MAE 263A Kinematics of Robotic Systems | Final Project Report | Team 9

Super Scooper

Contents

1 Introduction

In this project, we designed, built and programmed a robotic arm with the task of scooping and depositing materials. We experimented primarily with food and could successfully scoop granular materials such as flour. We attempted to extend this to thicker solids such as ice cream but discovered that we would need more specialised equipment to support them.
In our report, we begin with a discussion of the arm's design in section 2, followed by the derivation of the kinematics and inverse kinematics in 3, then the simulation of the task in 4, the assembly and programming in 5 and lastly an evaluation in 6.

2 Design

We constructed a 4 degrees of freedom (DOF) robot for the scooping task. The mechanism consists of 4 revolute joints. After making an initial concept design (shown in Figure 1) we created a CAD model using Autodesk Fusion 360.

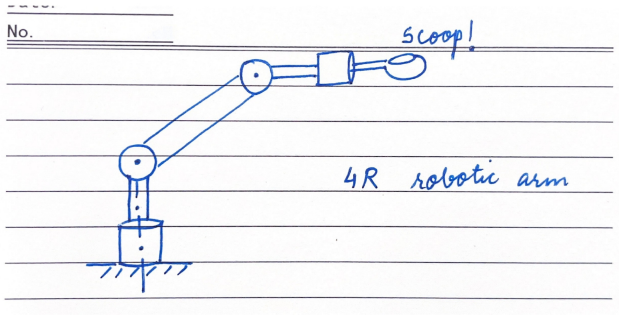
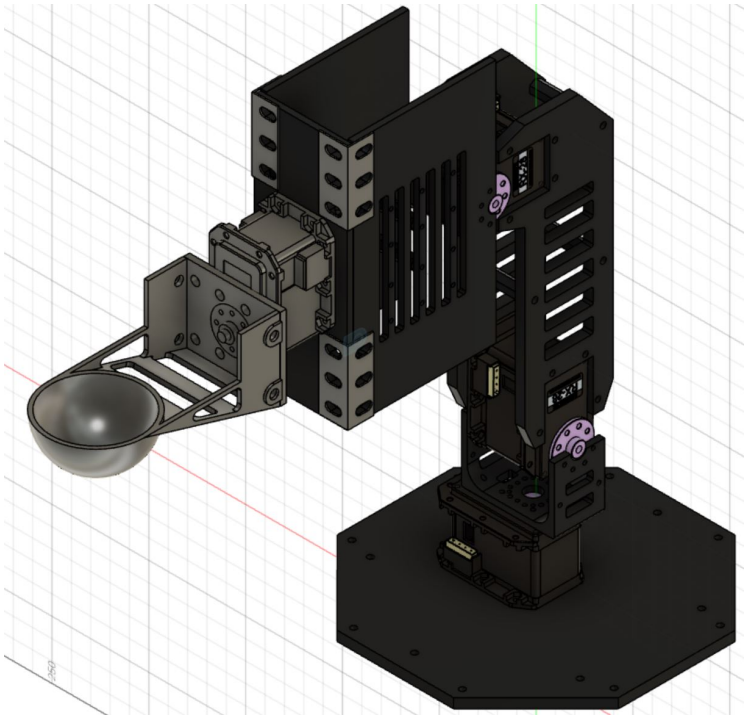


Figure 1: Conceptualization Stage Design

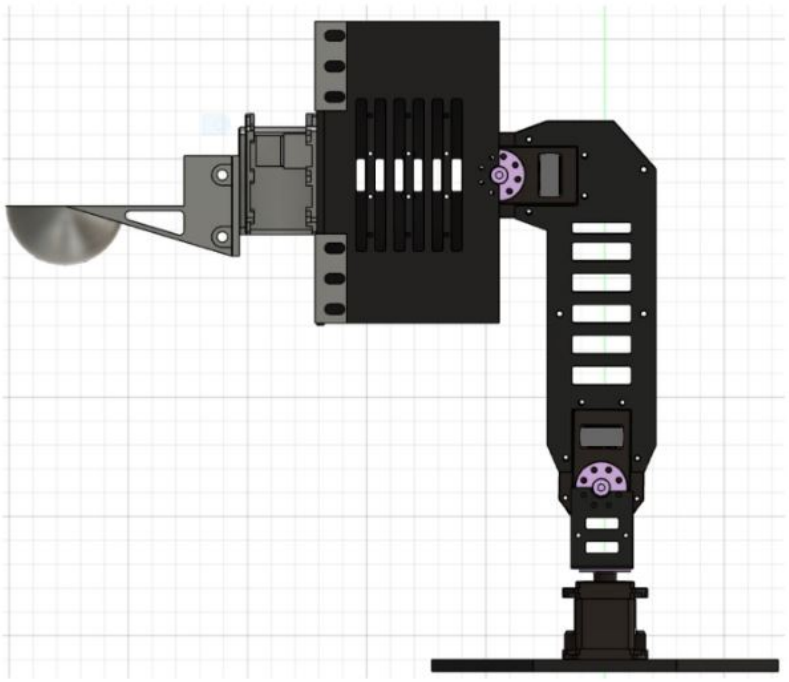
2.1 CAD & Fabrication

The CAD of the various linkages and the end-effector are shown in Appendix A. These were designed to be laser cut out of acrylic and later assembled with aluminium screws and brackets (sourced from Mechano kits). We chose this modular approach so we could flexibly adjust the prototype as we built it. The only exceptions are the end-effector, which we 3D-printed in one piece due to the curved shaped of the scoop and a few spacers to provide more lateral structural support.

The CAD model of the assembled robot is shown in 2a and 2b.



(a) Isometric View



(b) Side View

Figure 2: Robot CAD

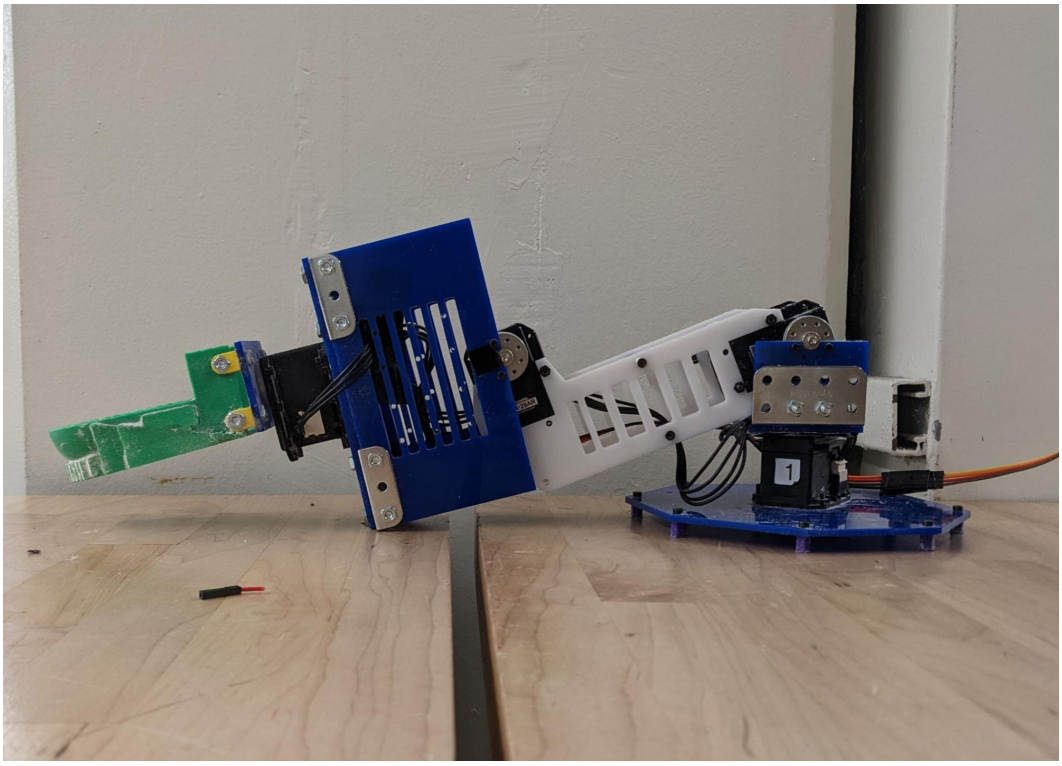
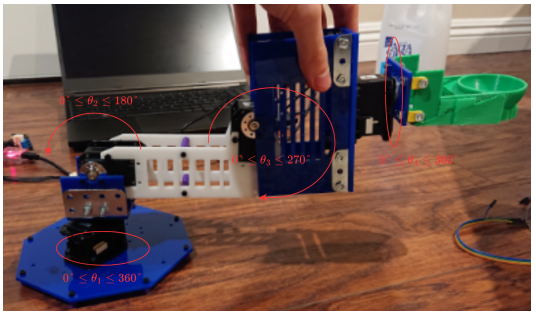


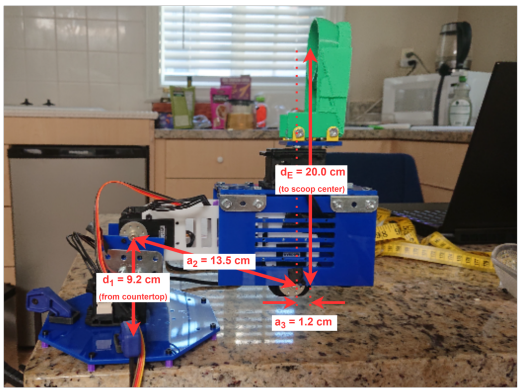
Figure 3: Fabricated Robot

2.2 Manipulator Configuration

The joint angle limits and the link lengths for the robot are shown in 4a and 4b respectively.



(a) Joint Angle Limits



(b) Link Lengths

Figure 4: Manipulator Configuration

2.3 Workspace Analysis

The workspace is an annular sphere as shown in 6. A visualization of the workspace, with extreme positions of the robot, is shown in 5. We can see that it roughly follows the annular sphere workspace that the MATLAB visualization indicates. Discrepancies arise between the theoretical and practical workspace realizations due to joint physical constraints.

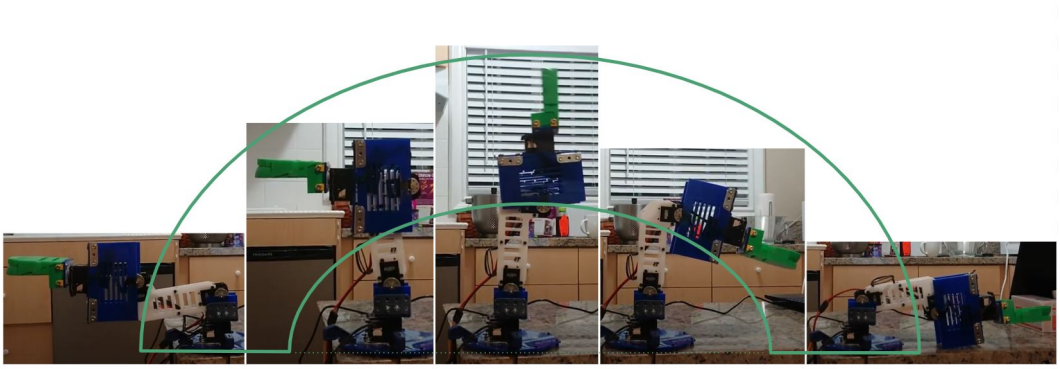


Figure 5: Workspace: Visualized using stills of the robot in different configurations

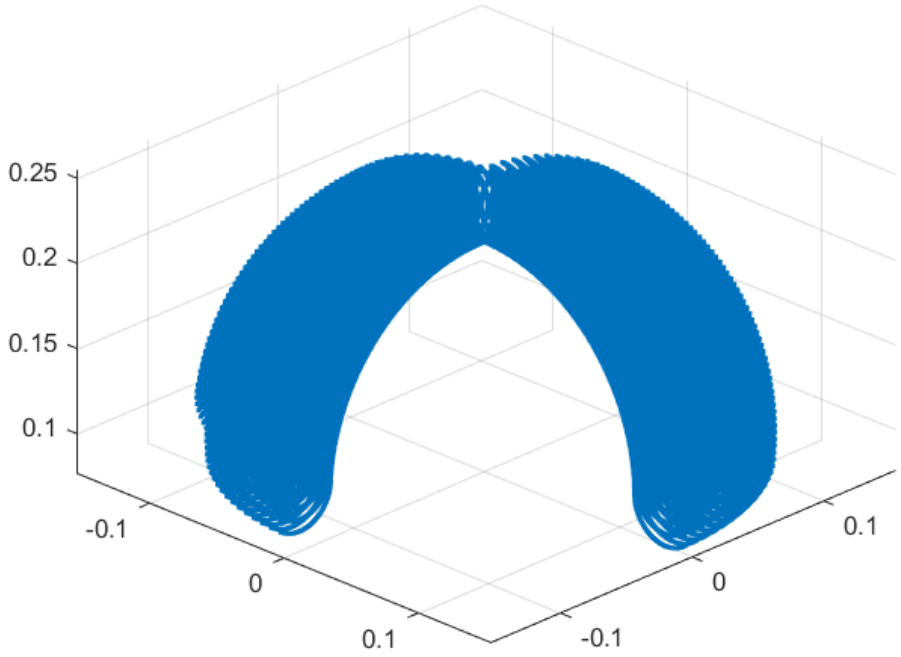


Figure 6: Workspace: Cross-section generated using MATLAB

3 Kinematics

3.1 Forward Kinematics

To begin calculating the forward kinematics of the robot, we must review the frame attachments. There are a total of six frames of interest: the base frame, the four joint frames, and the end effector frame. These are pictured below, along with the Denavit-Hartenberg (DH) parameters between frames. Symbolic variables are used for the nonzero link lengths and offsets for the forward and inverse kinematic derivations. To make trajectory planning easier, the origin of the base frame was placed on the ground instead of coincident with the 1st joint frame origin. Additionally, the end effector's origin was displaced along the z-axis of the 4th joint frame to the center of the spoon attachment, level with the circular edge of the spoon.

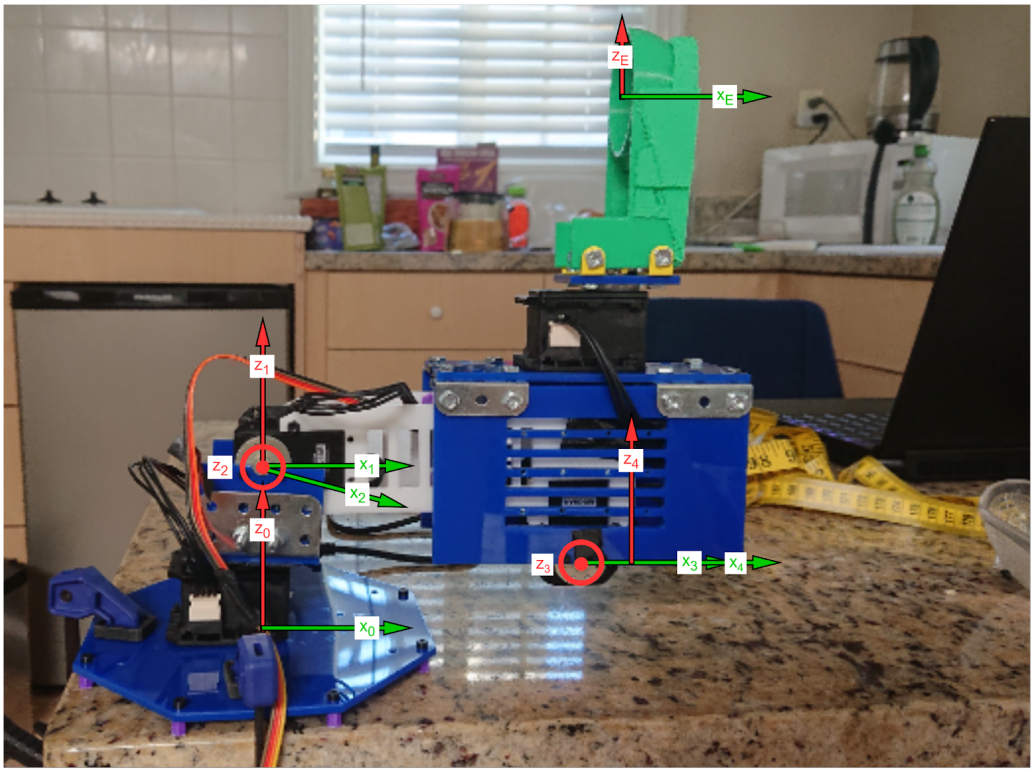


Figure 7: Frame attachments for our robot. Joint axes $\hat{z}_2$ and $\hat{z}_3$ point out of the page.

Table 1: DH parameters for our robot.

$i-1$	i	α_{i-1}	a_{i-1}	d_i	θ_i
0	1	$0°$	0	d_1	θ_1
1	2	$90°$	0	0	θ_2
2	3	$0°$	L_2	0	θ_3
3	4	$-90°$	L_3	0	θ_4
4	E	$0°$	0	d_E	$0°$

From the DH parameters, we can calculate the transformation matrices between adjacent frames. For more compact notation, $\cos\theta_1$ is denoted c_1, $\sin\theta_1$ is denoted s_1, and so forth.

$$ {}^0_1\mathbf{T} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1_2\mathbf{T} = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^2_3\mathbf{T} = \begin{bmatrix} c_3 & -s_3 & 0 & L_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^3_4\mathbf{T} = \begin{bmatrix} c_4 & -s_4 & 0 & L_3 \\ 0 & 0 & 1 & 0 \\ -s_4 & -c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^4_E\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_E \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Multiplying these in order and using angle sum identities repeatedly yields the transformation matrix relating the end effector frame to the base frame:

$$ {}^{0}_{E}\mathbf{T} = \begin{bmatrix} c_1c_4c_{23} - s_1s_4 & -s_1c_4 - c_1s_4c_{23} & -c_1s_{23} & (L_2c_2 + L_3c_{23} - d_Es_{23})c_1 \\ c_1s_4 + s_1c_4c_{23} & c_1c_4 - s_1s_4c_{23} & -s_1s_{23} & (L_2c_2 + L_3c_{23} - d_Es_{23})s_1 \\ c_4s_{23} & -s_4s_{23} & c_{23} & d_1 + L_2s_2 + L_3s_{23} + d_Ec_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Here c_{23} and s_{23} refer to $\cos(\theta_2 + \theta_3)$ and $\sin(\theta_2 + \theta_3)$, respectively.

3.2 Inverse Kinematics

All sets of possible joint angles to achieve a desired end effector position and orientation can be found by solving the inverse kinematics problem for the robot. We first equate the transformation matrix from the forward kinematics solution to a desired transformation matrix:

$$ {}^{0}_{E}\mathbf{T}_{FK} = {}^{0}_{E}\mathbf{T}_{desired} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The general solution process for each joint angle θ is to first determine $\sin\theta$ and $\cos\theta$, and then solve for $\theta = \text{atan2}(\sin\theta, \cos\theta)$. This method is useful in practice, since checking if $|\sin\theta| \leq 1$, $|\cos\theta| \leq 1$, and $\sin^2\theta + \cos^2\theta = 1$ for all joint angles reveals whether or not the desired transformation matrix is well-formed (i.e. corresponds to a feasible orientation and position).

To start, we observe that θ_3 cannot be calculated immediately; however, the sum $\theta_2 + \theta_3 = \theta_{23}$ can easily be solved for:

$$ s_{23} = \pm\sqrt{r_{13}^2 + r_{23}^2}, \quad c_{23} = r_{33} $$

$$ \theta_{23} = \text{atan2}(s_{23}, c_{23}) $$

θ_2 is only decoupled from θ_3 in the position command, so we solve for it using those terms:

$$ s_2 = \frac{p_z - d_Ec_{23} - L_3s_{23} - d_1}{L_2}, \quad c_2 = \frac{\pm\sqrt{p_x^2 + p_y^2} + d_Es_{23} - L_3c_{23}}{L_2} $$

$$ \theta_2 = \text{atan2}(s_2, c_2) $$

θ_3 can now be solved for easily:

$$ \theta_3 = \theta_{23} - \theta_2 $$

Finally, θ_1 and θ_4 can be solved using θ_{23}:

$$ s_1 = \frac{-r_{23}}{s_{23}}, \quad c_1 = \frac{-r_{13}}{s_{23}} $$

$$ \theta_1 = \text{atan2}(s_1, c_1) $$

and

$$ s_4 = \frac{-r_{32}}{s_{23}}, \quad c_4 = \frac{r_{31}}{s_{23}} $$

$$ \theta_4 = \text{atan2}(s_4, c_4) $$

3.2.1 Multiple Solutions

Note that for a given command there are two possible values for θ_{23}, and for a give θ_{23} there are two possible values of θ_2. Since the remaining joint angles don't have any such ambiguities, there are theoretically four sets of possible joint angles for a desired position and orientation of the end effector. This does not apply to cases on the edge of the robot's workspace, where the number of feasible solutions decrease.

One method of dealing with multiple possible sets of joint angles during end effector trajectory design is to first separate the desired trajectory into multiple stages. For each stage, all four sets of joint angles are calculated for the complete maneuver; the first set that has no infeasibilities is used. If all sets contain infeasible points, then the trajectory must be redesigned. Of course, another method is simply to bypass all inverse kinematics calculations by choosing joint angles directly. Our objective does not require a very complex end effector trajectory, so either method is acceptable.

3.2.2 Singularity Case

The above method of calculating θ_1 and θ_4 does not work if $\sin\theta_{23} = 0$ and $\cos\theta_{23} = \pm 1$. This is when the first and fourth joint axes $\hat{z}_1$ and $\hat{z}_4$ become parallel. In this case, θ_1 must be calculated using the position command:

$$c_{23} = +1: \qquad\qquad c_{23} = -1:$$

$$\begin{aligned} R_{XY} &= L_2 c_2 + L_3 c_{23} - d_E s_{23} \\ &= L_2 c_2 + L_3 \end{aligned} \qquad\qquad \begin{aligned} R_{XY} &= L_2 c_2 + L_3 c_{23} - d_E s_{23} \\ &= L_2 c_2 - L_3 \end{aligned}$$

$$s_1 = \frac{p_y}{R_{XY}}, \quad c_1 = \frac{p_x}{R_{XY}}$$

$$\theta_1 = \mathrm{atan2}(s_1, c_1)$$

θ_4 can only be found from the coupling relation between θ_1 and θ_4:

$$c_{23} = +1: \qquad\qquad c_{23} = -1:$$

$$\begin{aligned} s_{14} &= r_{21}, \quad c_{14} = r_{11} \\ \theta_{14} &= \mathrm{atan2}(s_{14}, c_{14}) \\ \theta_4 &= \theta_{14} - \theta_1 \end{aligned} \qquad\qquad \begin{aligned} s_{4-1} &= r_{21}, \quad c_{4-1} = -r_{11} \\ \theta_{4-1} &= \mathrm{atan2}(s_{4-1}, c_{4-1}) \\ \theta_4 &= \theta_{4-1} + \theta_1 \end{aligned}$$

The case where $\cos\theta_{23} = +1$ is when the first motor and end effector are both pointed straight up, while the case where $\cos\theta_{23} = -1$ is when they are pointed straight down. The second case is almost never feasible nor desirable, but is included for completeness.

This isn't truly a singularity, since there is still a finite number of solutions for the joint angles. However, if the first joint axis $\hat{z}_1$ and $\hat{z}_4$ become coincident, then $R_{XY} = 0$ in addition to $\sin\theta_{23} = 0$. In this case θ_1 and θ_4 cannot be decoupled, and there is an infinite number of solutions for these two joint angles. We choose to handle this case by maintaining our current θ_1 angle and solving for θ_4 using the same method as above. This approach is most efficient in terms of motor power since only the end effector must be rotated.

4 Simulation

4.1 Simulation Design

When designing the simulation, the goal was to rapidly prototype potential paths and trajectories for that would optimize the robots performance. In lieu of a formal optimization process given the time constraints, we adopted a qualitative quasi-optimization process that would at least allow us to check for any obviously extraneous movements or clearly improper behaviors. Additionally, being able to run many simulations at high speeds and from different angles resulted in a great idea of the physical performance, which could be used as a criterion when analyzing hardware trials to determine where any problems originated (i.e., if the simulation claims the robot should be behaving in X manner, and it behaves in Y manner, then it is more likely that our hardware calibrations were off or the codebase for the hardware has a bug, rather than an error existing in the underlying kinematics).

4.2 Simulation Implementation and Challenges

Our simulation is implemented in MATLAB and is a visualization of our inverse kinematics as a three-dimensional plot. The joint lengths on the plot are accurate and precise relative to each other in-scale with the hardware. The positions of the goals (and intermediate goals) were determined such that simple angles would be utilized by the robot (this made error analysis much easier.) Not every position was explicitly mapped out, but rather intermittent goal points were established and then interpolated. In the beginning, the simulation had only two goal points, and more complexity was added to loosely optimize the path of the end effector. As more goal points were added, further interpolation between the positions was possible, until there were about ten distinct states that were precisely determined, with interpolation and smoothing performed on the interim positions.

While there were certain adjustments that had to be considered when moving from the simulation to hardware, it still provided a good reference to check the hardware against. Notably, it was challenging to improve the scooping motion of the end effector while using the simulation, but was easier when getting a more qualitative feel for its performance when implemented in hardware. This was for two reasons: 1) The simulation did not have any substance that was being scooped by the robot and therefore we could not know how much it would actually end up obtaining, and 2) There is a certain "feel" for a three-dimensional action that is hard to analyze when looking at its representation on a screen. We imagine this challenge is common in robotics given that even the simplest of actions can have an instrisic human intuition to them, such as getting the right rotation/torque while scooping ice cream (which is natural to us, but not natural to describe as a set of angles.)

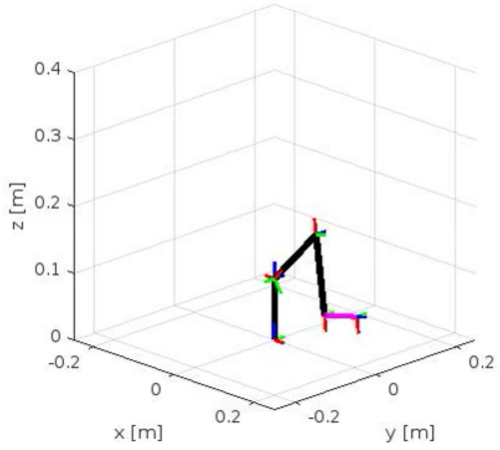


Figure 8: Simulation End Frame

5 Assembly & Electronics

5.1 Motor Testing

Before the robot was assembled, we verified each motor was functional and could be commanded. To do this, we used RoboPlus Launch and connected the motors to a laptop using U2D2. After modifying the ID number and labelling each motor, we could verify control my modifying the goal position parameter. After testing each individual motor, we set up a daisy chain of the motors as shown in Figure 9. This resembled the final configuration of the motors.

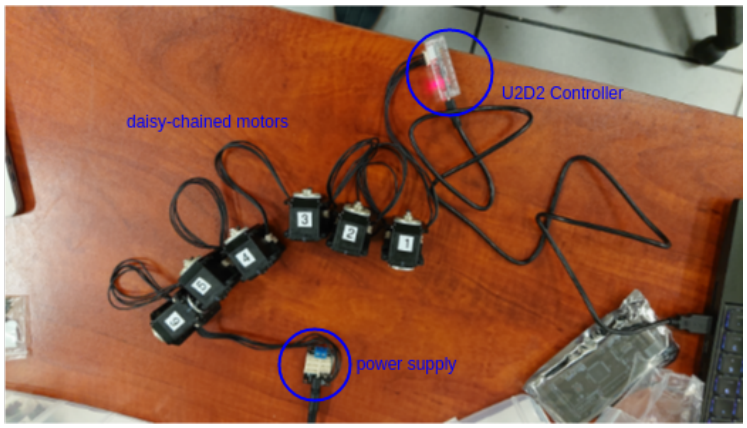


Figure 9: Annotated Diagram of Daisy Chained Motors

Besides verifying that the motors could work in a daisy chain configuration, we needed to be able to control the motors through a program. We opted to use Matlab due to it being the software the forward and inverse kinematics would be coded in. This required the use of the DynamixelSDK and a script provided to us to interface with the motors. Once the port was correctly configured, we could command the motors using Matlab.

5.2 Assembly

Once the parts were manufactured, the robot could be assembled. We used the provided nuts and bolts to join the parts together. The brackets were also used to provide structural support in assembling the parts together. The only difficulty we had in building the robot was the short length of some of the bolts. While they would hold the parts together sturdily, we would run into cases where we had to tighten the nuts because they would unfasten themselves.

The next challenging part was wiring up the motors. We had Motor 1 connected to the computer port, and then each subsequent motor would be connected to the motor before it. For example, the motor for the second joint was connected to Motor 1 and Motor 3. Then for Motor 6, we wired it up with Motor 5 and the power source. Throughout all of this, we had to keep in mind that the motors needed to have enough slack to rotate through its range of motion. As the motors rotated from the home position, the amount of wire needed to keep the connection in place would increase. If the wire was not long enough, then the rotation would force the connection between the motors to come apart, breaking the daisy chain. This was also balanced with introducing too much excess wiring between the motors. When too much wiring was added, the weight of the wire would sometimes pull the connection out or the excess wire would get caught between the joints. Therefore, we had to use electrical tape to secure parts of the wire to the links so that they would not get caught and pulled out.

When the final assembly was done, we could proceed onto demoing the robot with the software. The final assembly can be seen in Figure 3.

5.3 Demo

As stated previously, the goal of the robot was to scoop out ice cream from a source and place it into a destination bowl. This involved multiple steps, as covered in Section 4. We originally proceeded

with using position control for the motors, telling each motor the goal position for each step. This caused numerous issues. First, the speed at which these motors rotate was very fast, causing jerkiness. This was undesirable due to our goal of transporting a substance without spilling it. Another issue we faced was that the rotation direction could not be easily controlled with this method. When only a goal position was specified to the motor, it would rotate to get to that goal without analyzing to see if it would collide into an obstacle such as another link or the floor. Therefore, we decided to go with velocity control to move the motors. This was we could specify a rotation speed and a rotation direction.

5.3.1 Integration Challenges

We were met with difficulties getting the robot to execute the planned trajectory at the very beginning. The first challenge when sending commands came when the robot was fully assembled. One of the wires used was faulty, and any command sent to the motors could not reach its destination. This was relatively straight forward to debug because we could test each motor's connection individually, which allowed us to narrow in on the faulty connection.

Another hurdle we overcame was getting the motors to rotate to the desired angle. The default home position of the motors of 0° did not correspond to the home position denoted by the coordinate frame we used for our Forwards Kinematics. This meant that telling the motor to move to the 10° position would result to a different position than that calculated from Forwards Kinematics. With that type of result, the trajectory planned from Inverse Kinematics would also be incorrect. Therefore, we moved the robot to the "Home Position", took note of the angles read back from the motors, and used those values to offset each rotation. This allowed us to artificially set the "Home Position" for each motor to $\theta = 0^\circ$.

5.3.2 Final Trajectory

With all the hardware in place and software integrated, we could finally perform the planned trajectory of the robot. We first got the trajectory working without putting anything in its path. The angles we used were calculated from the Inverse Kinematics solution. Once this was working properly, we tried with a light material: flour. This was safe because there was no concern of getting liquids on any sensitive electronics and would be relatively easy to clean up if the robot malfunctioned. After finalizing the scooping motion and the rest of the trajectory, we re-ran it on ice cream. Unfortunately, it proved much more difficult to work with, being too tough for the motor torque when frozen and too viscous to deposit as it approached room temperature. It also posed a bigger hazard to the circuitry and motors, so we ceased further experiments after a few failed attempts.

6 Evaluation & Future Work

In this project we have successfully managed to design, assemble, program and demo our scooping robotic arm. Overall, we are very satisfied with this outcome, although there are clearly a few ways to improve it. For example, it is currently only able to reach and deposit very close locations, which limits its abilities. Additionally, as we demonstrated, it would require more force to interact with harder materials like frozen food. This can be improved simply with better specs: longer links and stronger motors. Additionally in the specific context of ice cream, an actual ice cream scoop features in-built sweeper that makes removing material from the scoop easier, which could be controlled by another motor.
One other aspect of course, is continuous operation, as the robot should ideally scoop as much material as is possible over multiple runs from a container. However, this is effectively a series of tasks, which requires sensory feedback (for example via vision) so the controller can plan suitable trajectories based on the material's position and quantity. Hopefully, we will learn how to do this in future classes.

Appendix A Robot Linkages

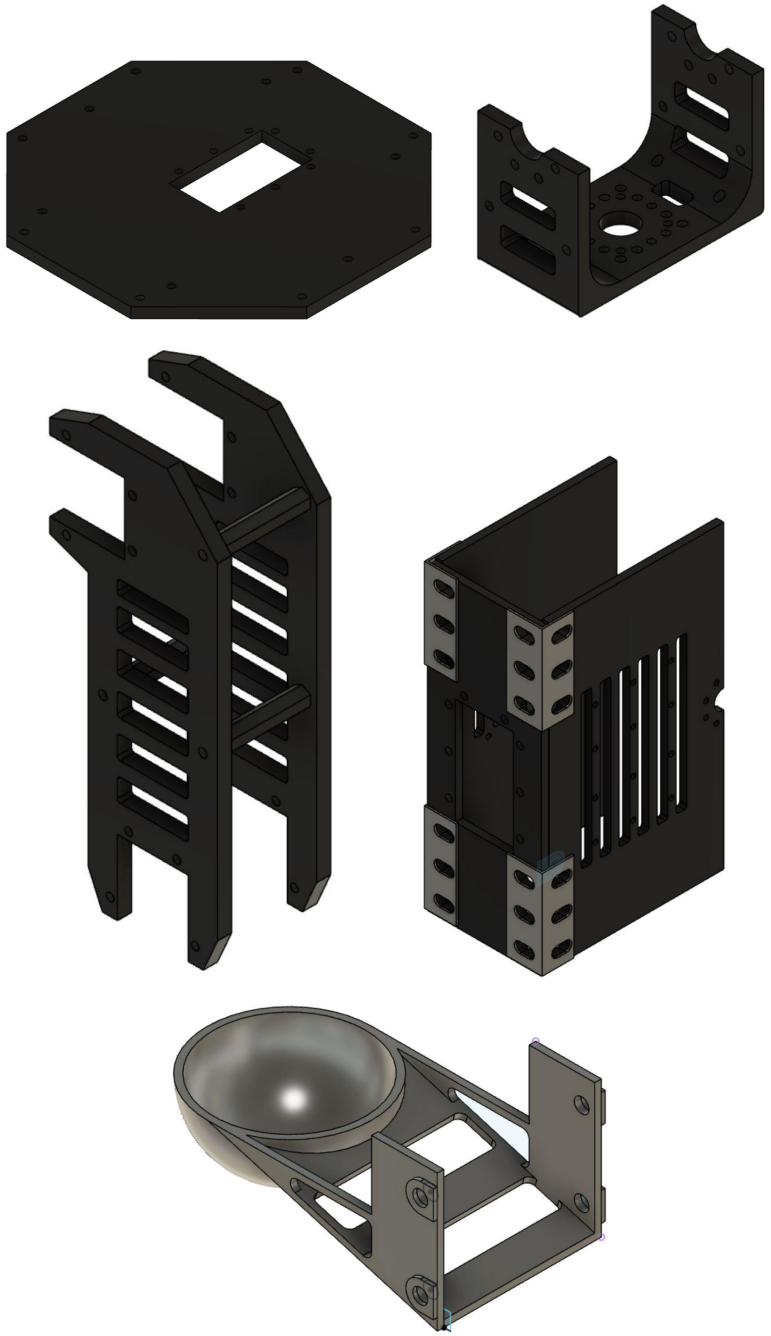


Figure 10: CAD of linkages