Imperial College London

EEE Department

Robotic Manipulation Report

Matilde Piccoli
Maximus Wickham
Bernard Benz

March 2022

Contents

1 Task 1

1.1 Forward and Inverse Kinematics

In order to build a kinematic model of the robotic arm, it was necessary to start by finding the DH parameters for each of the joint linkages and construct the DH transform. When deciding upon the parameters for the transform, the following rules were used taken from the John Craig textbook.

- a_i = the distance from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ measured along $\hat{X}_i$
- α_i = the angle from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ measured about $\hat{X}_i$
- d_i = the distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ measured along $\hat{Z}_i$
- θ_i = the angle from $\hat{X}_{i-1}$ to $\hat{X}_i$ measured about $\hat{Z}_i$

Using these rules, table 1 was constructed for the robotic arm with the linkages and axis as labelled in the diagram of figure 1. It was decided that the direction the robot is facing should be the positive x direction. This choice was not necessary as, since the first linkage has no arm length, the x and y direction could easily have been placed in any frame. However, this axis definition made the coordinate space line up with the physical grid in front of the robot. The second coordinate frame was chosen to give a value of α_1 of 90 instead of -90, again this choice was arbitrary and only chosen for the sake of cleanliness. The z axis for all the subsequent linkages was chosen to be in the same direction as the second linkage as this simplifies the overall transform as opposed to having z axis in opposite directions.

Linkage	a	α	d	θ
-1	0	0	0	0
0	0	90	D_0	θ_0
1	A_1	0	0	θ_1
2	A_2	0	0	θ_2
3	A_3	0	0	θ_3
4	0	0	0	0

Table 1: DH parameters

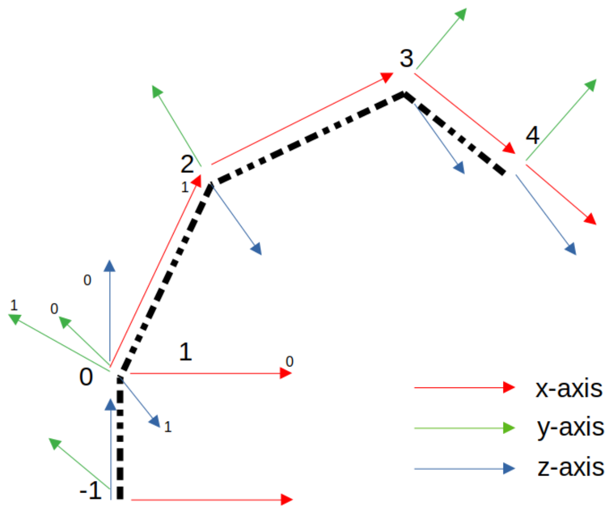

Figure 1: Axis of each frame

A final linkage is added to the robot with values of 0 for all DH parameters. This is necessary as the matrix used for the DH transform uses a value of a_{i-1} in the matrix for a given frame, meaning an extra frame is needed in order for a_3 to be included. Another frame with 0 values for DH parameters is placed at the base of the robot. This is required since d_i is measured along $\hat{Z}_i$ from $\hat{X}_{i-1}$, and therefore a frame is needed before the first linkage to have a reference

to measure this d value from. This first frame does not require a DH matrix, as the matrix formed would just be an identity matrix so creates no change. The overall transform we are trying to find is from the -1 frame to this final frame.
Once these parameters had been decided, a matrix could be constructed for each of the linkages by using the general matrix given in the lectures. The matrices are shown in the appendix in section 1.1. Having constructed the individual matrices the DH transform needed to calculate the position and orientation of each linkage relative to the first, using the matrix multiplication shown in the appendix in section 1.2.
The final DH transform gives us the orientation and position of the end of the robot gripper relative to the base of the robot in the coordinate frame of the first joint. If we consider a general rotation around each of the three axis, we can interpret better the rotation part of this transform; the relation between a general rotation matrix and ours is shown in the appendix in section 1.3. Since we know that the rotation around the z axis is the α in the equations (as this is the only joint that can rotate in this axis), we can see from the identity that β, i.e. the final gripper angle, is equal to $-(\theta_1 + \theta_2 + \theta_3)$. This means that in order to choose a final wrist angle we must fix the value of $(\theta_1 + \theta_2 + \theta_3)$.
The equations describing the relations between the joint angles and the final position of the gripper are also given by the DH transform.

$$x = C0(C123A_3 + C12A_2 + C1A_1) \tag{1}$$
$$y = S0(C123A_3 + C12A_2 + C1A_1) \tag{2}$$
$$z = S123A_3 + S12A_2 + S1A_1 + D_1 \tag{3}$$

These equations make solving the forward kinematics for the gripper positions trivial as the joint angles can simply be plugged into these equations. However, in order to find the joint angles from the gripper position, we must first perform some rearranging. Since there are 4 unknowns on the right hand side of the equations, we must first fix a value of $(\theta_1 + \theta_2 + \theta_3)$ corresponding to the final gripper angle. The rearrangement of these equation is shown in the appendix in section 1.4 and results in the following identities:

$$a = (x/C0) - C123A_3 \tag{4}$$
$$b = z - S123A_3 - D_1 \tag{5}$$
$$\theta_0 = tan^{-1}(\frac{y}{x}) \tag{6}$$
$$\theta_2 = \pm cos^{-1}(\frac{1}{2A_1A_2}((a^2 + b^2) - (A_1^2 + A_2^2))) \tag{7}$$
$$\theta_1 = tan^{-1}(b/a) - tan^{-1}(\pm A_2\sqrt{1 - cos(\theta_2)^2}/(A_1 + A_2cos(\theta_2))) \tag{8}$$
$$\theta_3 = (\theta_1 + \theta_2 + \theta_3) - \theta_1 - \theta_2 \tag{9}$$

With these equations we can compute the necessary joint angles to reach an end positions. The choice of the positive or negative solution to the inverse cosine used for θ_2 determines whether the up or down elbow solution is given.

1.2 Simulation

With both the forward and inverse kinematics having been solved, it was possible to simulate the robot graphically in Matlab. In order to do this, an end position of the gripper had to be specified, as well as the end gripper angle. This data was processed by a function which found angles needed to reach this position using the equations found previously and was then able to use these angles and the DH-transforms to find the position of each linkage. Once the position of each linkage had been obtained, the coordinates could be plotted in 3D to visualise the robot and continuously changing the end coordinates of the plot allowed for animations, such as the robot tracing a square.
In order to visualise the axis at each linkage, the rotation matrix for each of the DH-Transforms was taken and this was applied to the end coordinates of the axis of the base frame. The axis positions then had the offset of the position they belonged to added before being plotted. The equation used to give the end coordinates of axis of length 5 is as follows.

$$A = R\begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix} + P$$

Where R is the rotation matrix for the given linkage and P is the position of the linkage
A is the end coordinates of the axis for a given linkage

Figure 2 shows the robot outline in 3D, which was a useful visualisation to check that the inverse kinematics were working as expected. In this image, the end position of the gripper was set to $(15, -2.5, 15)$ with a 0 radiant gripper

angle. Animations were created demonstrating the simulation tracing the outline of a square in each of the three frames, as shown in the accompanying video presentation. In order to generate this plot, either the linkage angles or the end position can be supplied allowing testing of both forward and inverse kinematics. The kinematics were also checked by ensuring that the angles supplied by the inverse kinematics returned the original position when passed through the forward kinematics.
When creating the animation of the robot for each side of the square, one of the end coordinates was given values within a linear space of length 5, rotating which axis was changed 3 times to give the 4 sides of the square. Images showing the final square drawn in each video are shown in the appendix in figures 3,4 and 5. Figure 2 shows a square drawn by the simulation in 3D. The video presentation demonstrates the testing of the forward kinematics through the adjustment of the linkage angles.

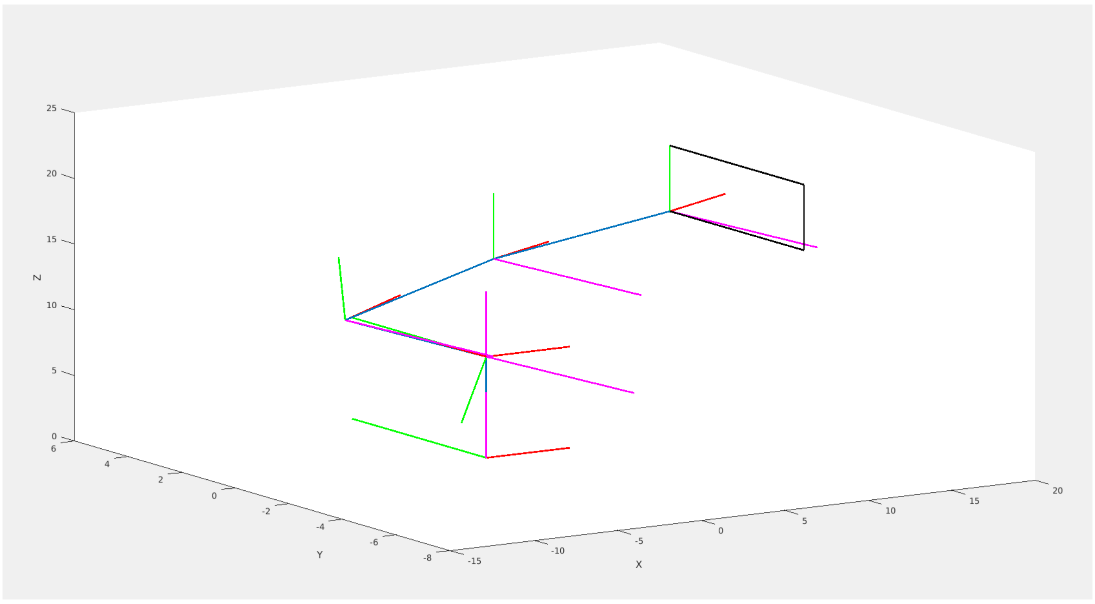

Figure 2: 3D square drawing by the robot simulation, red for x axis, green for y axis and purple for z axis

2 Task 2

2.1 Code Structure

For the purpose of maintainable and easy-to-debug code, the task of moving and flipping cubes was implemented using multiple nested classes:

- KinematicModel: this class's aim is to initialise and call the functions implementing the kinematics of the movements, as explained in task 1; the correctness of the kinematic was tested extensively both with the simulator and the actual robots to assure the correct functionality also with possible edge cases (e.g. when the coordinates are zeroed); the actual lengths of the robot joints were first measured and progressively tune to reach the highest accuracy in positioning the tip of the gripper in the given position;

- RobotModel: this class calls an instance of KinematicModel and uses its functionality to transform positions in the xyz space into servo values; in this class, the values of the joints' angles are set and angles offset were added and tested to account for the dis-alignment of the actual physical joints. When converting from radians used in the kinematic model to servo values the offsets needed for each of the servos and the direction scale factor must be found. This is necessary to ensure that a rotation for a linkage in the model corresponds to the correct rotation of the corresponding servo and that the servo moves in the correct direction when the rotation is changed. Between linkage 1 and 2 there is an angle offset caused by linkage 2 not lying on the x-axis of linkage 1 and the negative of the offset must also then be applied to linkage 2. The offsets and direction scales for each servo that were used are shown in table 2.

Servo	Angle Offset	Direction Scale
0	0	1
1	$-0.18356 - \pi/2$	-1
2	$0.18356 + \pi/2$	-1
3	π	-1

Table 2: Offsets and scaling for servo angles

- RobotController: this class calls an instance of RobotModel and is responsible for the actual communication with the servos; it initialises the robot parameters, such as baud rate, torque, speed and acceleration of each joint, and implements a low level function to move between two positions;
- CubeController: at this point, a different class for each tasks was implemented, calling an instance of RobotController; for task 2, CubeController contains all the functions responsible for moving, flipping and stacking cubes; the cube dimension and grid of possible positions are defined, as well as the speed and trajectory computation for each type of movement; here, a main function calls all the different type of movements so that in the test script only this function has to be called after opening the port and, once position, flipping angle and number of the stack are input, the robot is ready to implement the whole sequence of actions on its own, increasing the simplicity and efficiency of the code.

2.2 Moving algorithms

A full overview on the functions implemented is provided below:

- Moving: for this task, two moving modes have been implemented, a constant-speed mode and a time-based one, which set the speed depending on the distance; a trajectory function enables to use different degrees of interpolation (linear, cubic etc.) to pre-calculate the trajectory of the movement and tune the speed depending on the proximity to the final/initial position; this makes the the movements smoother and more precise, while not having to compromise too much when it comes to speed.
- Flipping: to perform the flipping of cube we have to first compute whether, given the physical constraints, the robot can grab the cube horizontally and/or vertically in a given position; once that is defined for each position needed, the robot can decide the most convenient type of flip to perform: flip whilst moving to the final position, flip on the spot, or move to a position in which both orientation of the gripper (and therefore flipping action) are possible, then flip and move back. Then, the direction of the flip is computed depending on the angle needed and the initial/final gripper angles are set; finally, the action is called and the sequence of movements are computed one after the other.
- Stacking: for this task, the above functions were maintained and an additional parameter setting the number of cubes stacked is passed into the main function so that the algorithm can be performed as normal, but the height at which the cube are picked/moved/placed changes accordingly.
- Others: more low level functions were implemented for testing purposes and so that the robot is still able to perform elementary actions without much computation. Some examples are: open and close gripper, go up or down on the z axis, set speed.

The final algorithm for moving, flipping and stacking cubes is as follows: compute the sequence of orientations of the gripper required by the possible flips input; open the gripper; go up (elevated position with respect to the cubes so that the movement do not interfere with the cubes/cubes holders, dependent on the height of possible stacks); move to the position of the cube; grab the cube and, if necessary, perform flipping (on the spot, while moving, or using an external position,as computed before); move the cube to the final position (if not different to the initial one)(final release height dependent on the number of cubes in the stack); open the gripper; move away.

2.3 Testing

All the possible combinations of movements and positions were thoroughly tested to make sure the code did not break with any edge cases; after the functional testing, parameters such as error margin, trajectory type and speed were extensively tuned to achieve an optimal trade-off between speed and smoothness of the movements, while maintaining the required precision to perform any action.

2.4 Performance evaluation

The main focus was to maximise the precision of movements first, and then, depending on the task, a higher smoothness of the movements or higher speed was prioritised, making sure the accuracy was maintained.

- Accuracy: all the assumption on the robot and object's measures were checked against and its parameters adjusted to take into account any error; testing has been performed on the accuracy, and the moving algorithms where chosen accordingly (time-base vs speed-based mode); to further improve the precision, a PID has been implemented to adjust the final position depending on the error perceived in the movement by the robot; the consistency of final positions between different runs of the same movement was successfully checked and the final testing lead to an error margin below 3mm.
- Smoothness: this was mainly improved by adjusting the n of points used for the trajectory, the speed and the acceleration; the code was also check to make sure the computation would not slow down the movements and, for this reason, it was also decided to compute the sequence of positions first, and then move.
- Speed: the limit on the speed were set by the trade-off with the smoothness and accuracy; it was noticed that if the speed was too high, the robot did not have the time to perceive the error in position and would not adjust it; in the end it was decided to use a variable-speed setting for actions that have different range/ require different precision and each of them was manually tuned through testing.

3 Task 3

In order to complete this task three different problems had to be solved: how to pick up and secure the pen, how to draw lines, and how to draw circles.

3.1 Grabbing the pen

The design of our custom pen gripper had to meet several requirements. Firstly, it was important that the pen could be securely held once the gripper was closed without being able to rotate or wriggle. To ensure this our gripper was shaped so as to have as much contact with the pen itself as possible wrapping around its length with two semi circular extrusions. Secondly, we wanted to ensure that the robot was able to draw at any point on the plate provided, which meant being able to reach any point above the plate at a horizontal gripper angle and at the correct height. By holding the pen as close to the gripper mechanism as possible the robot was able to hold the pen at a horizontal level closer too its base solving this problem.
The position the pen was picked up from was extremely close to the base of the robot, this was to ensure the distance from the robot to the pen was as small as possible, meaning slight errors in the servo angles or simple mechanical play in the servos resulted in a smaller distance error due to the smaller radius this angle error was working across. This meant the pen could be picked up as accurately as possible.

3.2 Drawing Lines

When instructing the robot to draw lines a function was used to take the start and end position of the line to be drawn, as well as whether the robot should start and end in an up or down pen position. The trajectory for drawing the lines was then calculated using interpolation, with the power used for the interpolation tuned to give the best results accompanied with a time based servo control to ensure smooth lines being drawn. To minimise the delay between instructions, the inverse kinematics for the entire line was calculated before the start of the drawing, and then the raw servo values were sent directly when performing the drawing to maximise smoothness. When reaching the first position of the line, error correction was performed automatically. This was due to the fact that, in time-based setting for the servo speed, it was found that often the servos would stop moving before reaching the exact position required. For this reason, a strategy similar to a PID has been implemented: a position greater than the desired final position would be sent, with this extra distance getting progressively smaller to encourage the servo to reach a greater degree of precision. To reduce the need for as much vertical accuracy, our gripper was designed allowing the pen to slide slightly upwards, but then be pushed back down if force was removed. This added play allowed the robot to push the pen harder against the plate, allowing to make contact even if the vertical position of the gripper was slightly too high whilst drawing.

3.3 Drawing Circles

When drawing circles, a similar method to the lines was used, but in this case a consistent distance between points was used when splitting up the arc into sub points, as a constant speed was preferable whilst changing direction. A number of points along the circle's perimeter dependent on the arcs radius and angle are computed and the inverse kinematics solved for these points before instructing the robot to move to the required servo positions. It was found that due to a large amount of play in the first servo and this servo also having the most effect on the end position the circle drawn would often look slightly ovular, squashed perpendicular to the line between the robot base and the circle centre. To account for this slight error the circle points were adjusted before solving the inverse kinematics. To do this the following equation was applied to each point.

$$P = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} C \text{ where } C \text{ is the circle centre}$$
$$p = \text{ position of point on circle to be adjusted}$$
$$d = \text{ distance from } p \text{ to line from } C \text{ to robot base}$$
$$p_{new} = P * d * A \text{ where } A \text{ is a constant}$$

This method essentially moves each point a small amount further away from the line between the circle centre and the point itself to account for the error due to the play in the robot base joint.

4 Task 4

4.1 Our idea

For the open-ended task, we wanted to go beyond the basic functionality of the robot demonstrated with the previously; we identified the following elements to explore within the new task:

- less well-determined environment: we wanted a task that could simulate a real-world problem, with many variables possibly changing between different runs, pushing us to design the algorithm and the gripper in a more robust way
- more versatile gripper: the idea is to have a gripper that, depending on its orientation, can manipulate objects with different shapes and texture in different ways
- extra degree of freedom: we found that not be able approach an object from different orientations was a big limitation of the robot and, given that it was not allowed to add an extra joint, we still looked for a way to rotate the object instead

With these goals in mind, we came up with the idea of implementing a sushi-maker robot, a task that would require as much precision as versatility of the gripper.

4.2 Our Implementation

The sushi-maker consisted of two main parts:

- A multipurpose gripper that can work both as pliers to pick ingredients like salmon and seaweed, and a mould to format and move the rice; on top of this, the gripper is also able to hold the knob of the rotating plate or manipulate other objects; all the functionality of the gripper were tested extensively to make sure that the design is solid enough to enabling a good performance with different conditions of the objects (many variables changed from different runs, e.g the stickiness of the rice, the precise size of the seaweed etc.)
- A serving plate spinning on a support; a rotation can be performed by the gripper by dragging a knob, so that the ingredients can be placed and manipulated in any orientation. This feature requires that the moving algorithm of the robot is precise enough to perform a regular circle to spin the knob without shifting the plate; to counteract the possible errors in the movements (precision within 5mm), the knob was design so that it can slide a few centimetres in all three axis: this system was thoroughly tested and resulted in great precision in the rotation (within 2 degrees), well beyond the requirements for making a form of sushi with consistency. An equation had to be formed to work out how much to rotate the plate to allow the robot to pick up an object placed with an orientation parallel to the robots x direction such that after the rotation its orientation would be parallel to the radius between it and the robot base. The following equation was used to find the angle to rotate the plate θ and the new position of the object on the plate.

$$p = \text{ position of object relative to the centre of the plate}$$
$$\text{plate must have a } y \text{ position of } 0$$

$$d = \text{ distance from robot to plate centre}$$
$$\theta = tan^{-1}(|p|/d) * sign(p[1])$$
$$p_{new} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} p$$

The sushi-making task consists in the following algorithm:

- Pick and place rice: mould-gripper is inserted in the rice, it closes to compresses the rice in a cuboid shape, it lifts it and move it to the rotating place; various density and textures of rice were tested and all worked well enough to stick to the gripper whilst its moving and also maintain the shape when the gripper is opening; to ensure this, the speed of the robot was tuned and different functions were implemented for each action so to set the optimal speed and trajectory algorithm for each action individually.
- Pick and place salmon: the gripper rotates to use the pliers side of it to pick the slice of salmon and place it on the rice form; the strength of the gripper hold was tuned to be solid enough to give a margin of error in the salmon's dimension and yet not damage the slice.
- Rotate the plate: a 90° rotation of the plate (with the rice and salmon on) is performed at this point so that the seaweed can be placed perpendicularly to the sushi length; the knob at the extremity of the place is grabbed by the gripper, which by performing the required circular trajectory, is therefore also spinning the plate to the required angle.
- Pick and place the seaweed: the same actions performed on the salmon are done here for the seaweed, although further tuning was required given the difference in size, weight and texture of the ingredients.
- Rotate the plate back to the original position: this is performed both to demonstrate the precision of the rotation (compared against the initial position of the plate) and also so that the sushi is in the right orientation to be formatted and picked up if needed.
- Wrap the seaweed around the sushi: the gripper grabs the rice with the salmon and seaweed on it, and by doing so the seaweed is compressed on the rice and the sushi takes its final shape; this action was also thoroughly tested to make sure the ingredients are not squished and broken in the process.
- Serve the sushi: the sushi is now pretty and ready to be eaten! If needed, the gripper can grab the whole piece and place it on a different service place and restart the whole process to make a new piece.

Appendix

1 Kinematics

1.1 DH Matrices

$$\mathrm{M}_0 = \begin{bmatrix} C0 & -S0 & 0 & 0 \\ S0 & C0 & 0 & 0 \\ 0 & 0 & 1 & D_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} C1 & -S1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} C2 & -S2 & 0 & A_1 \\ S2 & C2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} C3 & -S3 & 0 & A_2 \\ S3 & C3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & A_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2 DH Transforms

$$\mathrm{M}_0 M_1 = \begin{bmatrix} C0C1 & -C0S1 & S0 & 0 \\ S0C1 & -S0S1 & -C0 & 0 \\ S1 & C1 & 0 & D_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_0 M_1 M_2 = \begin{bmatrix} C0C12 & -C0S12 & S0 & C0C1A_1) \\ S0C12 & -S0S12 & -C0 & S0C1A_1) \\ S12 & C12 & 0 & S1A_1 + D_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_0 M_1 M_2 M_3 = \begin{bmatrix} C0C123 & -C0S123 & S0 & C0(C12A_2 + C1A_1) \\ S0C123 & -S0S123 & -C0 & S0(C12A_2 + C1A_1) \\ S123 & C123 & 0 & S12A_2 + S1A_1 + D_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_0 M_1 M_2 M_3 M_4 = \begin{bmatrix} C0C123 & -C0S123 & S0 & C0(C123A_3 + C12A_2 + C1A_1) \\ S0C123 & -S0S123 & -C0 & S0(C123A_3 + C12A_2 + C1A_1) \\ S123 & C123 & 0 & S123A_3 + S12A_2 + S1A_1 + D_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.3 Rotation Matrix

R$_z(\alpha)R_y(\beta)R_x(\gamma)$
$= \begin{bmatrix} cos(\alpha)cos(\beta) & cos(\alpha)sin(\beta)sin(\gamma) - sin(\alpha)cos(\gamma) & cos(\alpha)sin(\beta)sin(\gamma) + sin(\alpha)cos(\gamma) \\ sin(\alpha)cos(\beta) & sin(\alpha)sin(\beta)sin(\gamma) + cos(\alpha)cos(\gamma) & sin(\alpha)sin(\beta)sin(\gamma) - cos(\alpha)sin(\gamma) \\ -sin(\beta) & cos(\beta)sin(\gamma) & cos(\beta)cos(\gamma) \end{bmatrix}$
$= \begin{bmatrix} C0C123 & -C0S123 & S0 \\ S0C123 & -S0S123 & -C0 \\ S123 & C123 & 0 \end{bmatrix}$

1.4 Inverse Kinematics Solution

cos$(\theta + \phi) = cos(\theta)cos(\phi) - sin(\theta)sin(\phi)$
$sin(\theta + \phi) = cos(\theta)sin(\phi) + cos(\phi)sin(\theta)$
$sin(\theta)^2 + cos(\theta)^2 = 1$
The above trig identities are assumed to rearrange the following equations
$x = C0(C123A_3 + C12A_2 + C1A_1)$
$y = S0(C123A_3 + C12A_2 + C1A_1)$
$z = S123A_3 + S12A_2 + S1A_1 + D_1$
$\frac{y}{x} = S0/C0$
$\frac{y}{x} = tan(\theta_0)$
$\theta_0 = tan^{-1}(\frac{y}{x})$
$x = C0(C123A_3 + C12A_2 + C1A_1)$
$(x/C0) - C123A_3 = C12A_2 + C1A_1 = a$
Where a is known as $C0$ and $C123$ are known
$y = S0(C123A_3 + C12A_2 + C1A_1)$
$(y/S0) - C123A_3 = C12A_2 + C1_A1 = a$
This identity can be used if $C0 = 0$ to avoid division by 0
$z = S123A_3 + S12A_2 + S1A_1 + D_1$
$z - S123A_3 - D_1 = S12A_2 + S1A_1 = b$
Where b is known as $S123$ and D_1 are known
$b = S12A_2 + S1A_1$
$a = C12A_2 + C1A_1$
$a^2 = C12^2A_2^2 + C1^2A_1^2 + 2C1C12$
$b^2 = S12^2A_2^2 + S1^2A_1^2 + 2S1S12$
$a^2 + b^2 = A_1^2(C1^2 + S1^2) + A_2^2(C12^2 + S12^2) + 2A_1A_2(C1C12 + S1S12)$
$a^2 + b^2 = (A_1^2 + A_2^2) + 2A_1A_2(C1C12 + S1S12)$
$a^2 + b^2 = (A_1^2 + A_2^2) + 2A_1A_2C2$
$cos(\theta_2) = \frac{1}{2A_1A_2}((a^2 + b^2) - (A_1^2 + A_2^2))$
$\theta_2 = \pm cos^{-1}(\frac{1}{2A_1A_2}((a^2 + b^2) - (A_1^2 + A_2^2)))$
The sign chosen here decides whether the up or down elbow solution will be given
$M_1 = C2A_2 + A_1$
$M_2 = S2A_2$
Where M_1 and M_2 are known values at this stage as θ_2 is known
$a = M_1C1 - M_1S1$
$b = M_1S1 + M_2S1$
Create two new unknowns ϕ and P that have the following constraints
$\frac{sin(-\theta_1+\phi)}{cos(-\theta_1+\phi)} = \frac{M_2}{M_1}$
$Psin(-\theta_1 + \phi) = M_2$
$Pcos(-\theta_1 + \phi) = M_1$
$tan(-\theta_1 + \phi) = \frac{M_2}{M_1}$
$b = P(cos(-\theta_1 + \phi)sin(\theta_1) + sin(-\theta_1 + \phi)cos(\theta_1))$
Using the $sin(\theta + \phi)$ identity
$b = Psin(\phi)$
$a = P(cos(-\theta_1 + \phi)C1 - sin(-\theta_1 + \phi)S1)$
Using the $cos(\theta + \phi)$ identity
$a = Pcos(\phi)$
$tan(\phi) = \frac{b}{a}$

$\theta_1 = tan^{-1}(b/a) - tan^{-1}(M_2/M_1)$
$\theta_1 = tan^{-1}(b/a) - tan^{-1}(A_2 sin(\theta_2)/(A_1 + A_2 cos(\theta_2)))$
Use the pre-decided gripper angle $(\theta_1 + \theta_2 + \theta_3)$
$\theta_3 = (\theta_1 + \theta_2 + \theta_3) - \theta_1 - \theta_2$
$a = (x/C0) - C123A_3$
$b = z - S123A_3 - D_1$
$\theta_0 = tan^{-1}(\frac{y}{x})$
$\theta_2 = \pm cos^{-1}(\frac{1}{2A_1A_2}((a^2 + b^2) - (A_1^2 + A_2^2)))$
$\theta_1 = tan^{-1}(b/a) - tan^{-1}(\pm A_2\sqrt{1 - cos(\theta_2)^2}/(A_1 + A_2 cos(\theta_2)))$
$\theta_3 = (\theta_1 + \theta_2 + \theta_3) - \theta_1 - \theta_2$

2 Simulation

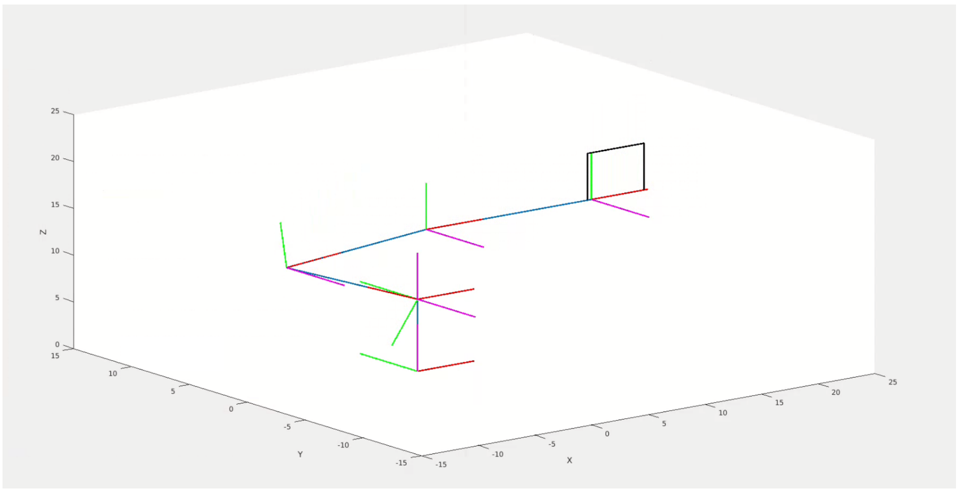

Figure 3: Simulation drawing a square in x,z plane, red for x axis, green for y axis and purple for z axis

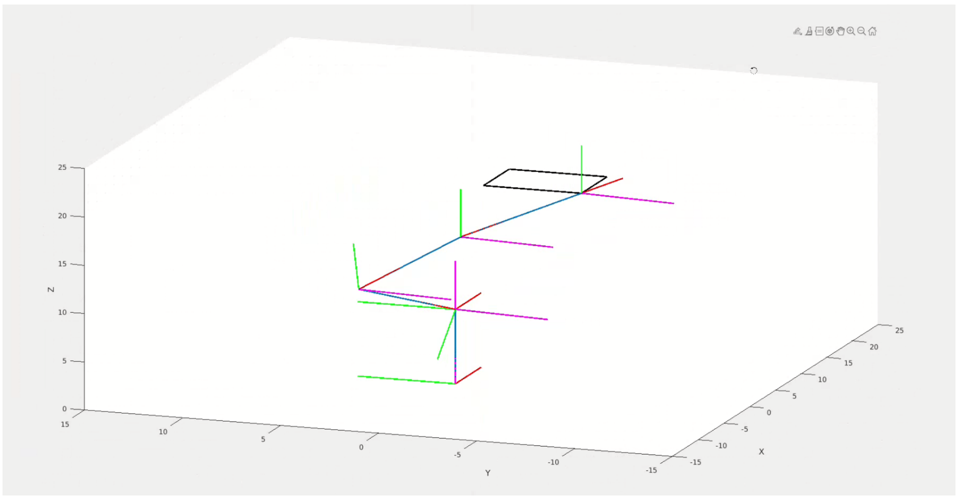

Figure 4: Simulation drawing a square in y,x plane, red for x axis, green for y axis and purple for z axis

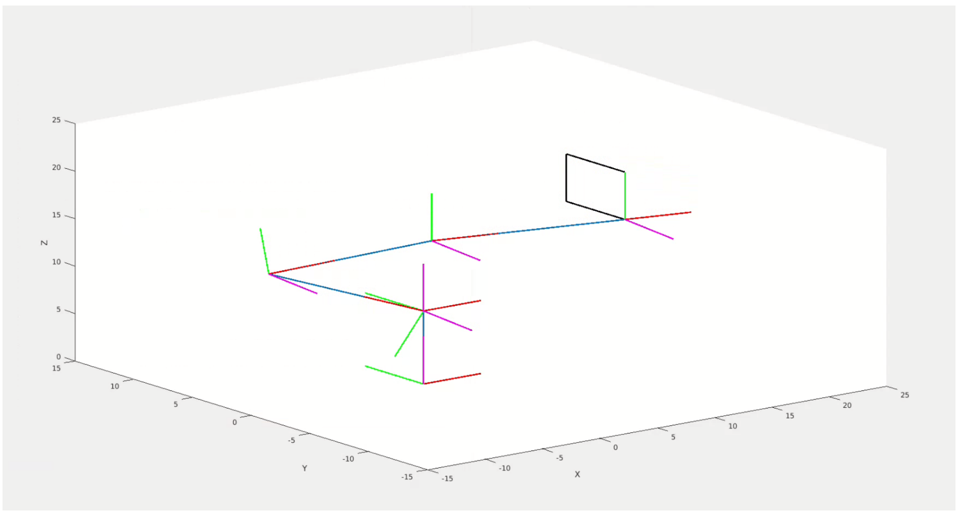

Figure 5: Simulation drawing a square in y,z plane, red for x axis, green for y axis and purple for z axis

3 CAD models

Multiple CAD models have being designed using SolidWork, both for the gripper and pen holder used in task 3 and the rotating plate and gripper of task 4. The successful design are shown below ():

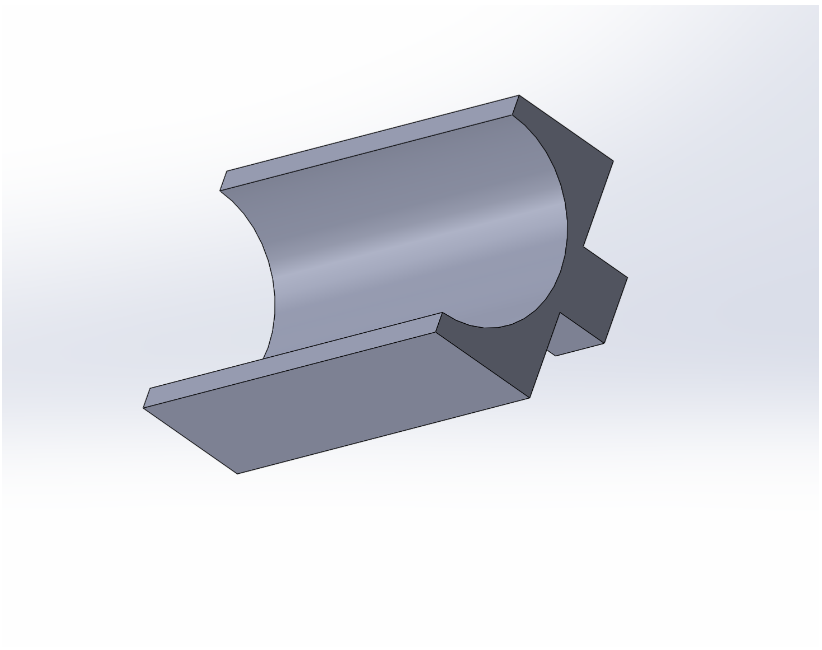

Figure 6: CAD model of the pen holder (task 3)

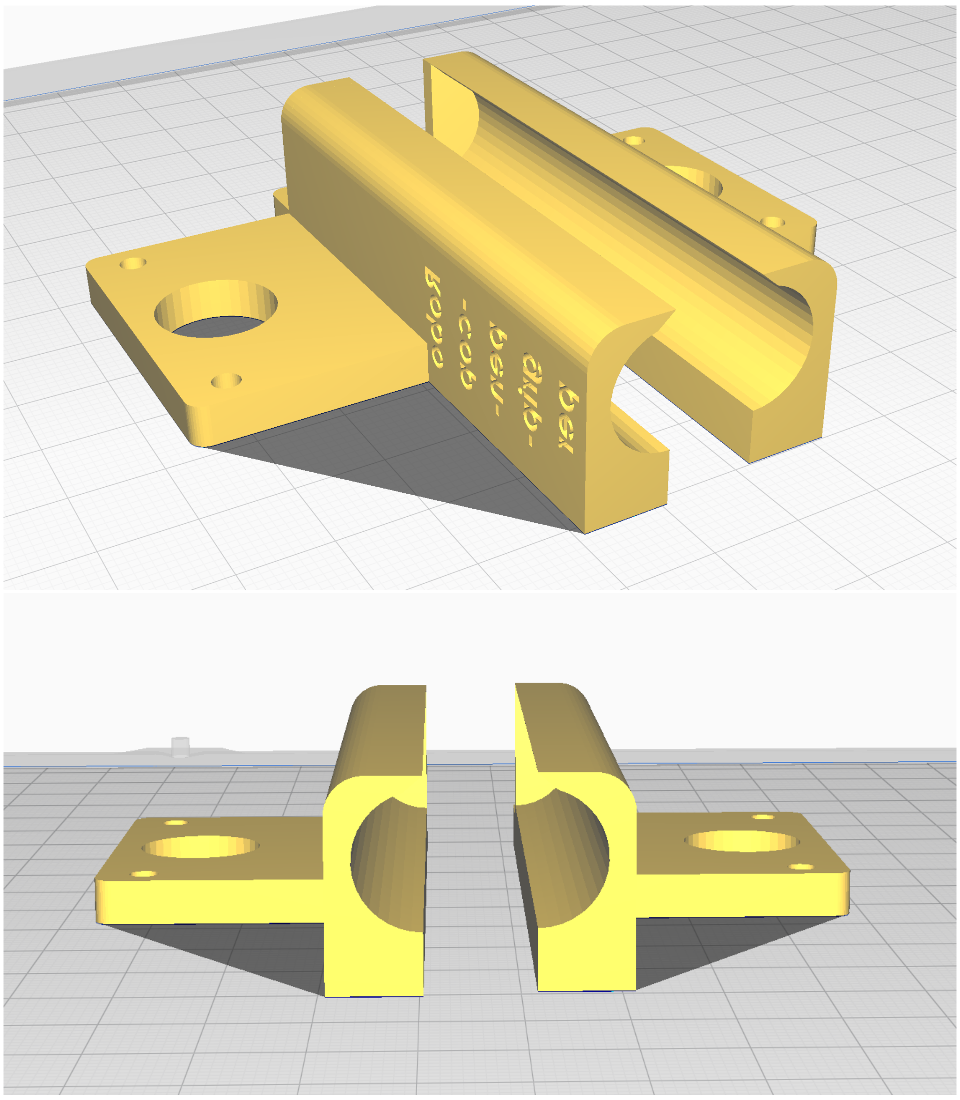

Figure 7: CAD model of the pen gripper (task 3)

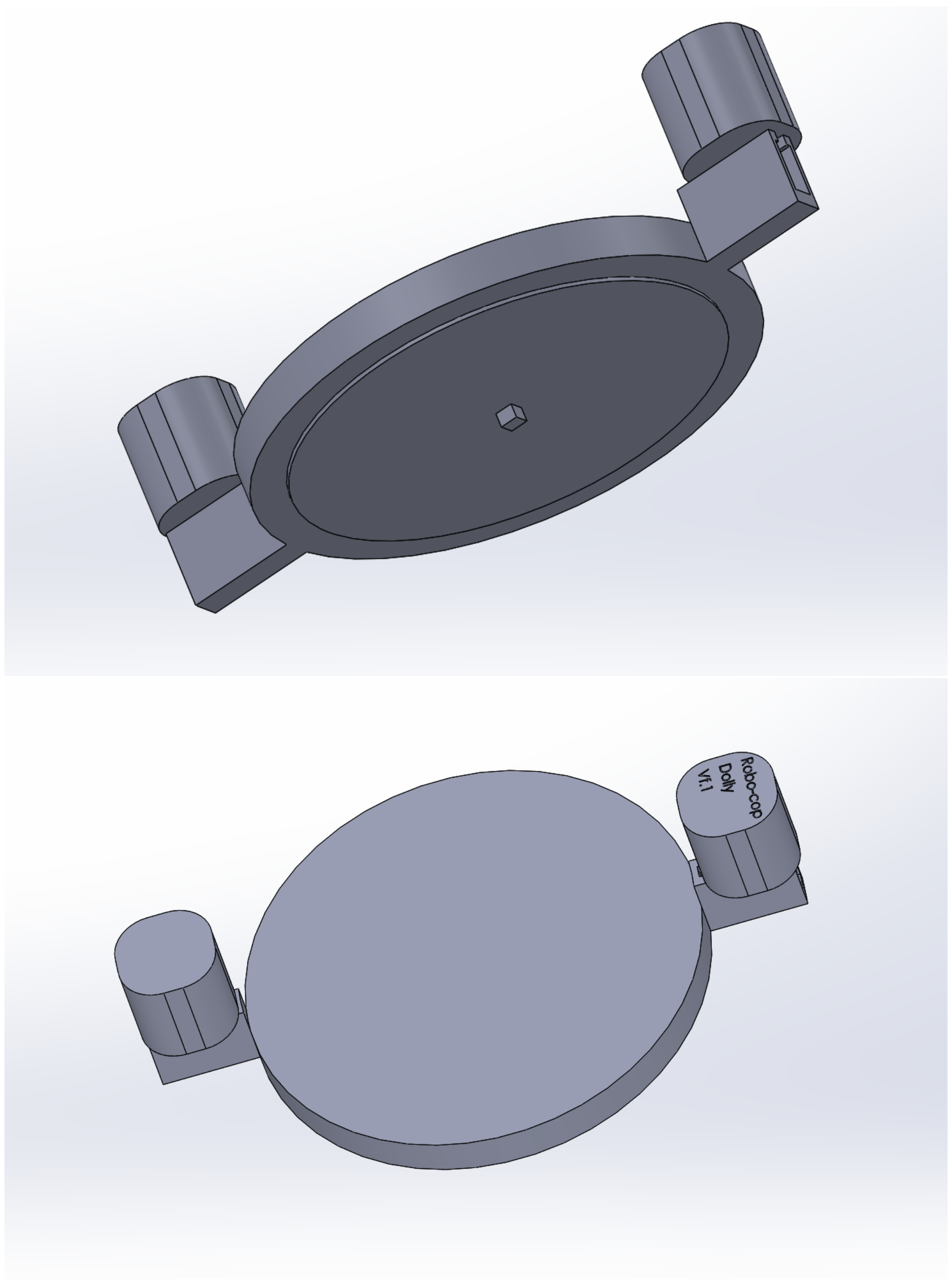

Figure 8: CAD model of rotating plate (task 4)

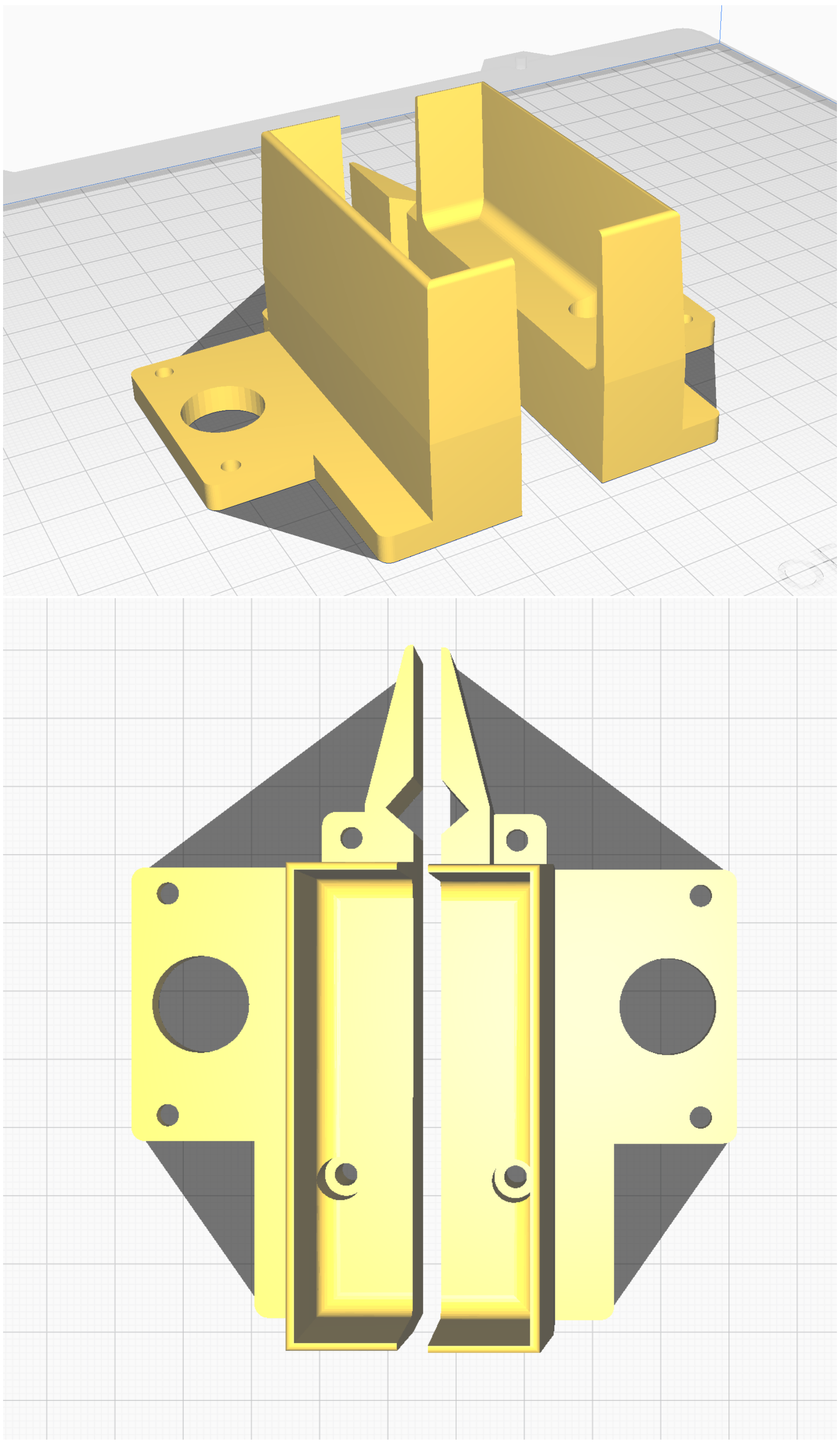

Figure 9: CAD model of the sushi gripper (task 4)