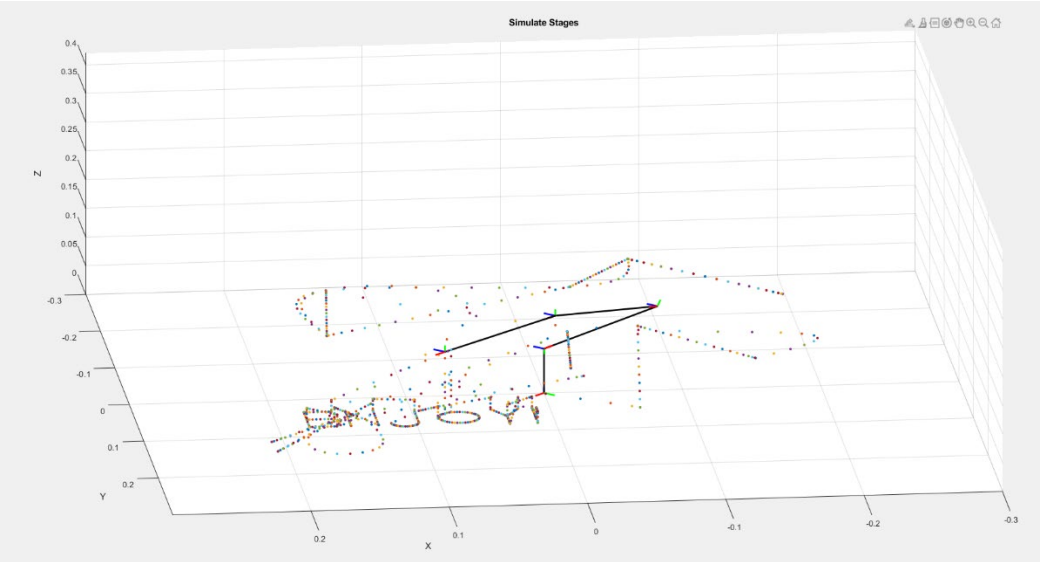
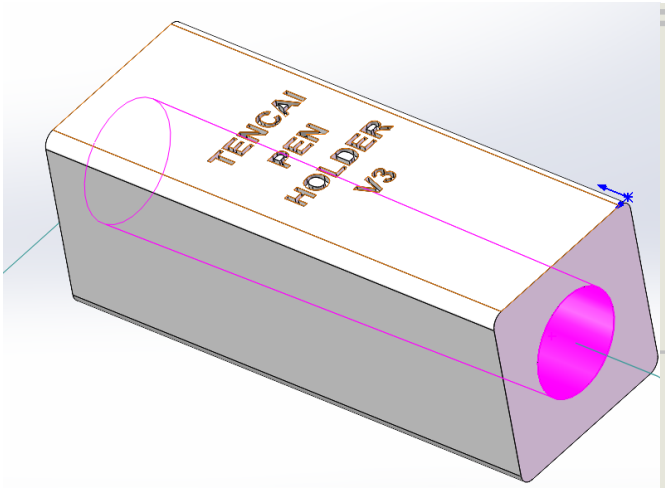
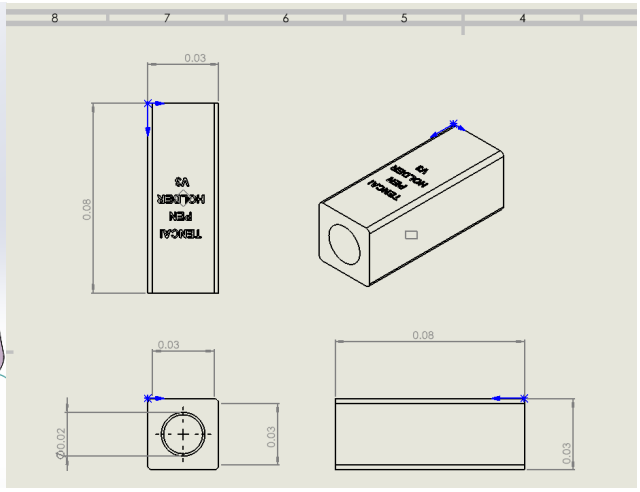


Figure 11: Screenshot of completed task 4 MATLAB simulation

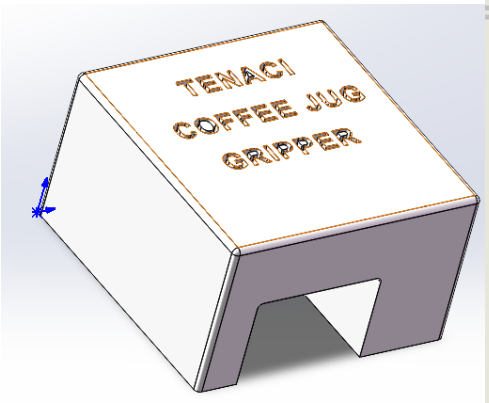
Pen holder CAD drawing

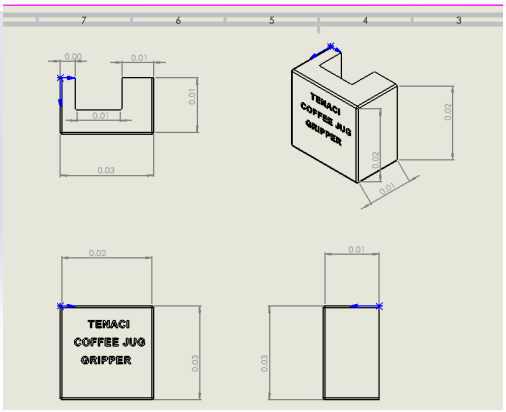




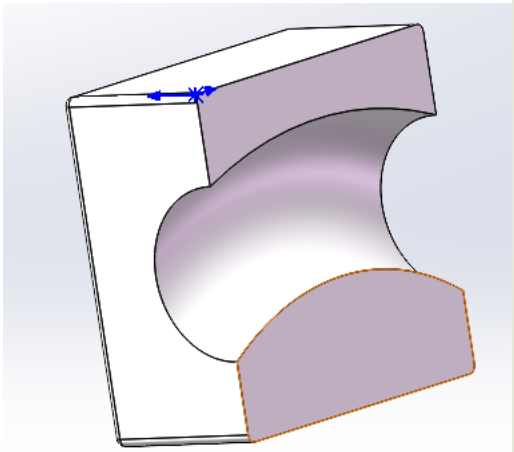
Task 4 tool adaptor CAD drawings

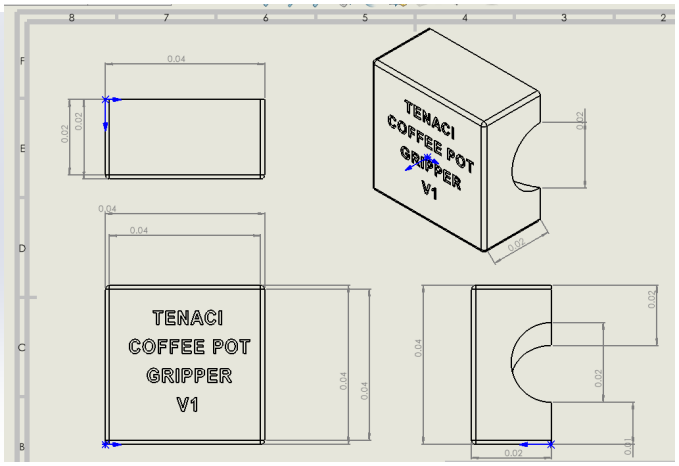
Coffee pot adaptor





Tea pot adaptor





References

Denk, T., 2017. *Cubic Spline Interpolation.* [Online]
Available at: https://timodenk.com/blog/cubic-spline-interpolation/#:~:text=Cubic%20spline%20interpolation%20is%20a,of%20multiple%20cubic%20piecewise%20polynomials.
[Accessed 26 February 2022].

Hlaváč, V., 2021. *Robot trajectory generation.* Prague: Czech Technical University in Prague.

Melore, C., 2021. *StudyFinds.* [Online]
Available at: https://www.studyfinds.org/morning-people-no-time-breakfast/
[Accessed 10 March 2022].

Moley Robotics, 2022. *Moley Robotics.* [Online]
Available at: https://moley.com/

Robotis, n.d. *Robotis e-Manual: XM430-W350.* [Online]
Available at: https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/
[Accessed 8 3 2022].

Robotis, n.d. *Simulation.* [Online]
Available at: https://emanual.robotis.com/docs/en/platform/openmanipulator_x/ros_simulation/#launch-gazebo

Spence, C., 2017. Breakfast: The most important meal of the day?. *International Journal of Gastronomy and Food Science,* Volume 8, pp. 1-6.