Y3 Robotics Manipulation Report - BOB

Alexander Pondaven 01730117   Tian Yi Lim 01564851   Shuanghua Liu 01729607

1 Task 1 - Modelling the Robot

1.1 Coordinate Frame Assignment and DH Notation Table

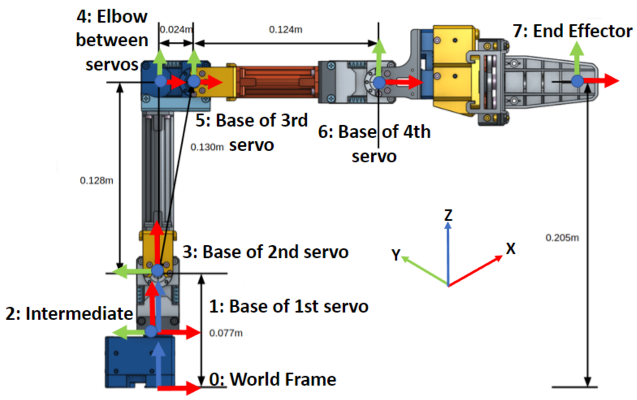

Figure 1: Coordinate Frame Assignment

i	α_{i-1}	a_{i-1}	d_i	θ_i	Remarks
0	0	0	0	0	World Frame
1	0	0	34	θ_1	Base of first servo
2	$\frac{\pi}{2}$	0	0	$\frac{\pi}{2}$	Intermediate
3	0	43	0	θ_2	Base of second servo
4	0	128	0	$-\frac{\pi}{2}$	Elbow between servos
5	0	24	0	θ_3	Base of third servo
6	0	124	0	θ_4	Base of fourth servo
7	0	126	0	0	End effector

Table 1: Task 1 DH Notation Table

The chosen coordinate frames are drawn in Figure 1 (Image from ROBOTIS e-Manual for OpenMANIPULATOR-X). The world frame is at the base of the robot and the first servo's frame is just above it. An intermediate frame is required to rotate around the x-and z-axes, so that the next frame at the base of the second servo is correctly positioned with the x-axis aligned with the link. Another frame is added at the right angle at the top right (fixed joint) and the next frame (i=5) describes the third servo. The fourth servo's frame is just along the link length. The end effector's frame is a fixed distance along the gripper from the previous frame. Each joint angle θ_i is also given in the DH Table 1.

1.2 Robot Graphical Simulation

The arm can be visualised in MATLAB as seen in Figure 2. Each link is a black line and each coordinate frame is found by multiplying the transformation matrices and plotting the three columns of the rotation matrix as the basis vectors. Each vector in the frame is coloured as: X - red, Y - green, Z - blue. The world frame is made larger than the others.

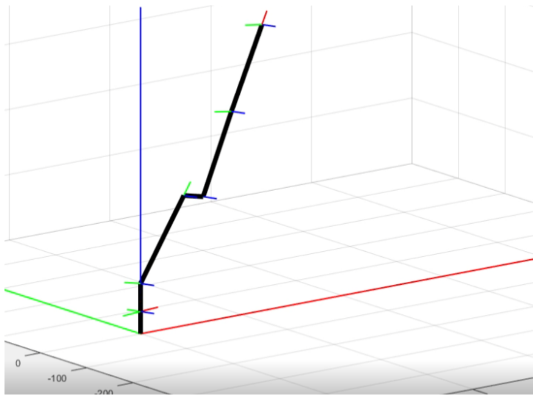

Figure 2: Task 1b Graphical Simulation in Matlab of arm and coordinate frames

1.3 Inverse Kinematics Solution

The IK problem can be simplified by realising that θ_1 can be calculated directly from the end effector position x, y, z as the arm moves radially around its centre and all other joints move in the same 2D plane. This means that transformation matrices T_i in the IK become 3x3. Therefore, the new coordinates in this plane can be defined as shown in Figure 5.

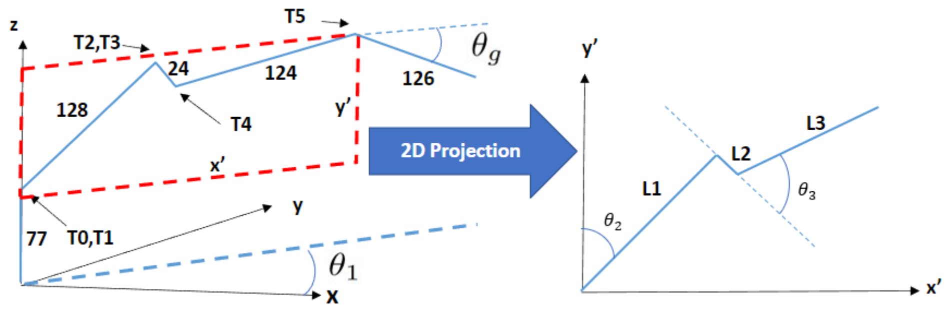

Figure 3: Task 1 IK Problem Re-formulation

The final link's angle can also be given as a parameter θ_g, the angle of the end effector relative to the world frame, as the tasks like picking up a cube must be done with the end effector pointing at a particular angle. This simplifies the problem to solving for θ_2 and θ_3 to get to the position of the joint before the end effector. The final angle θ_4 can then be derived from these results and θ_g. All IK calculations are in Appendix with the final results shown below.

$$\theta_1 = atan2(y, x)\ ,\ x' = \sqrt{x^2 + y^2} \text{ and } y' = z - 77$$

$$\theta_3 = atan2(\sin(\theta_3 + \alpha), \cos(\theta_3 + \alpha)) - \alpha \text{ where } \sin(\theta_3 + \alpha) = \frac{(x')^2 + (y')^2 - L_3^2 - L_2^2 - L_1^2}{2L_3\sqrt{L_2^2 + L_1^2}} \text{ and } \alpha = \arctan\frac{L_2}{L_1}$$

$$\theta_2 = atan2(\sin(\theta_2 + \alpha_2), \cos(\theta_2 + \alpha_2)) - \alpha_2$$
$$\text{where } \sin(\theta_2 + \alpha_2) = \frac{y'}{\sqrt{k_1^2 + k_2^2}}\ ,\ \alpha_2 = \arctan\frac{k_2}{k_1}\ ,\ k_1 = L_3c_3 + L_2 \text{ and } k_2 = L_3s_3 + L_1$$

$$\theta_4 = \theta_g - \theta_2 - \theta_3$$

1.4 Inverse Kinematic Simulation

A square can be drawn as in Figure 4 by linearly interpolating x, y, z points and doing IK on each position to get each joint's angle. The square is also traced out with coloured line objects in Matlab. Note that squares look distorted from the 3D point of view.

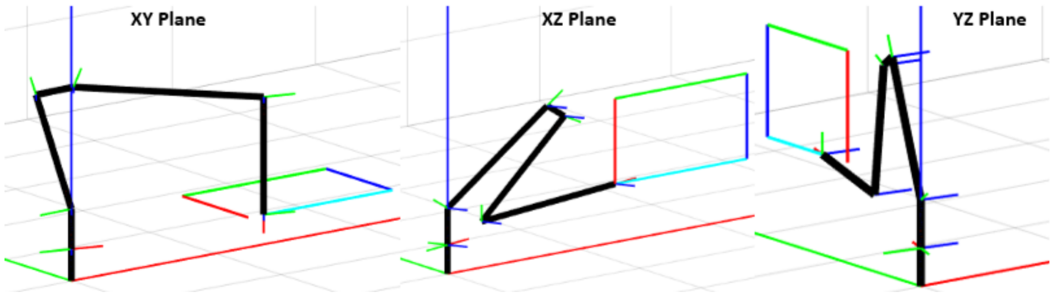

Figure 4: Task 1d IK Square tracing in all planes (Recording of simulation in video demo)

2 Task 2 - Pick and Place (Wooden Blocks)

2.1 Gripper Design

While the standard gripper is adequate for the task, problems arise when attempting to rotate cubes when the cube holder is too far away from the robot base, as the arm is too short to raise the end effector past a certain height given a specified orientation and x, y position. When attempting to grab cubes from cube holders that are in the close vicinity of other cube holders while $\theta_g = 0$, the bottom of the gripper may collide with the cube holders. These problems could be solved in software, but as their root cause was in hardware, an alternative gripper was designed.

Our alternative gripper (Figure 13) has a shorter distance from the adapter plates. This makes it possible to pick and place cubes with $\theta_g = -\frac{\pi}{2}$ further away from the arm and $\theta_g = 0$ closer to the arm. It is also offset from the centre of the gripper. This allows it to avoid hitting the lower part of the gripper on cube holders. With our alternative gripper, two additional entries are needed in the DH table, as shown in Table 2. One entry rotates the direction of the gripper 90° downward, and the next accounts for the additional distance of the end effector from the centre line of the gripper servo, labelled L5.

i	α_{i-1}	a_{i-1}	d_i	θ_i	Remarks
8	0	0	0	$-\frac{\pi}{2}$	Rotation downwards
9	0	L5	0	0	New end effector position

Table 2: Task 2 DH Notation Table Extension

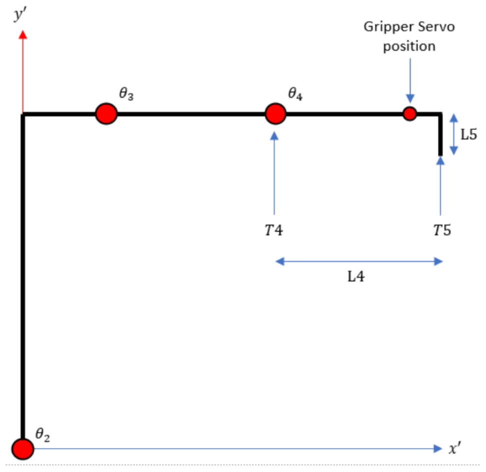

Figure 5: Task 2 modified linkage diagram

The standard inverse kinematics is adapted to the new gripper by calculating the coordinates of the gripper servo from the specified end effector coordinates.

To obtain $T4$ as before, we express $\Delta x'$ and $\Delta y'$ as the change in x' and y' from $T4$ to $T5$. We then obtain

$$\theta_{go} = \theta_g - atan2(L5, L4)$$
$$D_{go} = \sqrt{L5^2 + L4^2}$$
$$\Delta y' = D_{go} \sin(\theta_{go})$$
$$\Delta x' = D_{go} \cos(\theta_{go})$$

We then obtain

$$T4_{x'} = T5_{x'} - \Delta x'$$
$$T4_{y'} = T5_{y'} - \Delta y'$$

And the rest of the IK calculation follows.

2.2 Software Approach

A generalisable approach (see Figure 10) was taken to moving cubes. One must only specify the cube holder locations and the steps required to manipulate the cubes. This breaks down the movement of the cubes simply into their start and end cube holders and the required change in orientation. Furthermore, it allows for the cubes to be rotated and translated simultaneously. It then automatically calculates the entire trajectory for the arm before writing those joint angle instructions for the path to the servos.

2.3 Path Planning

The first step to generic motion is to find a feasible, collision-free path through the task space. To this end, A* search with some task-specific optimisations was applied.

A* search To move from one end effector orientation to another, four parameters are varied: The Cartesian locations (x, y, z) and orientation of the end effector θ_g. Applied naively, this results in a 4-dimensional search space, leading to long search times. However, note that if the start and end orientation are both feasible, then all intermediate orientations will also be feasible due to the lack of obstacles above the ground plane. Therefore, A* search only needs to be applied in 3 dimensions. We use a Euclidean distance heuristic as the robot arm can move arbitrarily in the occupancy grid coordinate system defined.

Creating Occupancy Grid The occupancy grid is a 3-dimensional grid in $|x', y', \theta_1|$ space. Choosing such a coordinate system more naturally maps to the IK that underpins the robot's motion, which simplifies further calculations. To create the grid, its limits and resolution are first defined in `createOccupancyGrid`. The input cube and cube holder centre locations are then translated from x, y, z Cartesian space to the modified occupancy grid space.

All possible grid locations are then iterated over, and if a grid location is within the bounding box of a cube or cube holder, it is considered 'occupied' and thus marked with a 1. If not, it is left as a 0. For x' and y' values, upper and lower bounds are straightforward to calculate. For θ_1 values, the upper and lower bounds α_1, α_3 are obtained by adding the angle to the near corner of the cubes to α_2, the θ_1 orientation of the cube. This is illustrated in Figure 6.

End Effector orientation interpolation As A* search does not search over end effector orientation, it is possible that for a commanded cube rotation, only two waypoints are produced (as the start and end waypoints differ only in orientation, not location). There is no guarantee that the resulting large movements in joint space will be collision-free. Therefore, it is necessary for a minimum number of task-space waypoints to be generated for every planned path if the end effector orientation changes.

To this end, the difference in orientation between waypoints is used to determine how many points to linearly interpolate for in task space. For smooth performance, we interpolate to obtain 15 intermediate points per 90° orientation change. This ensures that the end effector does not change in position while performing large orientation changes.

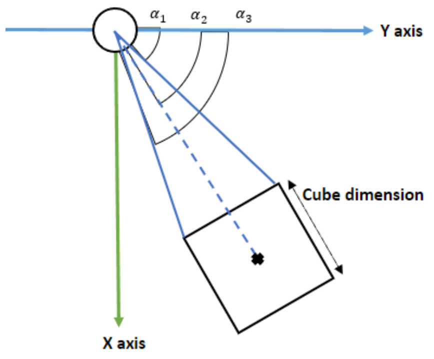

Figure 6: Occupancy Grid Visualisation

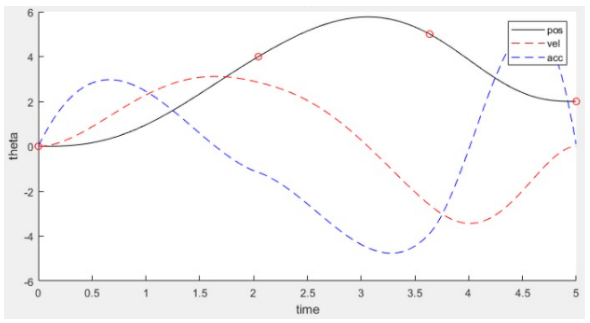

Figure 7: Quintic interpolation between waypoints

2.4 Trajectory Planning

Setting joints to linearly move from one waypoint to another led to jittery movement as joints did not arrive at the specified angle at the same time. If one angle is much smaller than the others, it moves quickly and stops, which is not smooth.

Quintic interpolation Trajectory planning resolves this issue by fitting the joint angles at each waypoint (via point) with a curve according to several constraints. The joint angles at each via point were fit with a 5th degree polynomial (quintic) to have a smooth cubic acceleration (2nd derivative). Each segment between via points fit its own quintic with 6 coefficients according to several constraints. These constraints enforced the same position, velocity and acceleration before and after each via point, which made the entire path smooth even though each segment was fit with its own quintic. The beginning and end started at zero velocity and acceleration. This became a simultaneous equation to solve in MATLAB (`interpQuinticTraj`) and the curve can be seen plotted in Figure 7. Each segment between via point angles for each joint is fit with its own quintic.

Assign Via Times The interpolation assumes knowledge of the time between each via point and the total time required for the path. This was important to tune correctly to avoid spikes in acceleration but to also move as quick as possible. Several heuristics were developed to help automatically determine these values. The total trajectory time was estimated as proportional to the sum of the max distance travelled out of the joints over all segments. The linear heuristic spaced the via times evenly over time. A velocity heuristic was also developed that assigns more time to segments with larger difference in joint angles. The acceleration heuristic used a 1D Laplace filter (difference in gradients) over the joint angles at via points to estimate the 2nd derivative (acceleration). More time was assigned to segments with a larger acceleration to avoid jerky movement and this metric led to the least amount of jerky movement compared to other heuristics, for example linearly spacing time points, or using a velocity-based heuristic.

2.5 End effector movement

Position control mode with waypoints was found to give jerky behaviour. Thus, Velocity control mode was used for the Dynamixels excluding the gripper. A Feedforward PID controller was used, with characteristic equation $v_{out} = v_{feedforward} + K_p e(t) + K_i \int e(t)dx + K_d \frac{de(t)}{dt}$

The controller loops until the last waypoint has been reached. Over each loop, it samples the intended velocity from the appropriate set of quintic coefficients for the feedforward term. The error term is obtained by comparing the current intended position (sampled from quintic) with the current servo position. Summing the error with an accumulator provides the integral component, while comparing the difference with the previous error provides the differential component.

The controller requires reading the current position and writing the desired velocity. In addition, the current controller velocity is also read out for debugging purposes. Using the `read4ByteRxTx` syntax thus requires 12 separate read/write operations to be performed each loop, which has a long latency. This then results in a lower sampling frequency, degrading the performance of the velocity controller. Hence, the `groupSyncRead` syntax was used, allowing for effectively three read/write operations per loop, improving the controller's responsiveness.

3 Task 3 - Trajectory Following (Drawing)

3.1 Gripper Design

The gripper design for this task involves a pen holder that holds the pen vertically, and a gripper that grips the pen vertically. The arm holds the pen with $\theta_g = 0$ using force closure. Non-slip matting was added to maximise the friction force and also deforms to act as form closure. Some cutouts were also added to decrease 3D printing time, allowing for a faster turnaround.

3.2 Software Approach

The approach for Task 3 (see Figure 11) involved following a trajectory by specifying the current position of the end effector, the lines' endpoint coordinates, the arc start point, center and angle of rotation. The waypoints are linearly interpolated and the arm follows this sequence: initial position→ pick up position→white board→drawing→deposit position. The joint angles at each via point are calculated using IK and via times are assigned for quintic interpolation. Joint angles are then written to servos using the feedforward PID controller.

3.3 Path Definition

The path for this task consists of three parts: picking up the pen, drawing, and putting the pen back. The picking up task first reads all servos' positions, and obtains an initial end effector coordinate using forward kinematics. The other coordinates are the positions for hovering above the pen, gripping the pen, hovering again, and moving to the start position of drawing.

The drawing task is defined by three lines and one arc. The waypoints for the lines are linearly interpolated between the endpoints of the lines. The waypoints of the semicircle are also linearly interpolated around its circumference.

The coordinates for putting the pen back are the final position of drawing and the hovering positions of the pen holder as defined in the picking task. All waypoints are generated linearly between these coordinates.

The semicircle function is a generic approach of generating the arc, which takes the center and start point of the arc, the angle of the arc, and the number of sample points. The way points are generated by $X_{arc} = radius * \cos(\theta) + X_{center}$ and $Y_{arc} = radius * \sin(\theta) + Y_{center}$. The angle θ is sampled linearly, so the waypoints are linearly spaced along the arc.

4 Task 4 - Buzz Wire Game

Our team wished to investigate the possibilities of alternative gripper designs that afforded an extra degree of freedom within the constraints of the coursework specification. This was for the robot arm to play a Buzz Wire game where the robot must manipulate a buzzer in such a way to prevent the metal loop from touching the wire. In doing so, the robot must be able to rotate the buzzer along its lengthwise axis, something not possible with the default robot gripper. This could be seen as a simulation of a robot performing a delicate operation, for example in a surgical operation.

The original game can be seen in Figure 35. The buzz wire device was modified to take a piece of brass wire as the wire to allow us to more easily shape the path followed by the robot (seen in Figure 36). In addition, the buzz wire was constrained to be in two dimensions to allow us to bend the wire more accurately to specific dimensions.

4.1 Software Approach

To complete the Buzz Wire game (see Figure 12), it takes in the configuration of the buzz wire (wire bends) as input and a linear path between these bends are generated. The angle of the buzzer tool was determined by the angle between adjacent coordinates along the path. This angle relative to the world was converted into the required buzzer angle relative to the gripper as explained below. A linear relationship between this buzzer angle and the required gripper angle θ_4 was determined empirically. These waypoints are converted into a trajectory as in previous tasks.

4.2 Gripper Modifications

The gripper jaws were used in a rack-and-pinion style to rotate the buzzer as required (seen Figure 34). This translates the linear motion of the gripper jaws back to a circular motion.

To ensure that the buzzer is kept at a fixed location relative to the gripper servo, rubber bands were used to hold a pair of circular gripper guides in between the gripper jaws. Each rubber band exerts the same tension on the central gripper guide, thereby keeping the buzzer centralised and allowing for the same inverse kinematic calculations to be performed.

The rubber band hooks are delicate and narrow and were unsuitable to be printed together with the main gripper jaws. Therefore, they are separate parts which are subsequently attached using screws. More pictures are in the Appendix. Refer to the video for an example of the mechanism working.

4.3 Calculating Buzzer Orientation

We define the required buzzer angle θ_{bz} as the angle the buzzer needs to be rotated to be parallel to the local buzz wire segment. Bearing in mind the coordinate system defines clockwise rotations as negative, $\theta_{bz} = \theta_B - \theta_1$ for any given combination of hip angle and local buzzer orientation (illustrated in Figure 8).

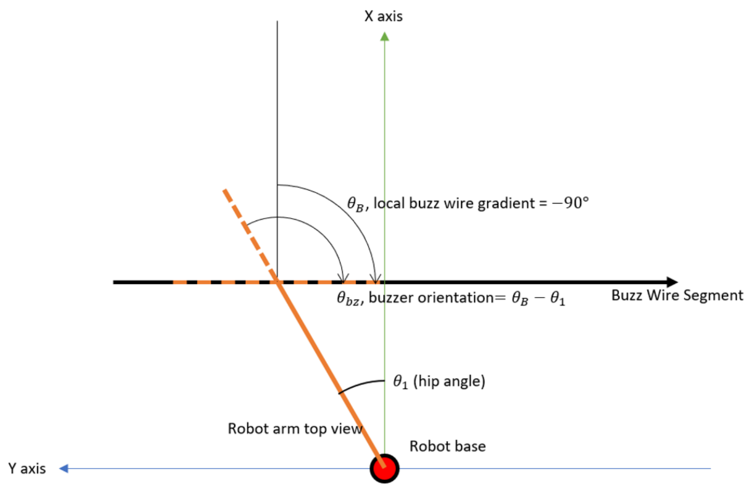
X axis
θ_B, local buzz wire gradient = $-90°$
θ_{bz}, buzzer orientation= $\theta_B - \theta_1$
Buzz Wire Segment
θ_1 (hip angle)
Robot arm top view
Robot base
Y axis

Figure 8: Buzzer Orientation Diagram

However, as the pointwise orientation of the buzzer corresponds to its instantaneous value, this results in sudden, jerky motion near corners. Therefore, a moving average filter is applied to the vector of buzzer orientation values using the `conv` function. Some look-ahead is also applied to the values by reading the filtered vector ahead of the current via point index. This allows the buzzer to rotate before encountering the corner for smoother motion.

To convert buzzer orientation into the actual angle for the gripper servo θ_4, a linear relation between the gripper position and buzzer orientation was assumed. The servo was placed into position control mode and the range of possible values for gripper position, along with their corresponding buzzer angles, were measured. A neutral position was decided as the buzzer angle corresponding to $\theta_1 = 0$ with a buzzer wire parallel to the y-axis, by our convention $\theta_{bz} = -90°$. This was assigned to be the middle of the gripper's range of motion. Empirically, a symmetrical range of motion of $\pm 90°$ from the horizontal was measured. Practically, this reliably allowed for buzz wire paths around $45°$ from the horizontal to be traced, informing our decision-making for the demonstration buzz wire path.

5 Conclusion and Further Work

In conclusion, velocity control was successfully implemented along with quintic interpolation for via points in joint space, allowing for smooth motion of the end effector through the task space for all three practical tasks.

However, due to the occupancy grid for the A* search being relatively sparse for performance reasons, the waypoint selection in task space was sometimes jerky and not direct. This could have been further smoothed out by perhaps using the waypoints as spline control points instead of directly, or some other method of sparsifying the waypoints.

It was observed that it was difficult to command the servos to move smoothly at very low speeds, as the inertia/static friction of the motors was difficult to overcome at low velocity demands. Furthermore, as the arm was extended far away from the base, the effect of gravity on the arm's links was non-negligible, leading to 'nodding' motion. To address this, perhaps a dynamical model of the robot operating on joint torques could be applied instead. In conjunction with this, finding a method to increase the sample rate of the control system would allow for control inputs to occur more responsively to external disturbances.

A IK Calculations

Define transformation matrices $T_0 - T_5$. Note $c_i = \cos\theta_i$ and $s_i = \sin\theta_i$.

$$T_0 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{(Rotate } +90^\circ) \; T_1 = \begin{bmatrix} c_2 & -s_2 & 0 \\ s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} T_2 = \begin{bmatrix} 1 & 0 & 128 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{(Rotate } -90^\circ) \; T_4 = \begin{bmatrix} c_3 & -s_3 & 24 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} T_5 = \begin{bmatrix} 1 & 0 & 124 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Apply transformations to get position in terms of angles

$$T_0T_1 = \begin{bmatrix} -s_2 & -c_2 & 0 \\ c_2 & -s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_0T_1T_2 = \begin{bmatrix} -s_2 & -c_2 & -128s_2 \\ c_2 & -s_2 & 128c_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_0T_1T_2T_3T_4 = \begin{bmatrix} c_2c_3 - s_2s_3 & -c_2s_3 - s_2c_3 & 24c_2 - 128s_2 \\ s_2c_3 + c_2s_3 & c_2c_3 - s_2s_3 & 24s_2 + 128c_2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3) & 24c_2 - 128s_2 \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 24s_2 + 128c_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_0T_1T_2T_3T_4T_5 = \begin{bmatrix} \cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3) & 124\cos(\theta_2+\theta_3) + 24c_2 - 128s_2 \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 124\sin(\theta_2+\theta_3) + 24s_2 + 128c_2 \\ 0 & 0 & 1 \end{bmatrix}$$

The position vector can be found from the rightmost column of the transformation matrix:

$$x' = L_3\cos(\theta_2+\theta_3) + L_2c_2 - L_1s_2$$
$$y' = L_3\sin(\theta_2+\theta_3) + L_2s_2 + L_1c_2$$

$$(x')^2 = L_3^2\cos^2(\theta_2+\theta_3) + L_2^2c_2^2 + L_1^2s_2^2 + 2L_3\cos(\theta_2+\theta_3)L_2c_2 - 2L_3\cos(\theta_2+\theta_3)L_1s_2 - 2L_1L_2s_2c_2$$
$$(y')^2 = L_3^2\sin^2(\theta_2+\theta_3) + L_2^2s_2^2 + L_1^2c_2^2 + 2L_3\sin(\theta_2+\theta_3)L_2s_2 + 2L_3\sin(\theta_2+\theta_3)L_1c_2 + 2L_1L_2c_2s_2$$
$$(x')^2 + (y')^2 = L_3^2 + L_2^2 + L_1^2 + 2L_3L_2\cos\theta_3 + 2L_3L_1\sin\theta_3$$

$$\frac{(x')^2 + (y')^2 - L_3^2 - L_2^2 - L_1^2}{2L_3} = L_2\cos\theta_3 + L_1\sin\theta_3$$

Solve for θ_3

By R-formula, $L_2\cos\theta_3 + L_1\sin\theta_3 = R\sin(\theta_3+\alpha)$ where $R = \sqrt{L_2^2 + L_1^2}$ and $\alpha = \arctan\dfrac{L_2}{L_1}$

$$\therefore \sin(\theta_3+\alpha) = \frac{(x')^2 + (y')^2 - L_3^2 - L_2^2 - L_1^2}{2L_3\sqrt{L_2^2 + L_1^2}}$$

This is checked if feasible. Calculate $\theta_3 = atan2(\sin(\theta_3+\alpha), \cos(\theta_3+\alpha)) - \alpha$. Now to find θ_2:

$$x' = L_3(c_2c_3 - s_2s_3) + L_2c_2 - L_1s_2 = k_1c_2 - k_2s_2$$
$$y' = L_3(s_2c_3 + c_2s_3) + L_2s_2 + L_1c_2 = k_1s_2 + k_2c_2$$

where $k_1 = L_3c_3 + L_2$ and $k_2 = L_3s_3 + L_1$ are known.

By R-formula, $y' = k_1 s_2 + k_2 c_2 = R_2 \sin(\theta_2 + \alpha_2)$ where $R_2 = \sqrt{k_1^2 + k_2^2}$ and $\alpha_2 = \arctan \dfrac{k_2}{k_1}$

Can then calculate $\theta_2 = atan2(\sin(\theta_2 + \alpha_2), \cos(\theta_2 + \alpha_2)) - \alpha_2$ as θ_3 and thus k_1 and k_2 are known.

The final angle $\theta_4 = \theta_g - \theta_2 - \theta_3$ as derived in Figure 9.

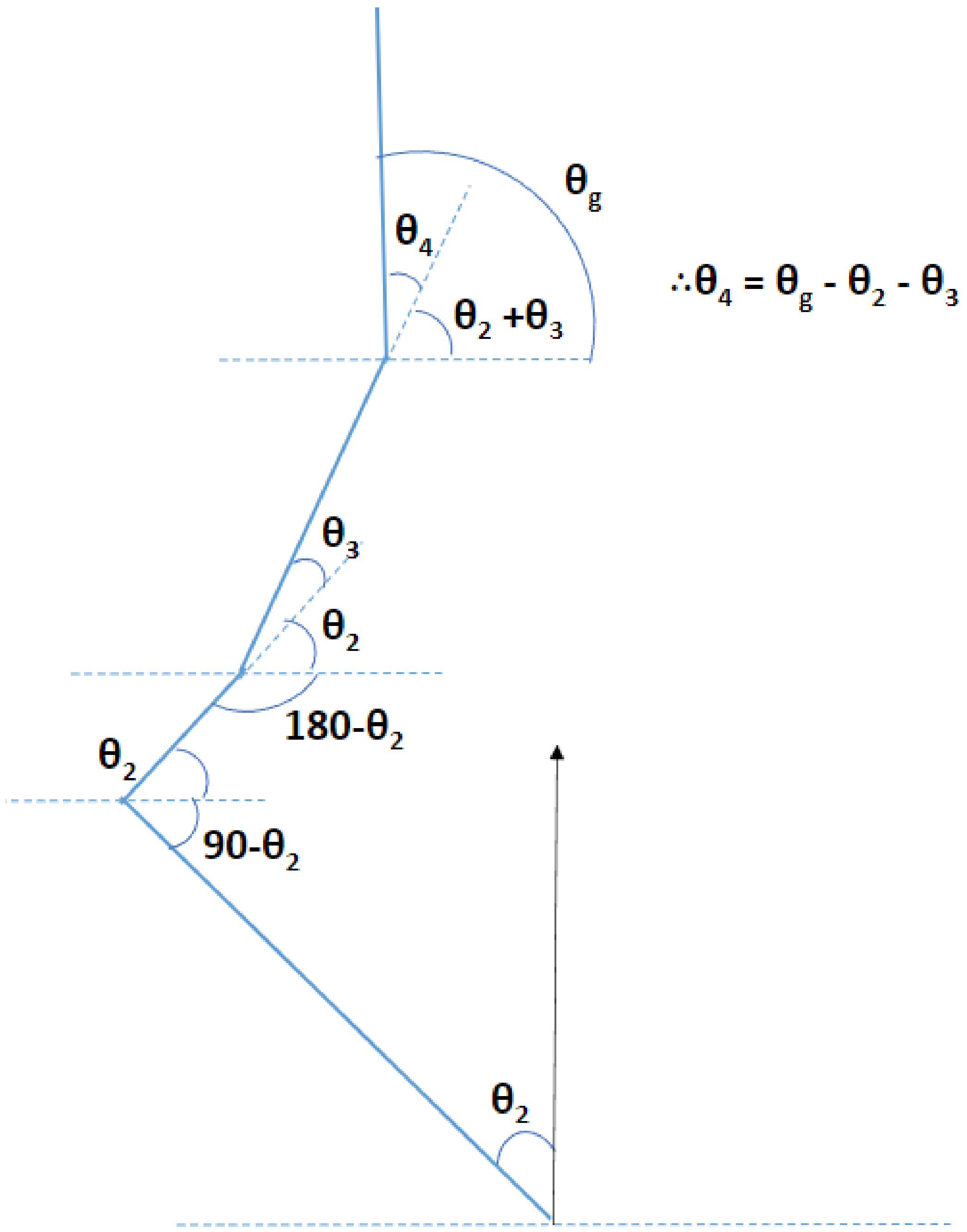

Figure 9: θ_4 derivation from other joint angles

B Software approaches

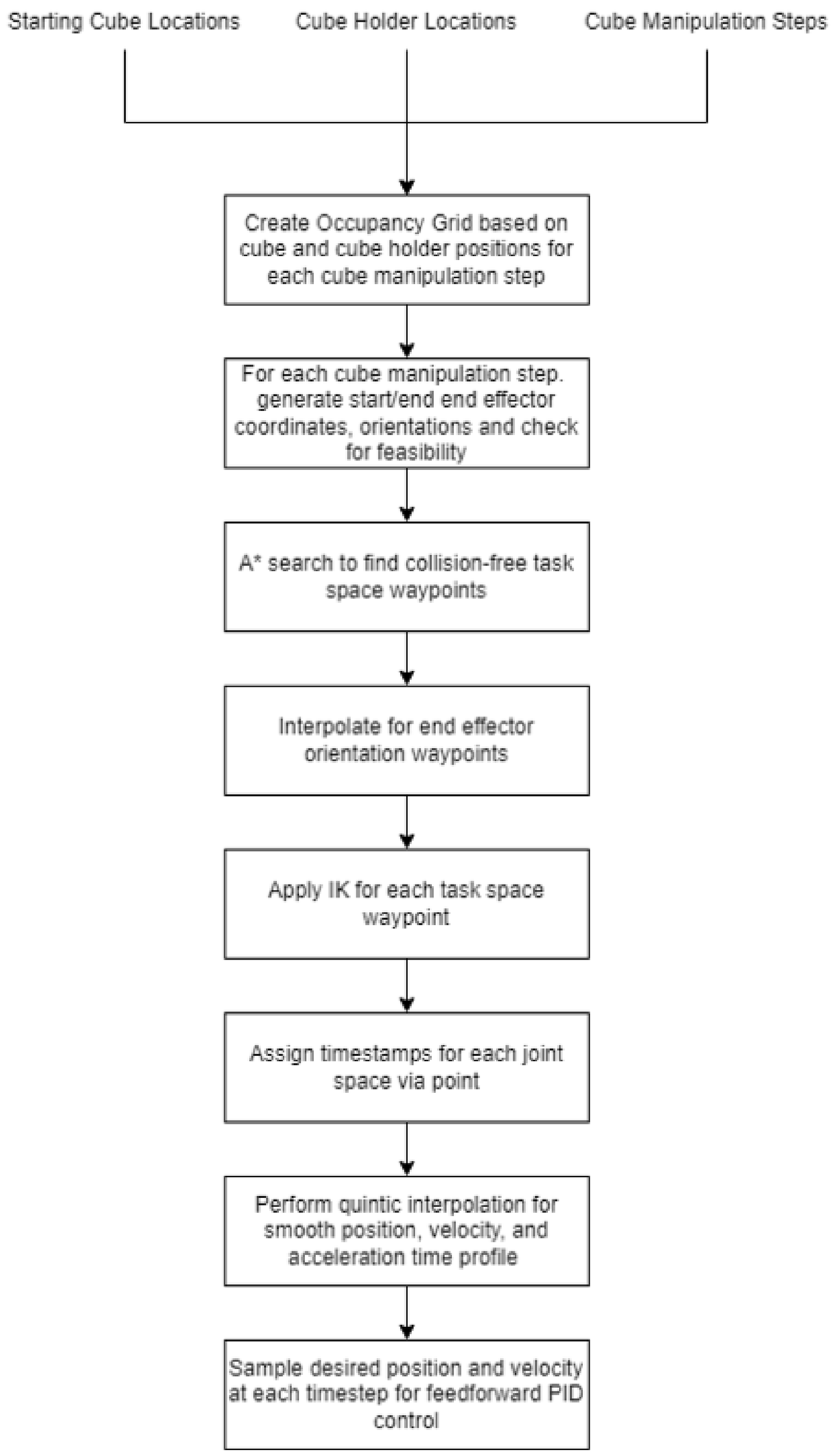
Starting Cube Locations
Cube Holder Locations
Cube Manipulation Steps
Create Occupancy Grid based on cube and cube holder positions for each cube manipulation step
For each cube manipulation step. generate start/end end effector coordinates, orientations and check for feasibility
A* search to find collision-free task space waypoints
Interpolate for end effector orientation waypoints
Apply IK for each task space waypoint
Assign timestamps for each joint space via point
Perform quintic interpolation for smooth position, velocity, and acceleration time profile
Sample desired position and velocity at each timestep for feedforward PID control

Figure 10: Task 2 Software Approach

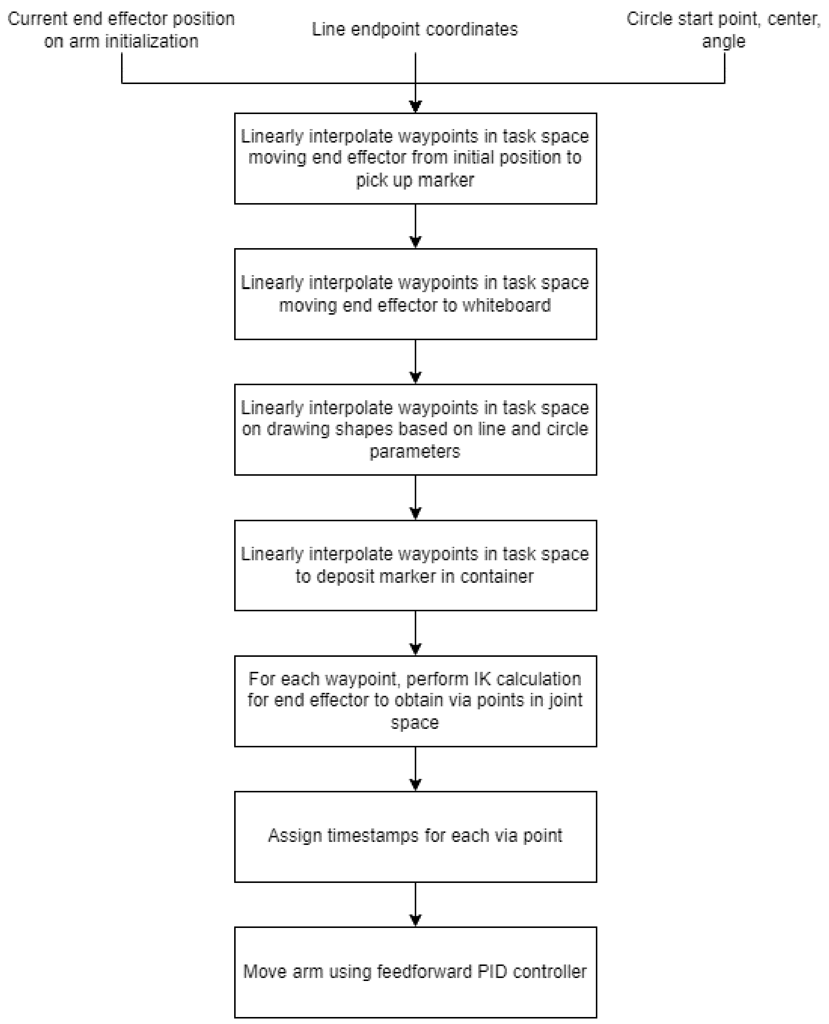
Current end effector position on arm initialization
Line endpoint coordinates
Circle start point, center, angle
Linearly interpolate waypoints in task space moving end effector from initial position to pick up marker
Linearly interpolate waypoints in task space moving end effector to whiteboard
Linearly interpolate waypoints in task space on drawing shapes based on line and circle parameters
Linearly interpolate waypoints in task space to deposit marker in container
For each waypoint, perform IK calculation for end effector to obtain via points in joint space
Assign timestamps for each via point
Move arm using feedforward PID controller

Figure 11: Task 3 Software Approach

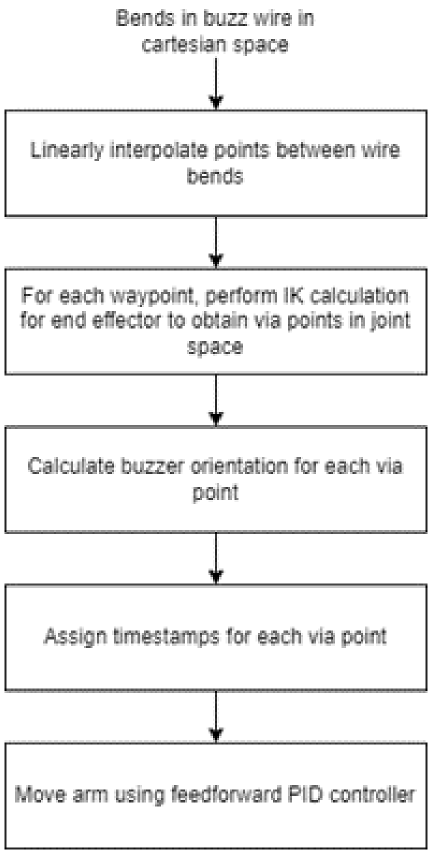
Bends in buzz wire in cartesian space
Linearly interpolate points between wire bends
For each waypoint, perform IK calculation for end effector to obtain via points in joint space
Calculate buzzer orientation for each via point
Assign timestamps for each via point
Move arm using feedforward PID controller

Figure 12: Task 4 software approach

C CAD Screenshots

C.1 Task 2 Gripper

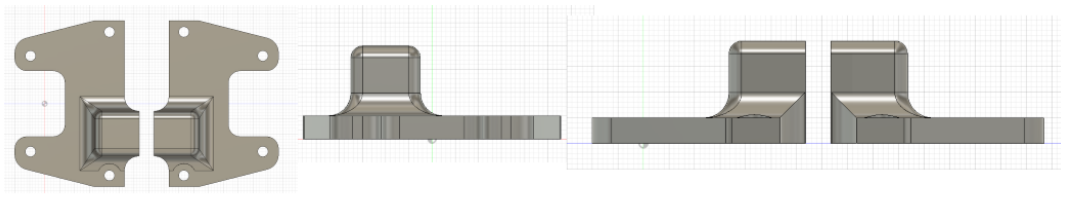

Figure 13: Gripper - Top view (left), Side view (middle), Front view (right)

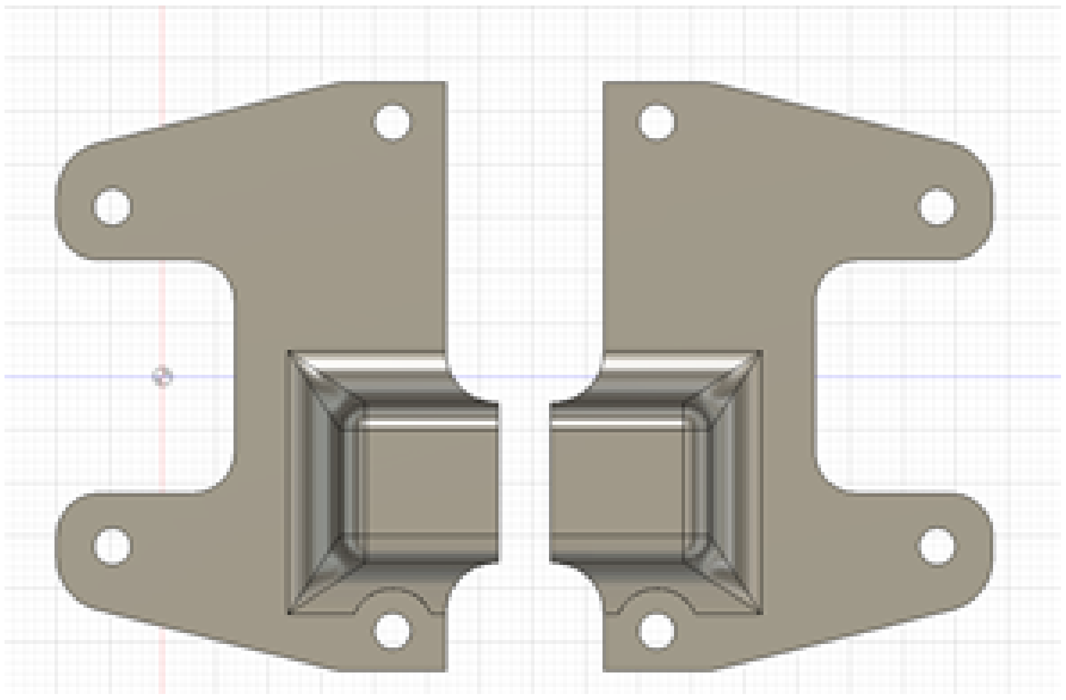

Figure 14: Top View

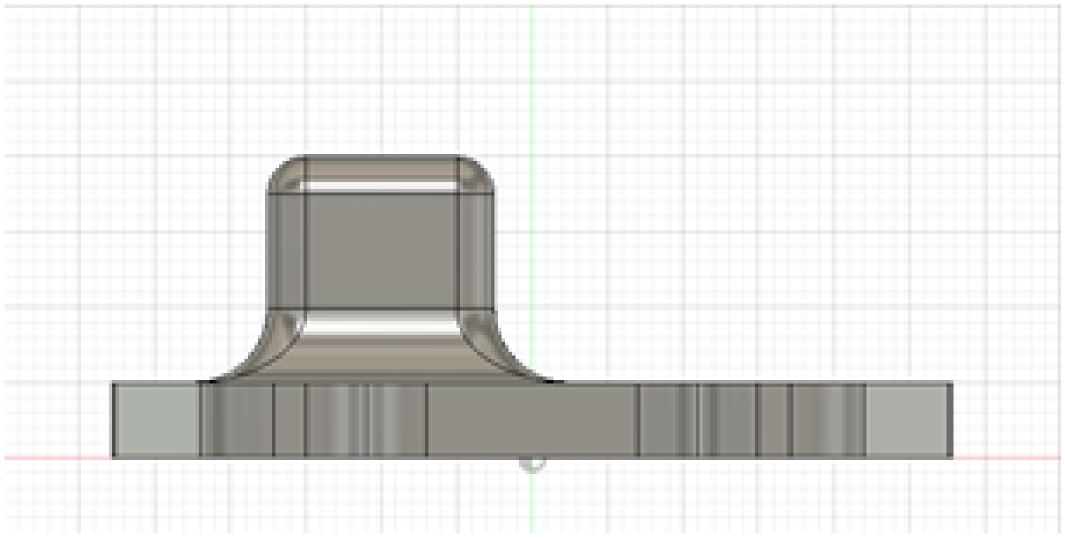

Figure 15: Front View

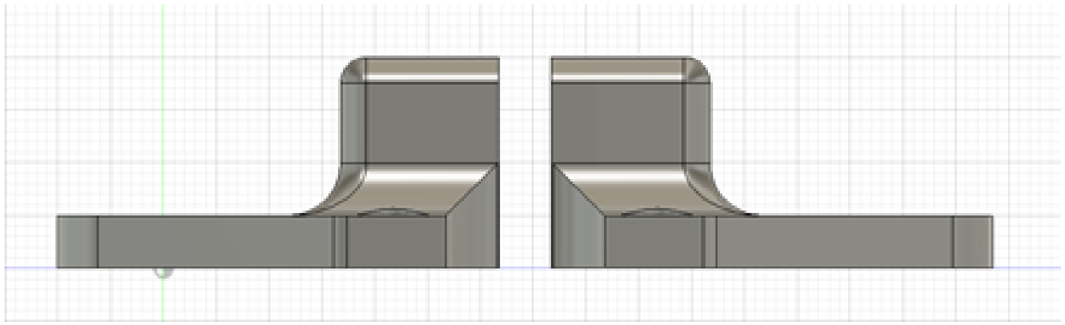

Figure 16: Side View

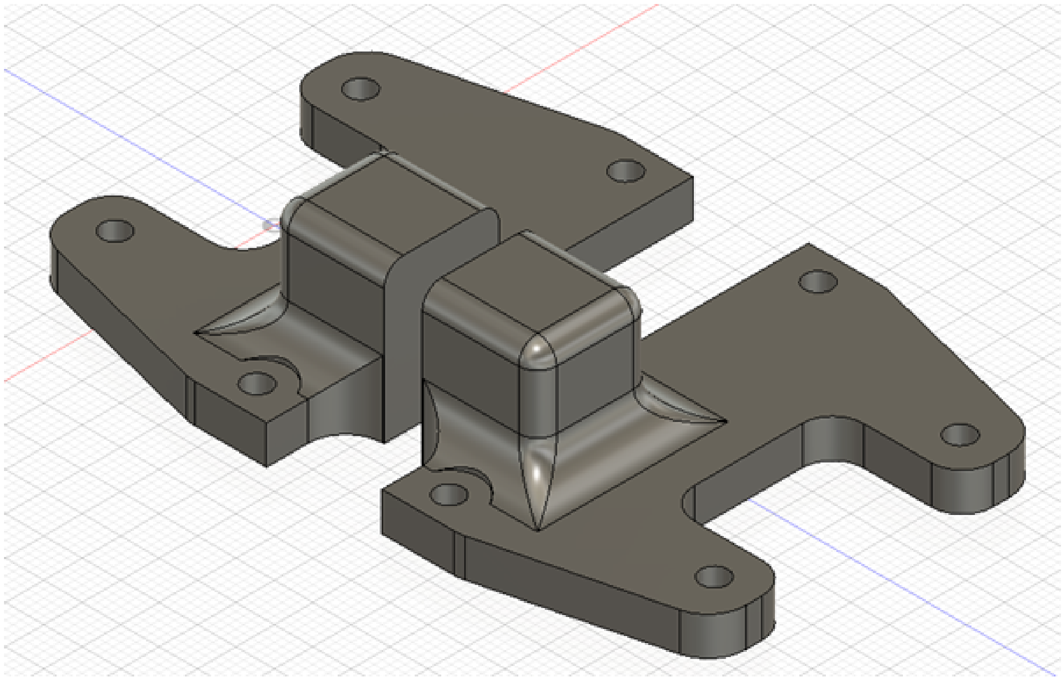
Figure 17: 3D View

Non-slip matting, cut to size, is attached to the edges of the jaws with double-sided tape to increase the grip of the gripper.

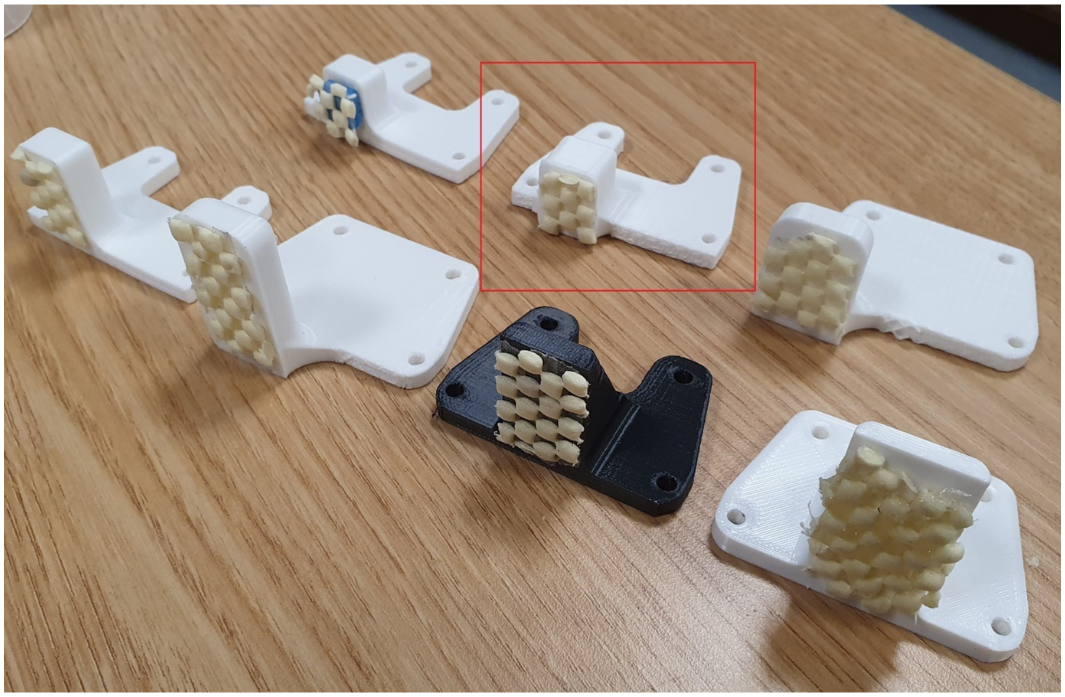
Figure 18: 3D Printed Models

Several iterations of the 3D printed grippers are shown, illustrating the construction methodology. The gripper with the red rectangle is the final one used.

C.2 Task 3 Gripper

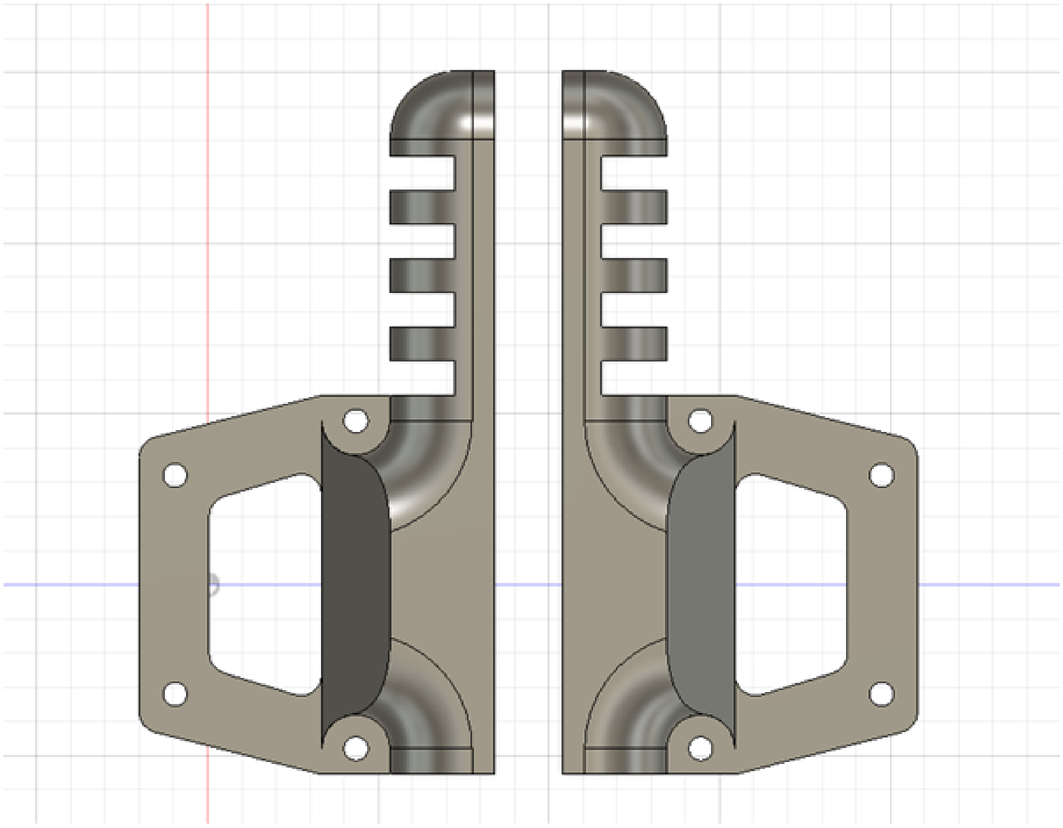

Figure 19: Top View

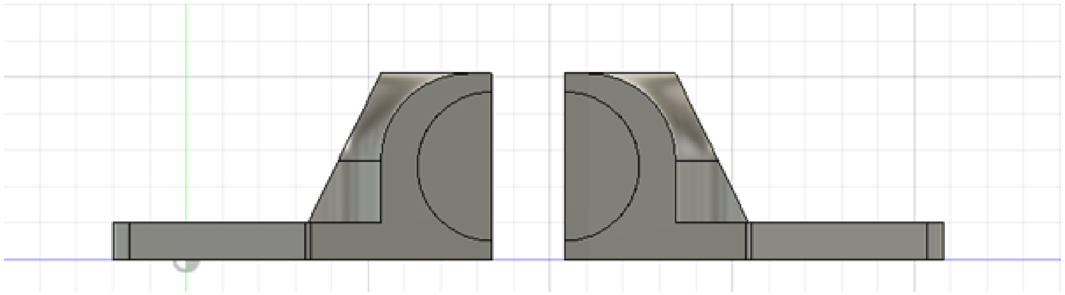

Figure 20: Front View

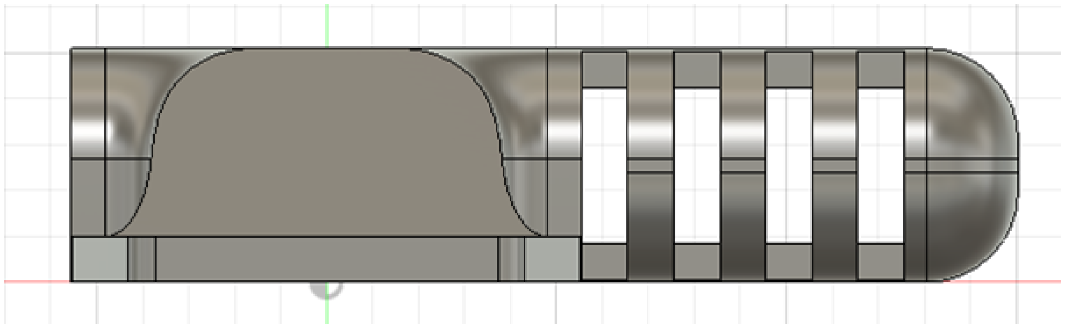
Figure 21: Side View

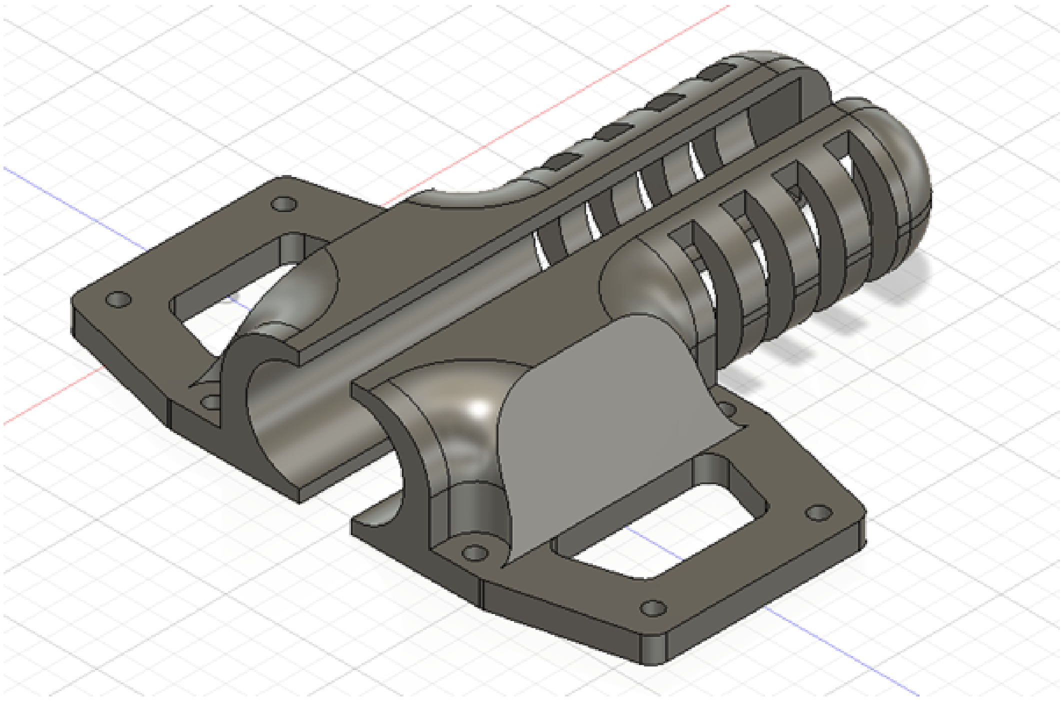
Figure 22: 3D View

As with the Task 2 gripper, non-slip matting is applied to the interior of the gripper to better conform to the shape of the marker.

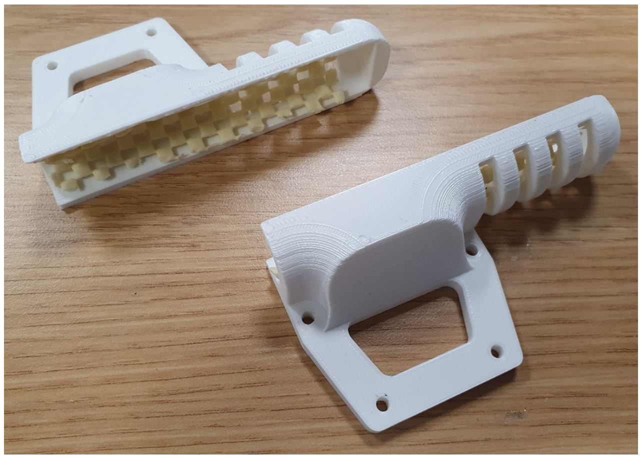

Figure 23: 3D Print

C.3 Task 3 Pen Holder

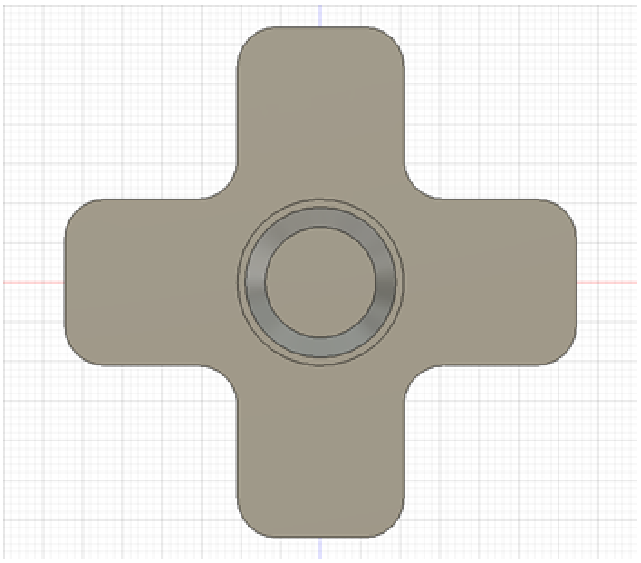

Figure 24: Top View

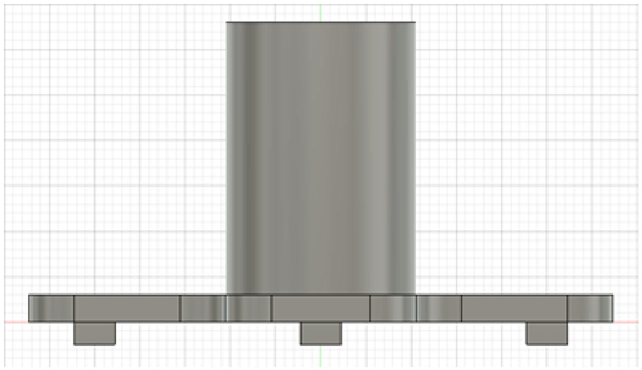

Figure 25: Front View

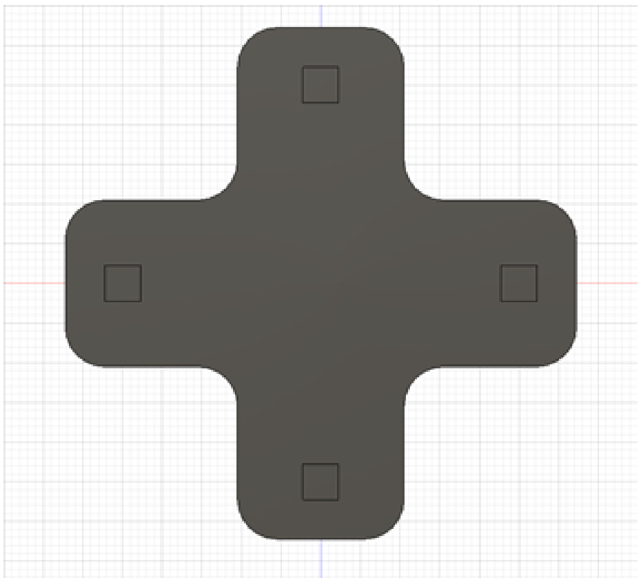
Figure 26: Side View

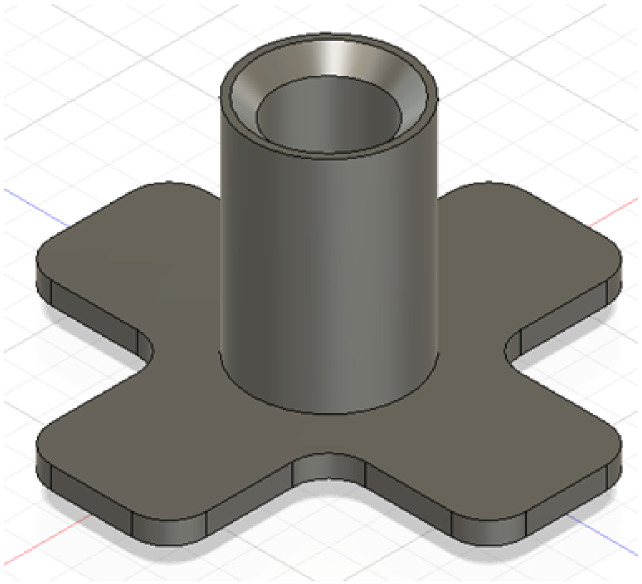
Figure 27: 3D View

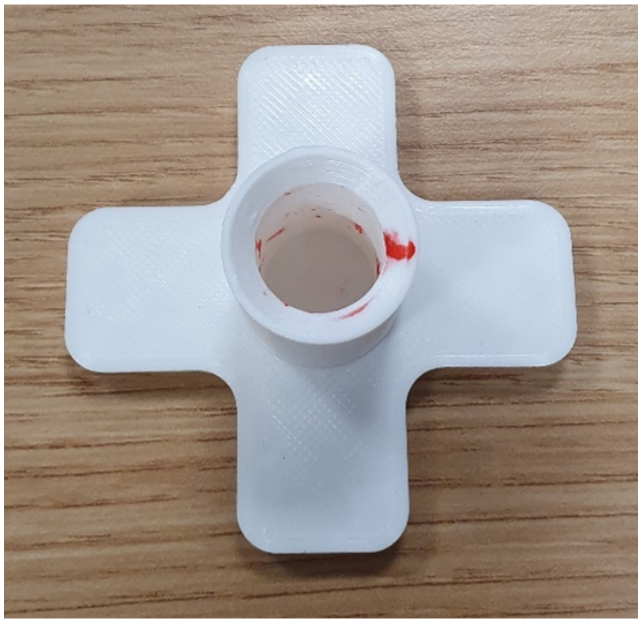
Figure 28: 3D Print

C.4 Task 4 Gripper

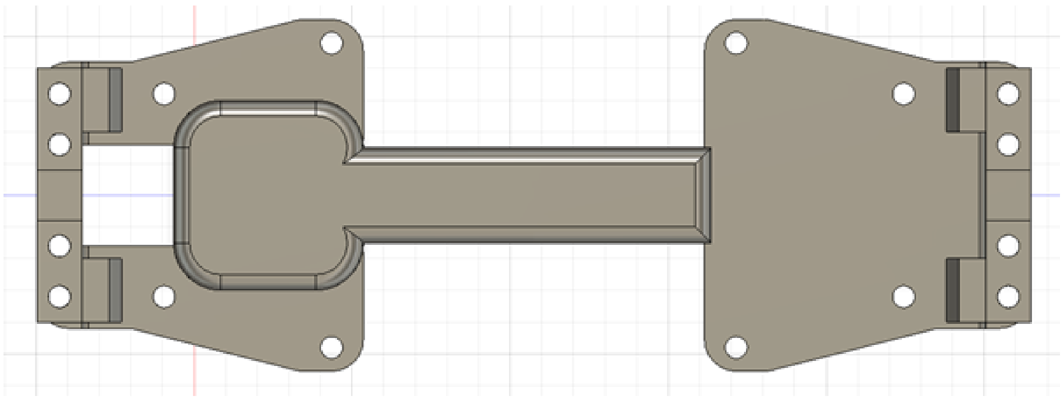
Figure 29: Top View

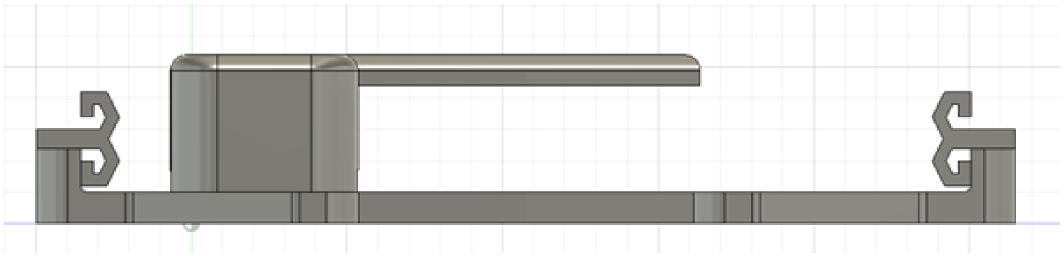
Figure 30: Side View

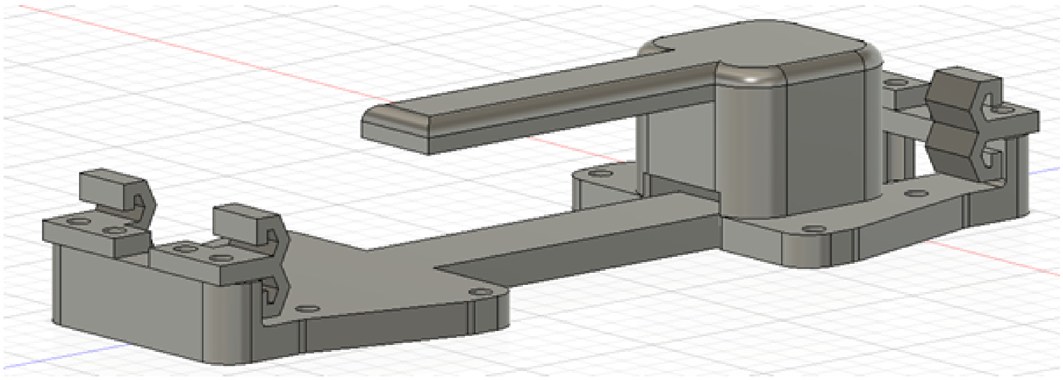
Figure 31: 3D View

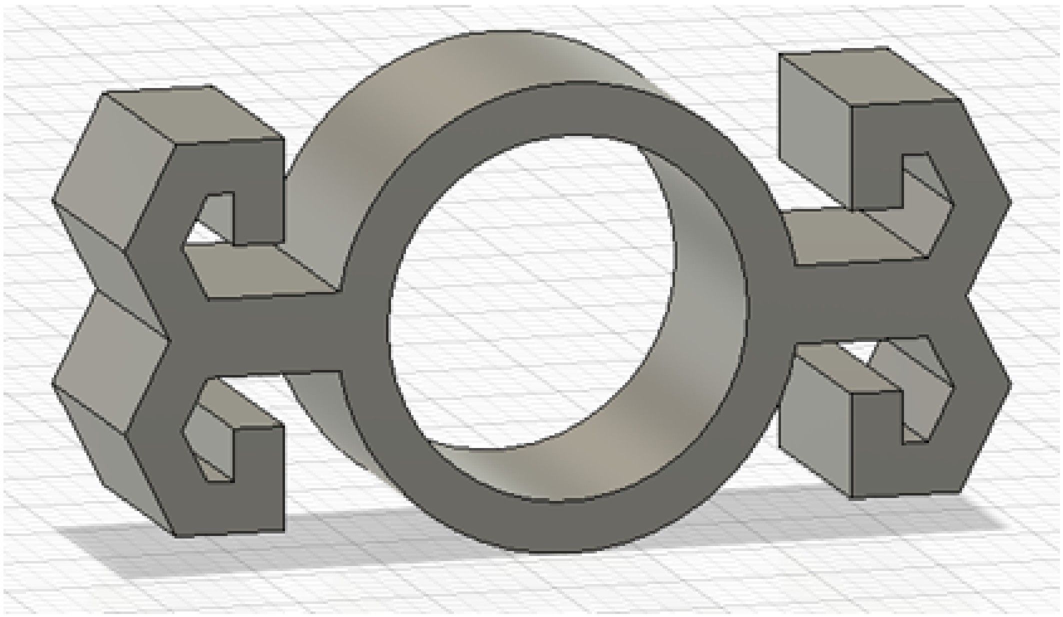
Figure 32: Buzz Wire Holder

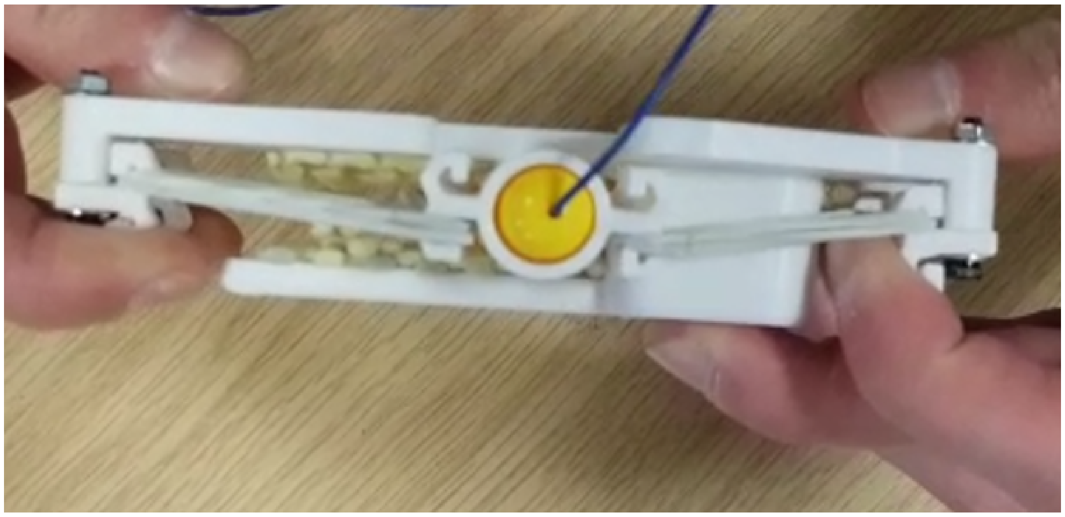

Figure 33: 3D Print Top View

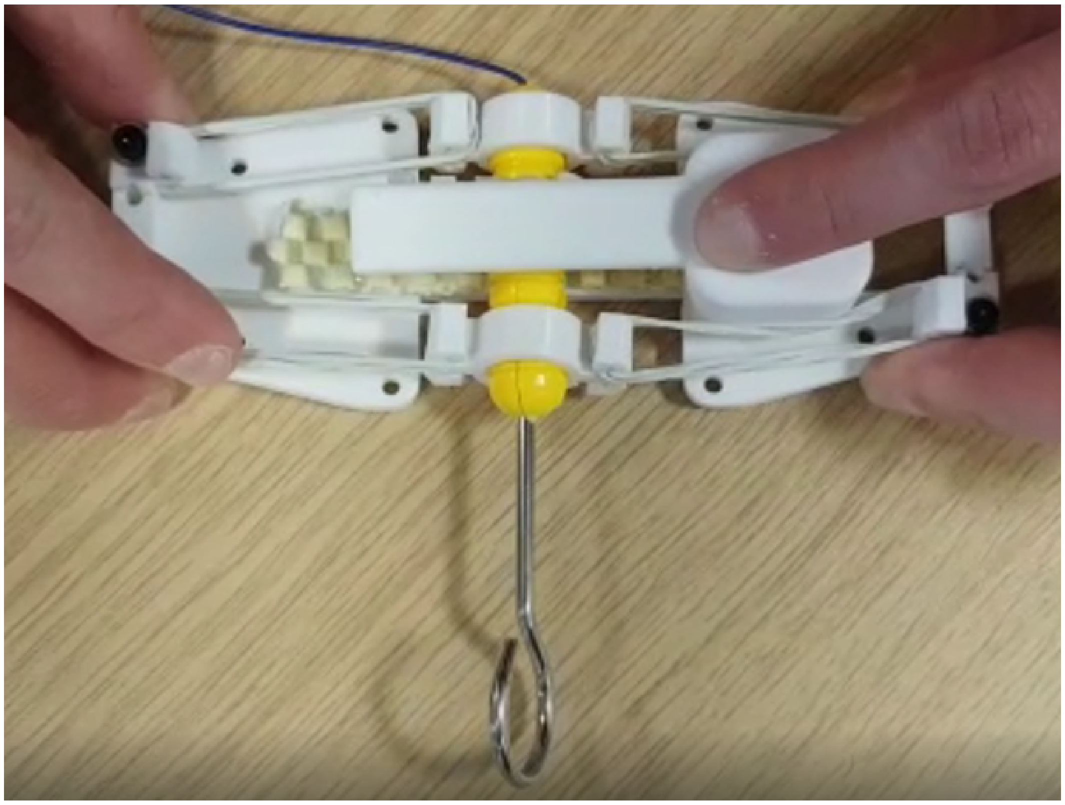

Figure 34: 3D Print Side View

C.5 Task 4 Buzzer Game

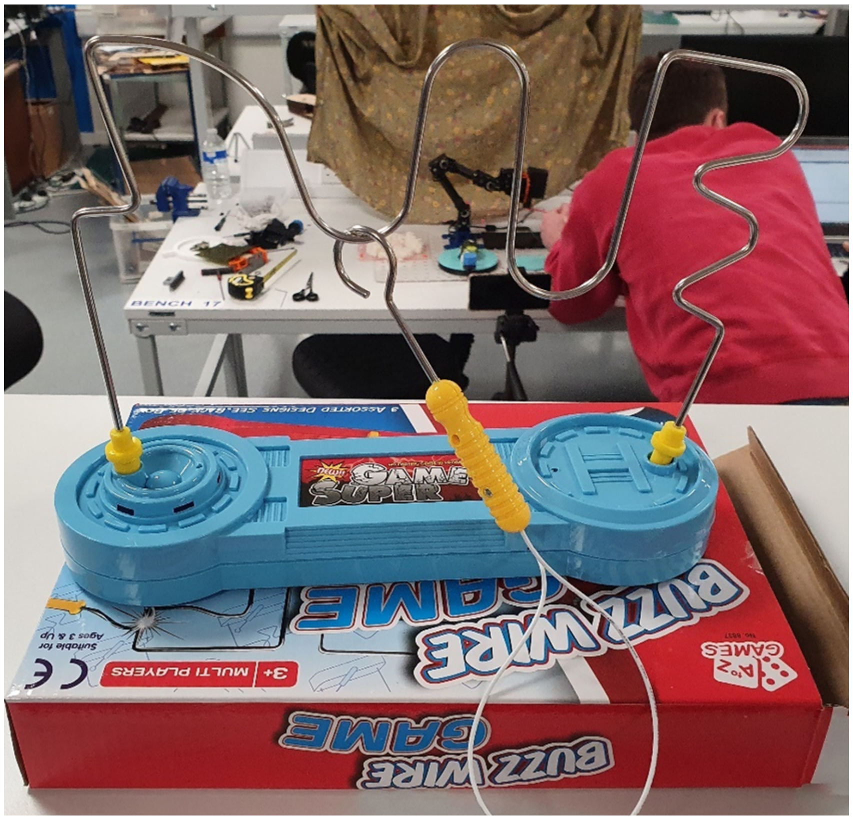

Figure 35: Original buzz wire game

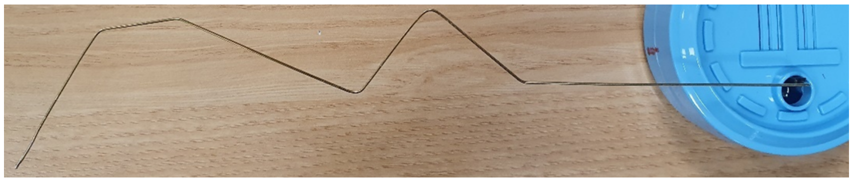

Figure 36: Modified buzz wire game