DANIEL TAKYI
SID: 200376303

UFUOMA AYA
SID: 200327306

DATE: 26/03/2020

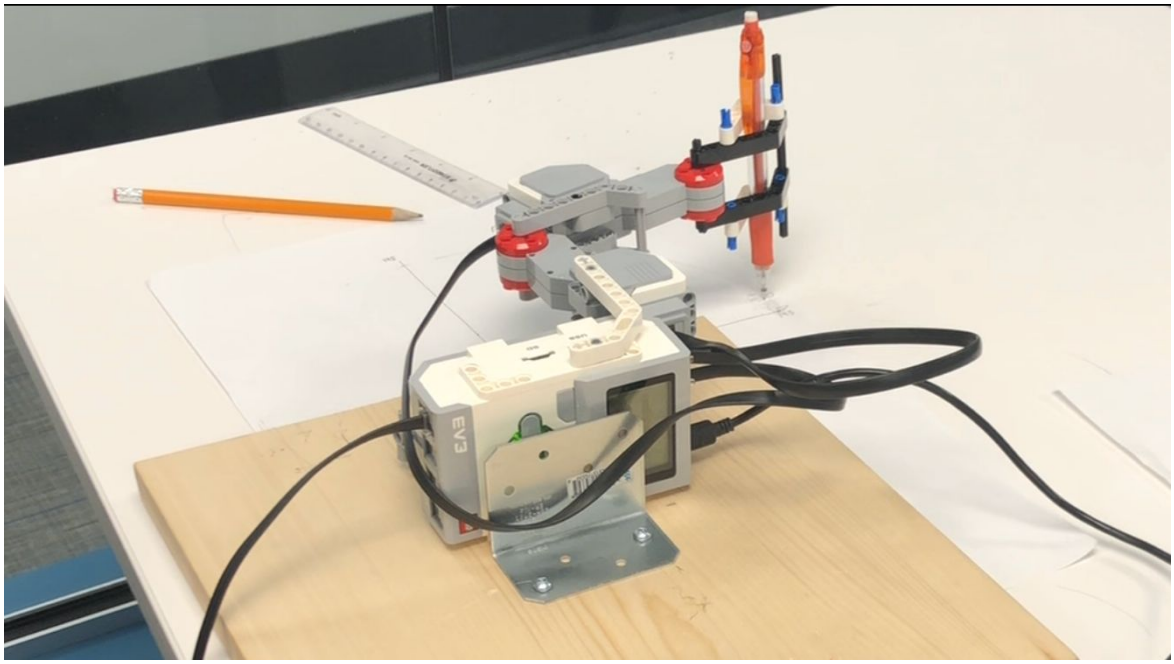
ENEL 484
PROJECT GROUP REPORT

# 2DOF ROBOT ARM USING PROVIDED LEGO EV3 MINDSTORMS ROBOT

## 1. BUILD

In the successful build of the 2DOF Robot Arm using provided Lego EV3 Mindstorms Robot, two large motors were utilized and we ensured that the two rotating joints were on the same plane. We created connections between the EV3 brick and the large motors via cables (35 cm / 14 in).

Also, we connected a touch sensor to the EV3 brick using a cable (25 cm / 10 in). Several beams, connector pegs, blocks and axles were used to ensure that the arm and attached pencil at the end effector were firm and steady, resulting in a required build as shown in the image below.




## 2. WORKSPACE

We started off defining our workspace by first finding the length of the arm. The length of the arm is 145 mm. The arm is split into two links. So we declared variables L1 and L2, and initialized them with

their real world lengths, 105mm and 40mm, respectively. When the robot arm is straight out, the end effector is at the coordinate (0, 145). When the arm is perpendicularly right of that and straight, the end effector is at (145, 0). When both angles of the motor are 0, the end effector is once again at (145, 0).

## 3. FUNCTIONALITIES

### *Part 1: Measure distance*

The function MeasureDistance is capable of measuring the space between two given points. In order to measure the distance between the two points we needed to first find the coordinates of the two points. This was done by going to each point, pressing the touch sensor, and using a calculate_coordinates function. In calculate_coordinates we are able to find the x and y coordinates for a point using these lines of code:

```
x_coordinate = L1*cosd(angle1) + L2*cosd(angle1+angle2);
y_coordinate = L1*sind(angle1) + L2*sind(angle1+angle2);
```

After finding the coordinates of each point we were able to calculate the distance between the two points using:

```
distance = sqrt((coordinates_2(1) - coordinates_1(1))^2 + (coordinates_2(2) -
    coordinates_1(2))^2);
```

Once the value is calculated, they are converted to a string format and then displayed on the screen.

### *Part 2: Measure angle*

The MeasureAngle function takes two intersecting lines and calculates the angle between them. First we take the arm to the intersection point of the two lines and press the touch sensor so it can find the

coordinates at that point. Then we press the sensor when the arm is at each line's end point and find those coordinates. We then calculate the angle between the two lines using:

```
m1 = ((coordinates_l1(2) - coordinates_int(2)) / (coordinates_l1(1) - coordinates_int(1)));
m2 = ((coordinates_l2(2) - coordinates_int(2)) / (coordinates_l2(1) - coordinates_int(1)));
angle = atand(abs((m2 - m1) / (1 + m2 * m1)));
```

After calculation we convert the value to string and display it on the robot's screen.

***Part 3: Give your robot the functionality to move to a defined point***

This functionality was split into two main functions. One was given the purpose to find the angles of the desired point, and the other's purpose is to move to the desired point using the angles found in the first function. In order to find the angles of the predefined point we used:

```
theta_1 = (atand(yw_coordinate/xw_coordinate)) - (acosd((xw_coordinate^2 +
    yw_coordinate^2+L1^2-L2^2)/(2*L1*(sqrt(xw_coordinate^2 + yw_coordinate^2)))));
theta_2 = -(180 - (acosd((L1^2+L2^2-xw_coordinate^2 - yw_coordinate^2)/(2*L1*L2))));
```

After finding the desired angles we can use our discrete controller from lab 5 to rotate the motors until they get to the desired angles for our predefined point. We did this in the following lines of code:

```
Tsim = 4;     % simulate over Tsim seconds
Ts = 0.01;    % sampling time
N= Tsim/Ts;   % number of steps to simulate
u1 = zeros(1,N); % plant A input
u2 = zeros(1,N); % plant B input
e1 = zeros(1,N); % error 1
e2 = zeros(1,N); % error 2
current_angle1 = zeros(1,N); % c1
current_angle2 = zeros(1,N); % c2
set_angle1 = angles(1);  % reference for A -- r1
set_angle2 = angles(2);  % reference for B -- r2
k=2;
```

```
while (k<=N)
    % --- Motor A controller
    current_angle1(k) = 180 + readRotation(mymotorA);
    e1(k) = set_angle1 - current_angle1(k);
    u1(k) = u1(k-1) + 0.412*e1(k)- 0.4114*e1(k-1);
    mymotorA.Speed = u1(k);

    %--- Motor B controller
    current_angle2(k) = 180 + readRotation(mymotorB);
    e2(k) = set_angle2 - current_angle2(k);
    u2(k) = u2(k-1) + 0.412*e2(k)- 0.4114*e2(k-1);
    mymotorB.Speed = u2(k);

    k = k + 1;
    pause(0.01)
```

***Part 4: Draw a straight line***

For this part, we are asked to draw a straight line between two predefined points in our workspace. Firstly, we find the slope between the two points. Then we calculate the y-intercept of the line.

```
slope = (coordinates_2(2) - coordinates_1(2))/(coordinates_2(1) - coordinates_1(1));
y_intercept = coordinates_2(2) - (slope*coordinates_2(1));
```

Then we use the slope and the y-intercept to define the straight line we need to draw. Once the line is defined we separate it into steps. The more steps used the more straight the line becomes, but this will take the arm a longer time to draw the line. The arm will draw a small part of the line for each step, so after all the steps are finished, the straight line will be complete. This is done in the following lines of code:

```
points_x = linspace(coordinates_1(1), coordinates_2(1), 25);
n = length(points_x);
points_y = zeros(1, n);
i = 0;
while i < n
    points_y(i+1) = slope*points_x(i+1) + y_intercept;
    i=i+1;
end

i = 0;
while (i < n)
    angles  = getAngle(points_x(i+1), points_y(i+1), L1, L2);
    SetAngles(mymotorA, mymotorB, angles);
    i=i+1;
end
```

**4. Demo (Video)**

https://www.dropbox.com/s/kuqmrb8ph9q64hi/ENEL%20484%20Project%20Demo.mp4?dl=0