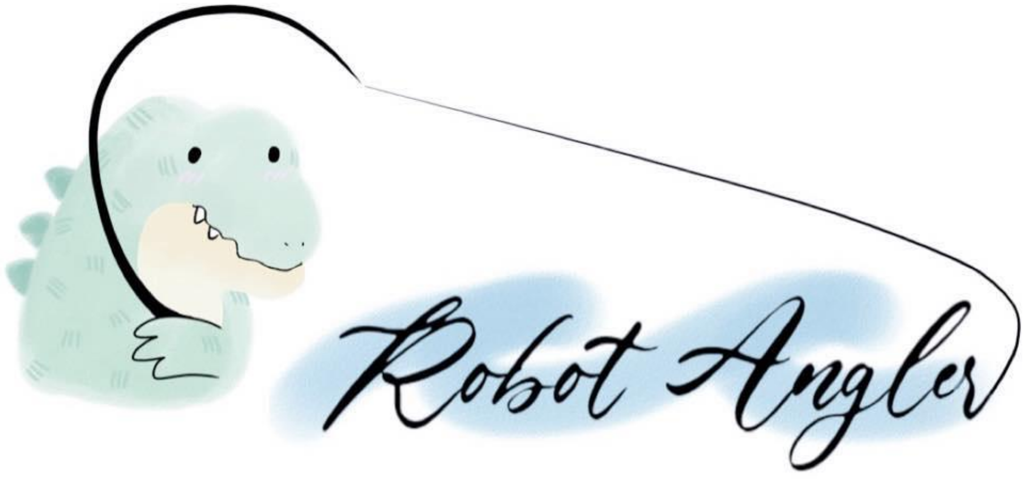


Robot Angler

MECH&AE C263A
Team 13

Alysa Kataoka, Cheryl Lee, Harry Sandstrom,
Jingran Meng, Jinyoung Kim

Table of Contents

Design

Manipulator Configuration

We used three prismatic joints and one revolute joint (PPRP) to complete our manipulator design. The overall configuration of our manipulator is shown below in Figure 1. Figure 1 also shows the frame attachments to each joint and the definition of our DH parameters. The DH parameters are discussed in greater detail in the Forwards Kinematics section of this report.

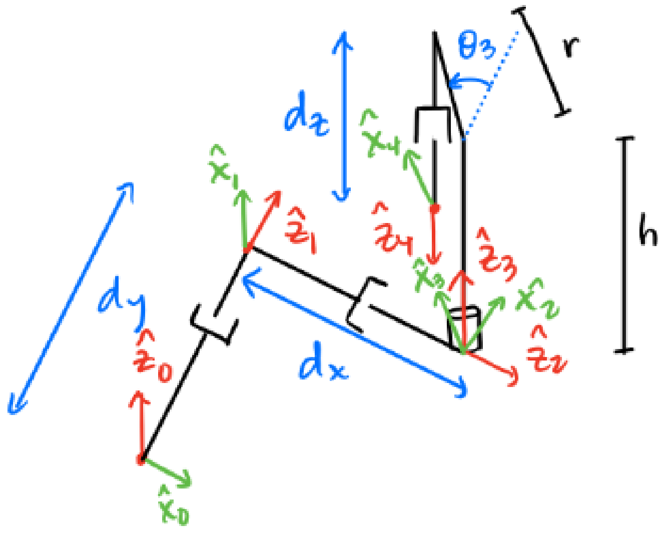


Figure 1. Configuration for our PPRP manipulator.

Hardware

Our design incorporates three prismatic joints and one rotational joint. The design uses four motors in total. The two prismatic joints that are parallel to the $\hat{x}_0$ and $\hat{y}_0$ axes use ball screws to transform Motor 1's and 2's rotational motions to linear translations. Next, a third motor is attached to the base of Joint 3 and directly rotates the vertical portion of our manipulator. The fourth and last joint in our manipulator is a prismatic joint that translates in the $\hat{z}_0$ direction. For this joint we are utilizing a winch and string system to translate the motor's rotational motion to linear translation. Our end effector is a magnet attached to the end of the string. The three prismatic joints and one rotational joint are shown below in Figure 2. Also, the physical assembly of our model is shown below in Figure 3.

The maximum possible x- and y- translation of the base of our manipulator is 15 cm. We are choosing to constrain our third joint's rotation to be within -90° to 90° as we want to prevent collision with the base structure. The 4th joint can translate our end effector a maximum of 15 cm in the ground frame z- direction.

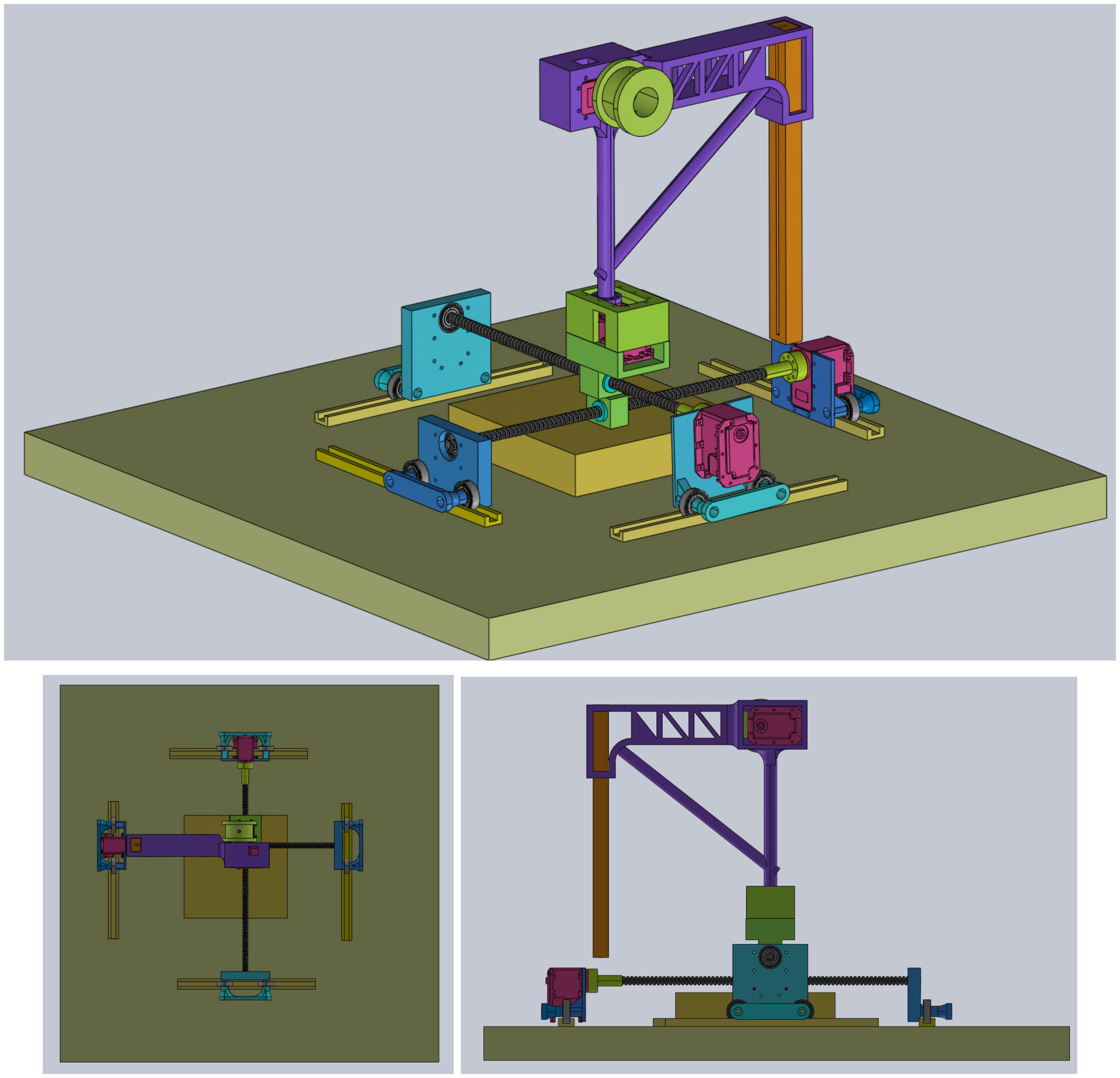


Figure 2. Perspective, top, and side view of CAD model.

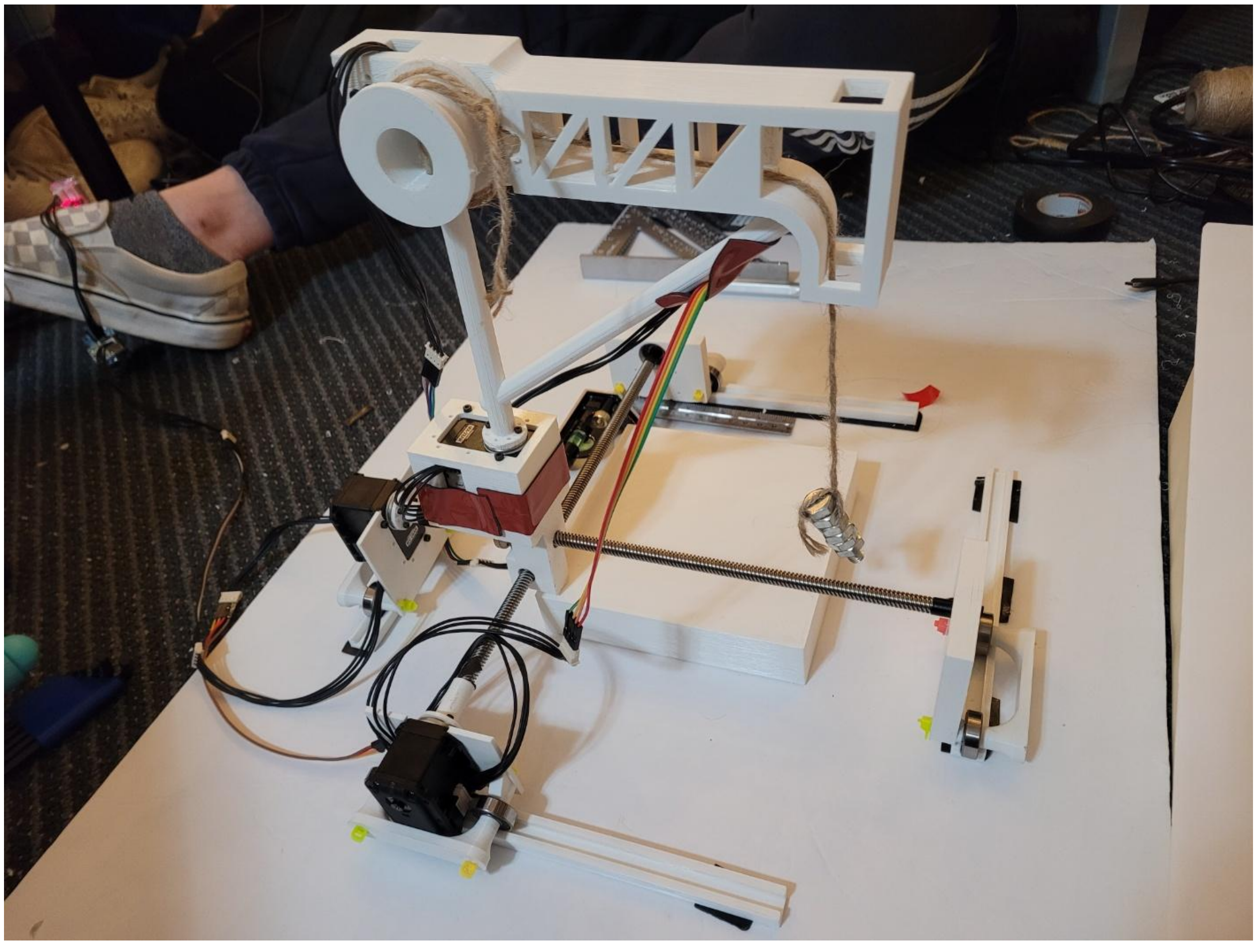


Figure 3. Hardware assembly

Workspace Analysis

The workspace of a robot includes all of the possible positions that can be reached by the end effector. For our robot, the orientation of the end effector was not specified; therefore, the entire reachable workspace was considered rather than only the dexterous workspace. As in the case of inverse kinematics, two separate cases were considered: one where d_x is constrained and the other when d_x is nonzero. In the constrained case, the workspace is visualized as the pink volume shown in Figure 4. The radius of the semi-cylindrical volumes correspond to the length of ‘r’, while the length of the straight portion of the horizontal edge is determined by y_{max}, which denotes the maximum range of our second prismatic joint. x_{max} is defined similarly. The height of the workspace is determined by h. In the scenario when the first prismatic joint is unconstrained, the workspace increases to the black volume shown in Figure 4. It can be observed that the workspace in this case is simply the sum of the constrained workspaces as d_x increases when the prismatic joint extends.

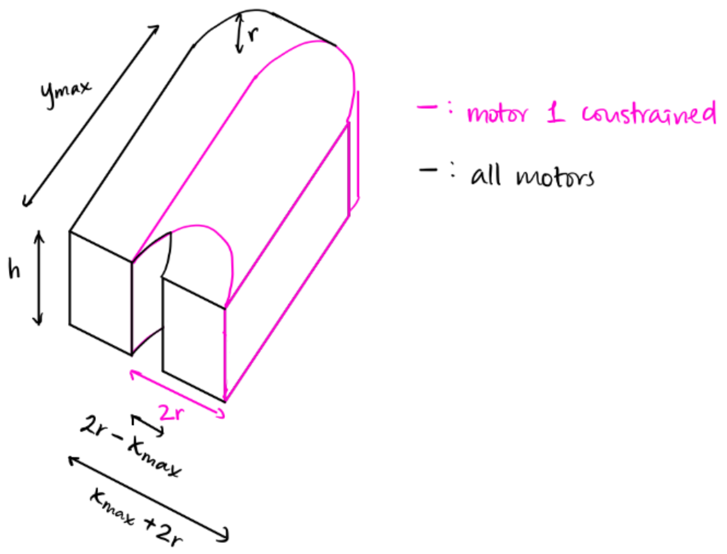


Figure 4. Workspace of the robot

Forward Kinematics

To derive our forward kinematics solution, we first derived our Denavit-Hartenberg (DH) parameters. These parameters are shown in Table 1 below. Please note that θ_3 is constrained to range from -90° to 90° only as we want our manipulator picking up items 'in front' of it. Additional detail on the configuration of our manipulator can be seen in the previous Manipulator Configuration section in Figure 1.

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	-90°	d_y	-90°
2	0	-90°	d_x	-90°
3	0	-90°	0	θ_3
4	r	180°	$h - d_z$	0

Table 1. DH parameters

Using the parameters shown above, our individual transformation matrices for each joint are as follows:

$$
{}^{0}_{1}T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_y \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{1}_{2}T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_x \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}_{3}T = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(\theta_3) & -\cos(\theta_3) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{3}_{4}T = \begin{bmatrix} 1 & 0 & 0 & r \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & d_z - h \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Using the relation that ${}^{0}_{4}T = {}^{0}_{1}T {}^{1}_{2}T {}^{2}_{3}T {}^{3}_{4}T$, we calculated the overall forward kinematic solution for our manipulator problem.

$$
{}^{0}_{4}T = \begin{bmatrix} -\sin(\theta_3) & \cos(\theta_3) & 0 & d_x - r\sin(\theta_3) \\ \cos(\theta_3) & \sin(\theta_3) & 0 & d_y + r\cos(\theta_3) \\ 0 & 0 & -1 & d_z - h \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Inverse Kinematics

To calculate our inverse kinematics solution, first we defined our problem. Given a desired $[p_x, p_y, p_z]^T$, we wish to find the following DH parameters: $[\theta_3, d_x, d_y, d_z]$. Since we have three desired constraints and four degrees of freedom, we split our solutions into two cases and define our d_x parameter based on our situation. Our first case is if our target is less than ‘r’ away from the origin in the $\hat{x}_0$ direction. In this case, we constrain $d_x = 0$. Next, if our target is more than ‘r’ away from the origin in the $\hat{x}_0$ direction, we defined d_x as such:

$$d_x = \begin{cases} p_x - r & if\ p_x > 0 \\ p_x + r & if\ p_x < 0 \end{cases}$$

From here, we are able to define the other three parameters. We used the geometric method to calculate our inverse kinematics solution which is shown below:

$$\theta_3 = \sin^{-1}\left(\frac{d_x - p_x}{r}\right)$$

$$d_y = \begin{cases} p_y - r & if\ \theta_3 = 0 \\ p_y - \frac{d_x - p_x}{\tan(\theta_3)} & otherwise \end{cases}$$

$$d_z = p_z - h$$

We noted that our solution only exists when $r \neq 0$, which should always be true for our physical system. Additionally, for our solution for d_y , we found that the solution was originally undefined when $\theta_3 = 0$. In this case, d_y is simply the difference between our target y coordinate and our manipulator's y-direction maximum reach length, or 'r'. This adjustment to our solution has been reflected above.

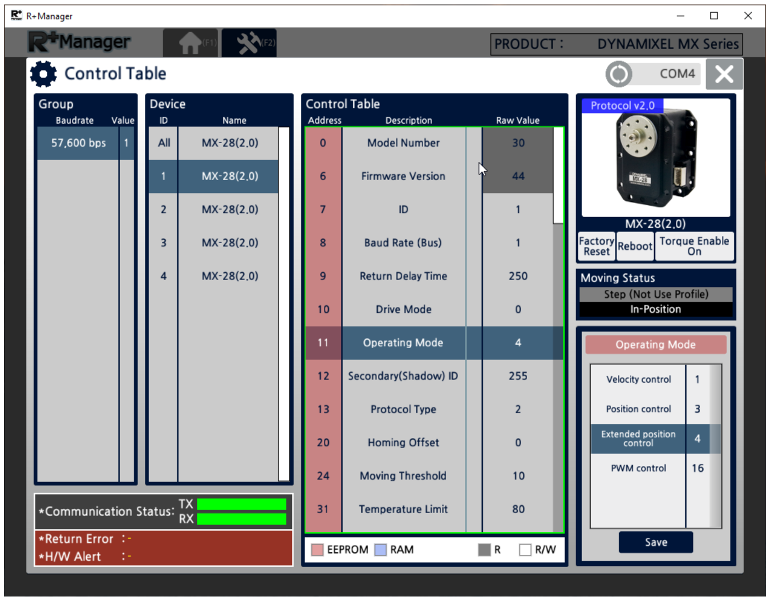
Simulation

To simulate our manipulator in action, we decided to test two separate cases - one where the manipulator's x-direction motion was constrained to be equal to 0, and one where the x-direction prismatic joints are used. Our manipulator works by starting at an 'origin' position which is where each DH parameter value is equal to 0, moving to the target location, and then moving back to the origin position before moving to the subsequent target location. Our code works by calculating the necessary DH parameters to reach the target location, then using linear interpolation to smoothly move between each configuration of the manipulator.

Our simulation is demonstrated in the animation file named simulation_animation attached to this report. Please note that in our simulation, each frame's x- and z- coordinate axes are denoted by the green and red lines respectively. Additionally, the target coordinate is marked by the red cross and the link that the end effector is attached to is denoted by the blue line. Our full code implementation can be seen in the attached .m file.

Control

To control motors, we first changed the device ID for four motors. Because we have three prismatic joints, we also changed the operating mode to *Extended position control* of all prismatic joints (motor 1, 2, and 4) to make sure they can rotate exceeding 2π (1 revolution).


R+Manager

PRODUCT : DYNAMIXEL MX Series

Control Table COM4

Group Baudrate	Value
57,600 bps	1

Device ID	Name
All	MX-28(2.0)
1	MX-28(2.0)
2	MX-28(2.0)
3	MX-28(2.0)
4	MX-28(2.0)

Address	Description	Raw Value
0	Model Number	30
6	Firmware Version	44
7	ID	1
8	Baud Rate (Bus)	1
9	Return Delay Time	250
10	Drive Mode	0
11	Operating Mode	4
12	Secondary(Shadow) ID	255
13	Protocol Type	2
20	Homing Offset	0
24	Moving Threshold	10
31	Temperature Limit	80

EEPROM RAM R R/W

*Communication Status: TX RX

*Return Error : -
*H/W Alert : -

Protocol v2.0

MX-28(2.0)

Factory Reset | Reboot | Torque Enable On

Moving Status
Step (Not Use Profile)
In-Position

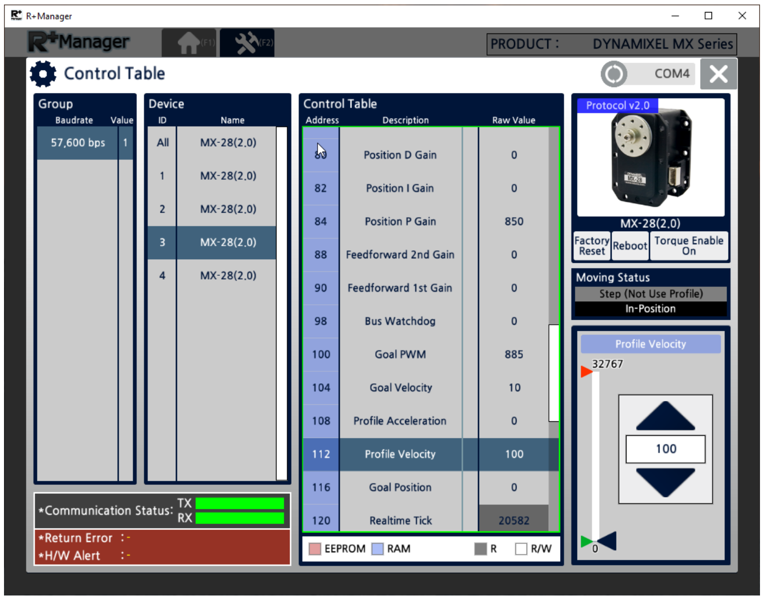
Operating Mode

Velocity control	1
Position control	3
Extended position control	4
PWM control	16

Save

Figure 5. Operating Mode: Extended position control

In case the revolute joint generates too much torque that 3-D printed models cannot hold, we changed the velocity of the revolute joint (motor 3). Also, in order to make the motor mounts do not come out of the sliding track, we lowered the velocity of the prismatic joints (motor 1, 2, and 4).


R+Manager

PRODUCT : DYNAMIXEL MX Series

Control Table COM4

Group Baudrate	Value
57,600 bps	1

Device ID	Name
All	MX-28(2.0)
1	MX-28(2.0)
2	MX-28(2.0)
3	MX-28(2.0)
4	MX-28(2.0)

Address	Description	Raw Value
80	Position D Gain	0
82	Position I Gain	0
84	Position P Gain	850
88	Feedforward 2nd Gain	0
90	Feedforward 1st Gain	0
98	Bus Watchdog	0
100	Goal PWM	885
104	Goal Velocity	10
108	Profile Acceleration	0
112	Profile Velocity	100
116	Goal Position	0
120	Realtime Tick	20582

EEPROM RAM R R/W

*Communication Status: TX RX

*Return Error : -
*H/W Alert : -

Protocol v2.0

MX-28(2.0)

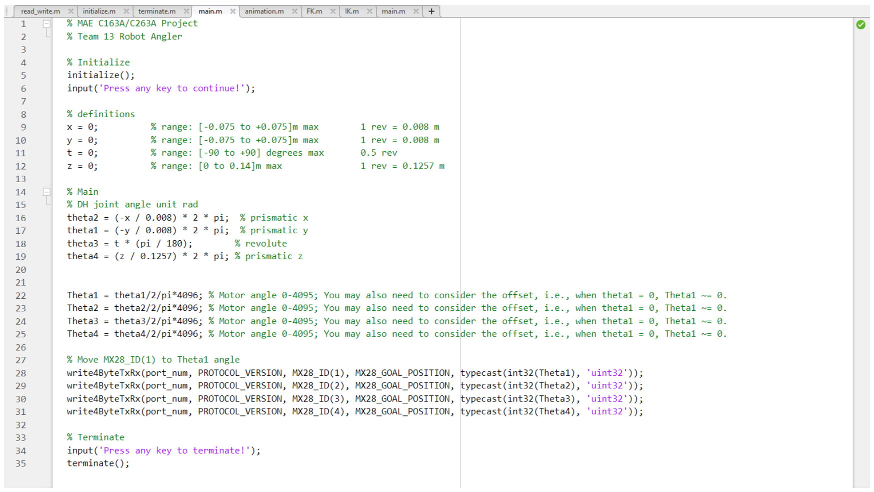
Factory Reset | Reboot | Torque Enable On

Moving Status
Step (Not Use Profile)
In-Position

Profile Velocity
32767
100
0

Figure 6. Profile Velocity

Then, we did the conversion for the prismatic joints based on the dimensions of 3-D printed design. The pitch is 8 mm and the pulley circumference is 0.1257 m; so we divided x, y distance by 8 mm and divided z by 0.1257 m then multiplied by 2π to get the angles in radians.



```matlab
% MAE C163A/C263A Project
% Team 13 Robot Angler

% Initialize
initialize();
input('Press any key to continue!');

% definitions
x = 0;          % range: [-0.075 to +0.075]m max        1 rev = 0.008 m
y = 0;          % range: [-0.075 to +0.075]m max        1 rev = 0.008 m
t = 0;          % range: [-90 to +90] degrees max       0.5 rev
z = 0;          % range: [0 to 0.14]m max               1 rev = 0.1257 m

% Main
% DH joint angle unit rad
theta2 = (-x / 0.008) * 2 * pi;  % prismatic x
theta1 = (-y / 0.008) * 2 * pi;  % prismatic y
theta3 = t * (pi / 180);        % revolute
theta4 = (z / 0.1257) * 2 * pi; % prismatic z


Theta1 = theta1/2/pi*4096; % Motor angle 0-4095; You may also need to consider the offset, i.e., when theta1 = 0, Theta1 ~= 0.
Theta2 = theta2/2/pi*4096; % Motor angle 0-4095; You may also need to consider the offset, i.e., when theta1 = 0, Theta1 ~= 0.
Theta3 = theta3/2/pi*4096; % Motor angle 0-4095; You may also need to consider the offset, i.e., when theta1 = 0, Theta1 ~= 0.
Theta4 = theta4/2/pi*4096; % Motor angle 0-4095; You may also need to consider the offset, i.e., when theta1 = 0, Theta1 ~= 0.

% Move MX28_ID(1) to Theta1 angle
write4ByteTxRx(port_num, PROTOCOL_VERSION, MX28_ID(1), MX28_GOAL_POSITION, typecast(int32(Theta1), 'uint32'));
write4ByteTxRx(port_num, PROTOCOL_VERSION, MX28_ID(2), MX28_GOAL_POSITION, typecast(int32(Theta2), 'uint32'));
write4ByteTxRx(port_num, PROTOCOL_VERSION, MX28_ID(3), MX28_GOAL_POSITION, typecast(int32(Theta3), 'uint32'));
write4ByteTxRx(port_num, PROTOCOL_VERSION, MX28_ID(4), MX28_GOAL_POSITION, typecast(int32(Theta4), 'uint32'));

% Terminate
input('Press any key to terminate!');
terminate();
```

Figure 7. Matlab conversion

After we adjusted the settings of all motors, we connected all motors in a daisy chain link and then assembled the motors and 3-D printed models. We also put some hex nuts on the end of the string to make the fishing rod go up and down smoothly. After finishing the assembly, we ran additional tests like zeroing all joints, updating the motor velocity to keep the robot stable when it moves, and testing if the robot can reach a designated point.

Highlights and Difficulties

Our team encountered various challenges in the building of the robot. When first testing the robot, we encountered the most issues with our first motor. The motor would often blink red and required us to readjust the motor velocity and initialize all the joints once again. Additionally, we had to create several iterations of the CAD design due to issues with clearance in physical assembly. This process was time-consuming due to the required printing time. Furthermore, the friction between the components made the robot difficult to move smoothly and locate the end effector to our goal position. For example, in Figure 2, we designed the orange colored rectangular prism in order to guide the string straight downward. However, it turned out that friction between the rectangular prism and purple colored link was too large. Therefore we had to remove the prism and make the string heavier by adding some loads at the end effector to guarantee straight movement of the end effector.

We also found that the design of our robot facilitated its implementation and performance in other ways. For example, the concept of utilizing a magnetic end effect and target enabled our robot to pick up the target more easily. Additionally, having three prismatic joints greatly simplified our over forward and inverse dynamics equations. Overall, our team was satisfied with the design and performance of our Robot Angler.