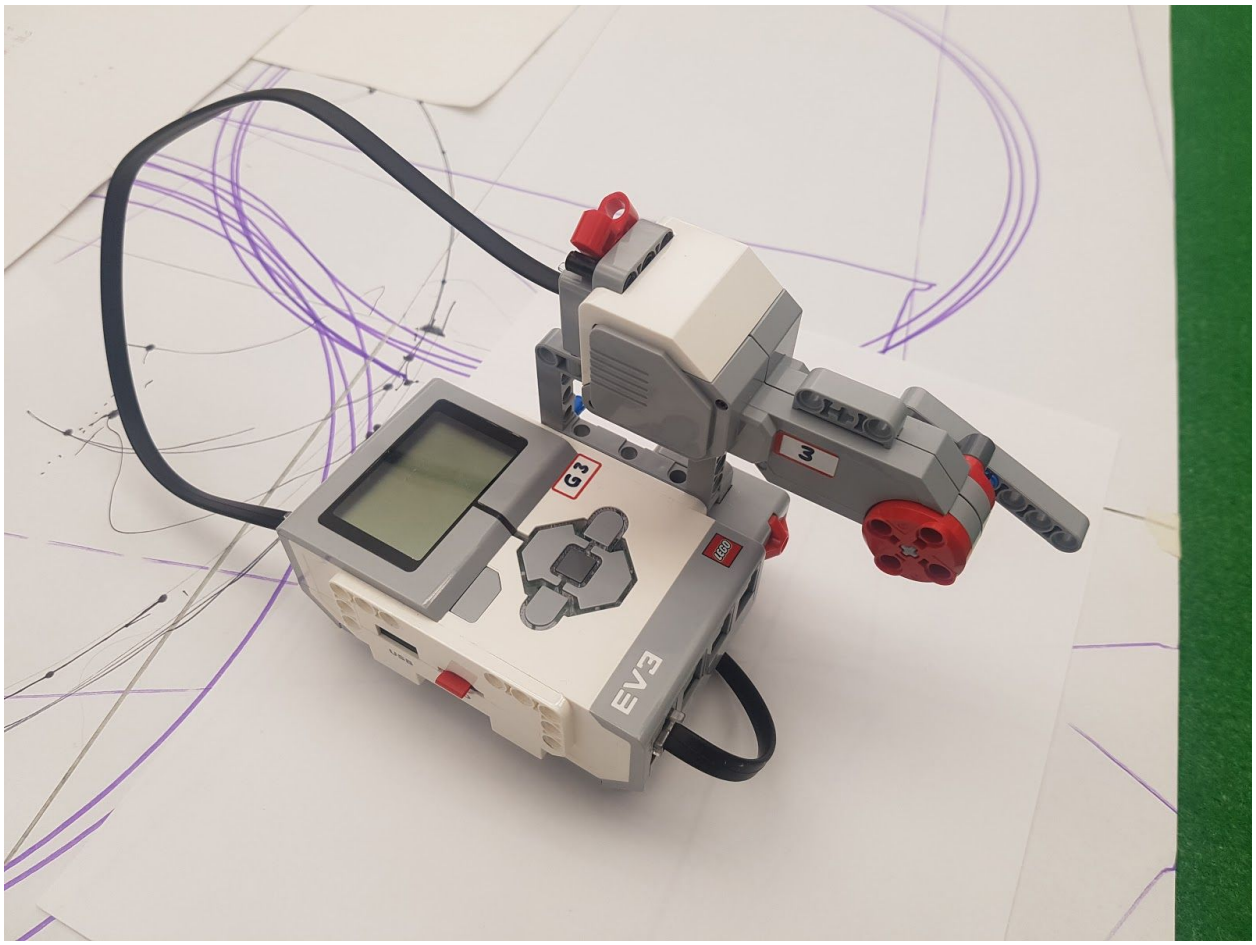
Motor Control, Robot arm Kinematics and Path planning

Group 06
Mohamed Ahmed, Chen Li, Canopus Tong

PID Control (part A)

Our first task was to implement the PID Control on a single servo motor which was connected to the Lego EV3 controller. The picture of the demo prototype was the following:



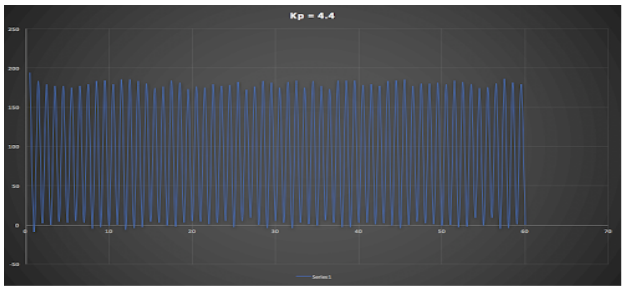
The formula that we used to implement the PID control was the following:

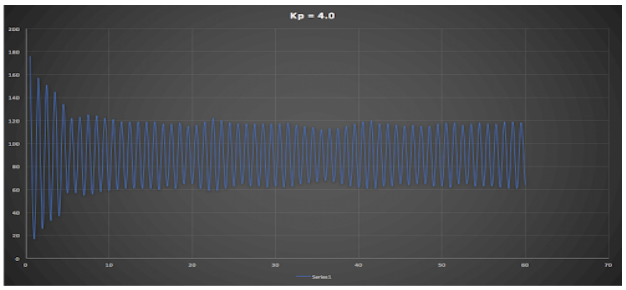
$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)d\tau + K_d\, \dot{e}(t)$$

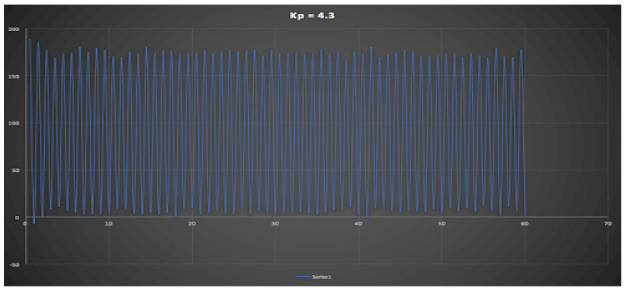
For the Proportional (P) controller, we set Ki and Kd to zero in order to test how the different P value will interfere with the stability of the system. The P (Kp) term is proportional to the current value. The controller multiplies the error by the Proportional Gain (Kp) to get the controller output.

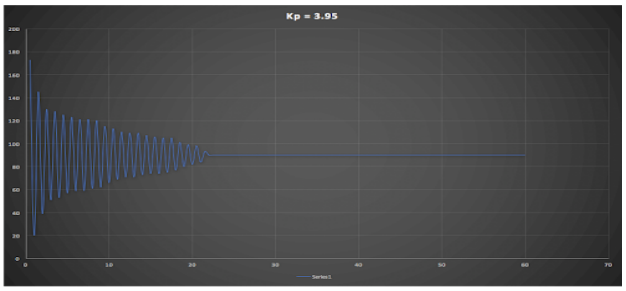
Data Analysis: Proportional

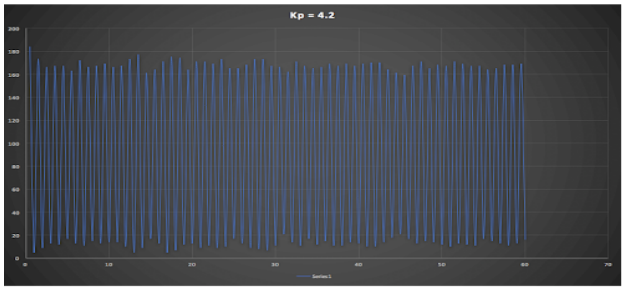
The data below were collected from the robot output. All graphs were have a test period of 60 seconds. To find the minimum Kd, the sampling period of 0.5 and sepint of 90 degrees was used in the experiment and different Kd values were chosen to be experiment: 4.4, 4.2, 4.1, 4.05, 4.0, 3.95, 3.9, 3.8, 3.5.

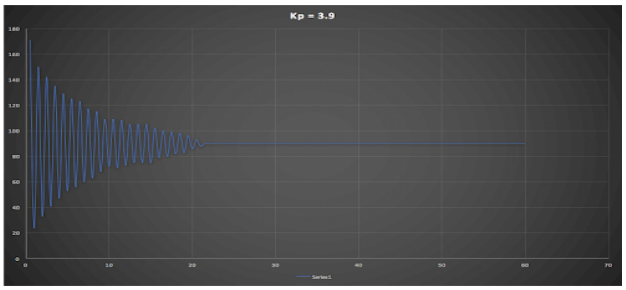


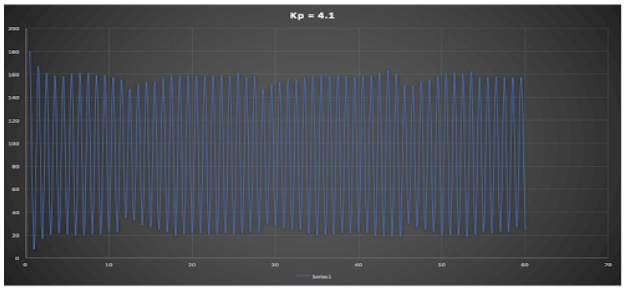


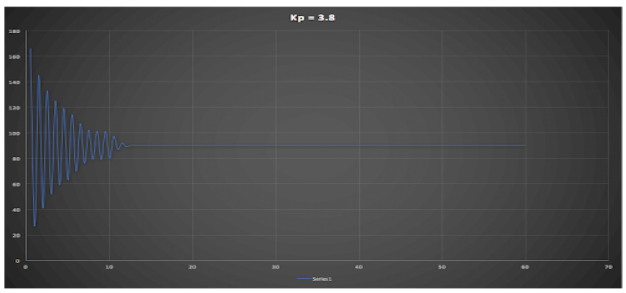


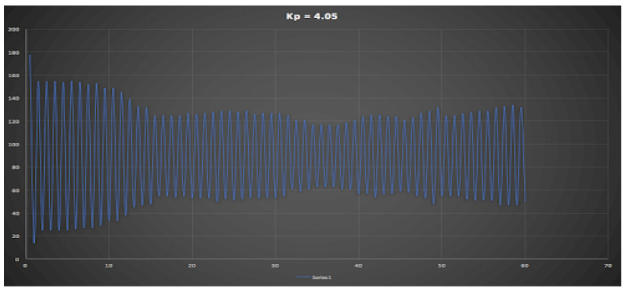


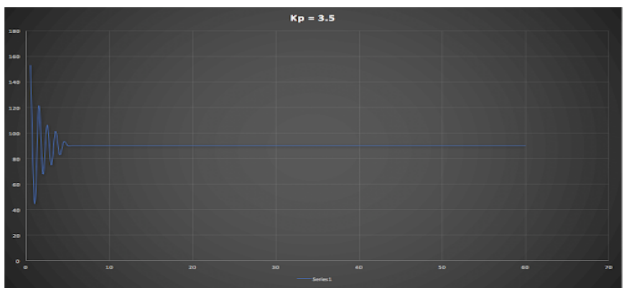












Observing the graphs above, Kp of 4.4 was the most unstable one and never converged in the 60 seconds and Kp of 3.5 was the most stable one and converged in 5.0 seconds. The minimum Kp that makes the system unstable was tested to be 4.0. For Kp of 4.0, as the graph suggested, convergence happened during the interval from 0 to 5 seconds but after 5 seconds the system stopped converging oscillation between the angle of 55 and 125 degrees. If the Kp was decreased to 3.95, the system converged in 22.0 seconds. From the observation of all the experiment, the lower the value of Kd, the faster the system converged, and vise versa.

There are some pros and cons for Proportional controller.
Pros:

1. Proportional controller was easy to test because there was only one variable to adjust.
2. Proportional controller needs less calculation in the program and will have a light program executing load on the controller.

Cons:

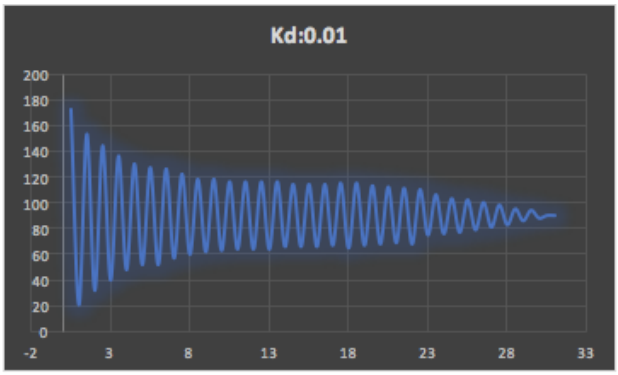
1. The system inevitably had oscillations given a sampling period of 0.5, which will lead to errors.
2. In some cases, the output was greater than the capability of the motor, which caused error to the program.

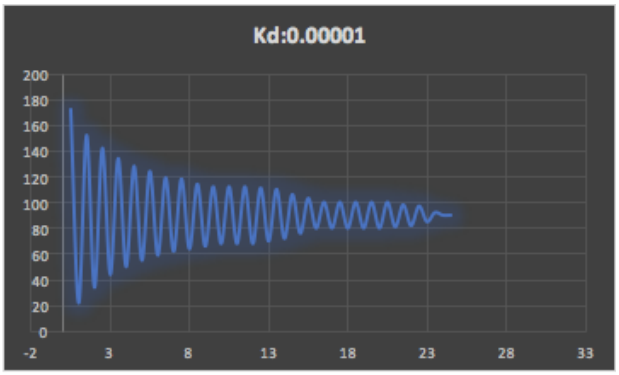
Proportional-Derivative (PD) controller

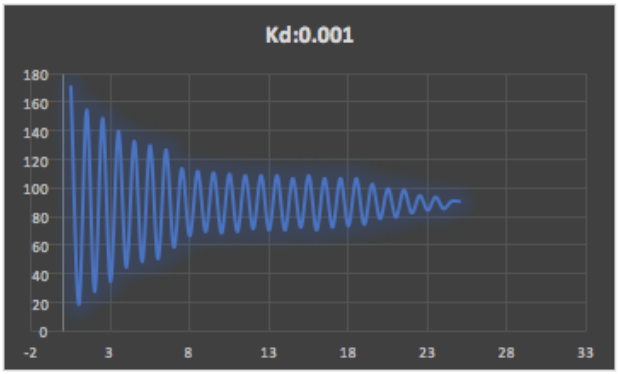
For Proportional-Derivative (PD) controller, the Kd value was set to zero in order to test how the different Kp and Ki value will interfere with the stability of the system. The Kd-term acted as an best-estimate of the future trend and he controller multiplies the error by the Proportional Gain and adds the derivative of error to proportional term.

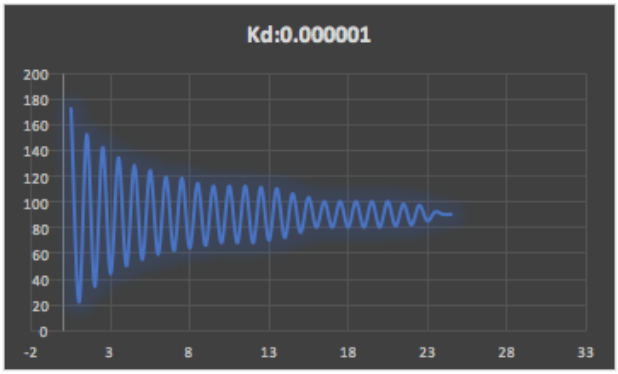
Data Analysis

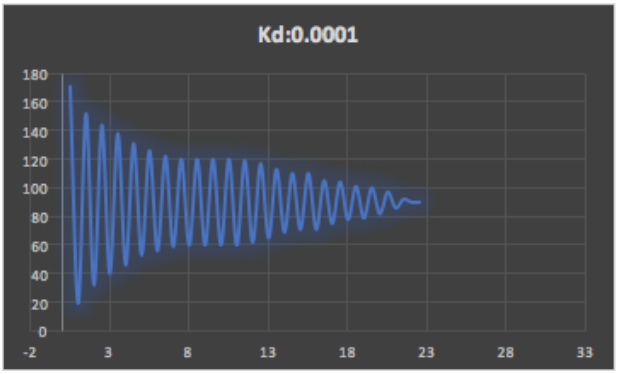
To find the effect of the the Kd-term, we used a set Kp of 3.95, sampling period of 0.5, setpoint of 90 degrees and adjusted Kd-term. The data below were collected from the robot output.











Inspecting the graph above, a Kd of 0.0001 was found to be the optimal Kd for this setup. With Kd = 0.0001, the system converged at an average time of 18.6 seconds out of 5 tests. Compared to the above Proportional Controller experiment with Kp = 3.95, Kd = 0 and converged time of 22.0 seconds, this PD Controller converged about 3.4 seconds quicker. The Kd-term improved stability, the oscillation was smaller than the P Controller and the funnel of the graph was a lot narrower in the PD graph. The steady-state error was about $\mp$ 1. In other words, if the setpoint was 90 degrees, sometimes the system converged to 89 or 91 degree but never reached exactly

90 degrees. Moverover, it was determined that Kp of 3.5, Kd of 0.00001 and sampling period of 0.1 seconds to have the fastest convergence rate.

There are some pros and cons for Proportional-Derivative controller

Pros:

1. Compared to the Proportional controller, this Proportional-Derivative (PD) controller had a faster convergence.

Cons:

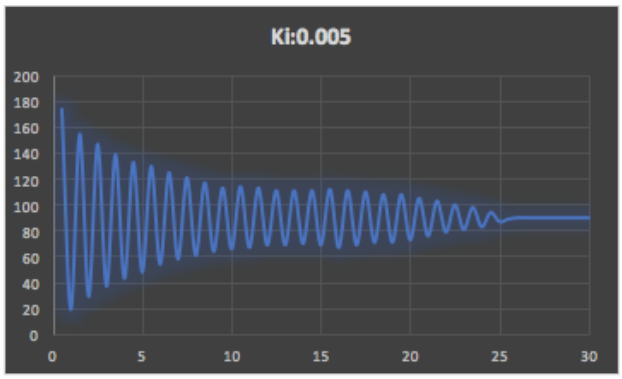
1. Similar to Proportional controller from the plots, the output had some errors inevitably.
2. When Kd was larger enough, the system might enlarge the errors and took longer to converge than the Proportional controller.

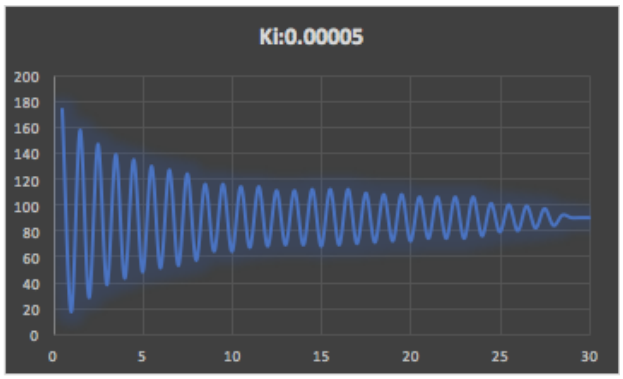
Proportional-Derivative-Integral (PID) controller

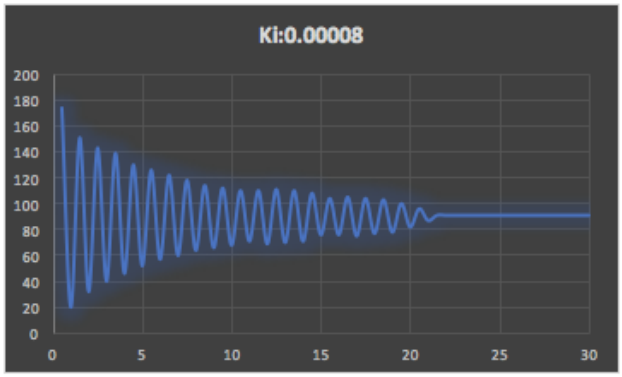
The Ki-term accounts for past values. The integral term ensures the state error is zero.

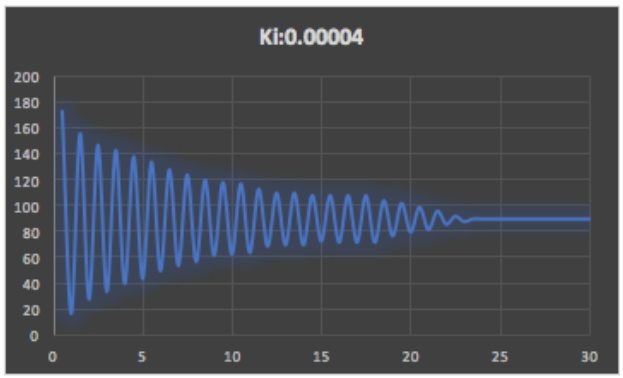
Data Analysis

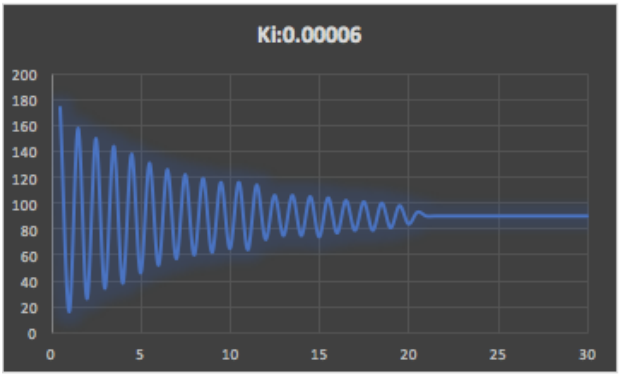
The plots below were deviated from the test data collected from the robot output by excel. Kp was set to be 3.95. Kd was set to be 0.00001. Sampling for the test below are 0.5 seconds.

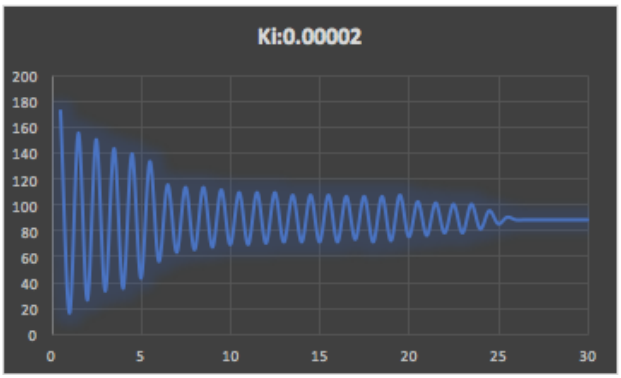












The Ki-term helped to improve the steady-state error, however convergence sometimes took longer than the PD controller. The optimal Ki value was found to be 0.00004. The Ki-term improved stability. With Ki = 0.00004, compared to Ki = 0, the funnel shape of the graph was a lot more narrower, meaning that the oscillation became smaller quicker. The Ki-term helped the controller to reach the setpoint more often, but not every test. The optimal tuning of the controller was determined to be Kp = 3.5, Ki = 0.00004, Kd = 0.00001 and sampling period of 0.1 seconds.

There are some pros and cons for Proportional-Derivative-Integral controller

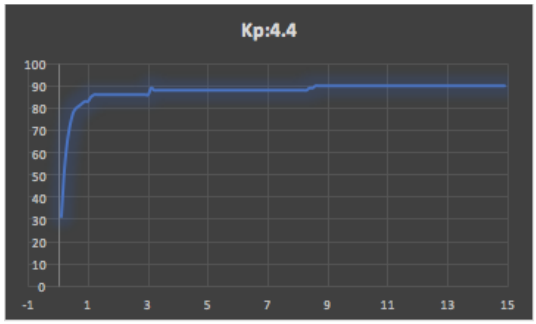
Pros:

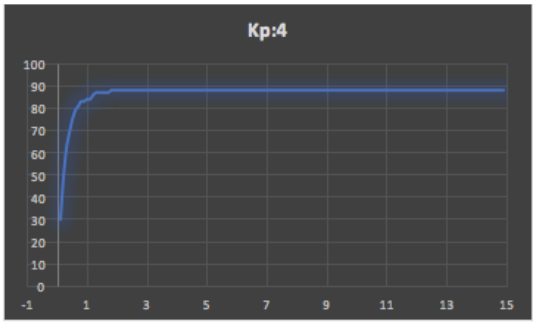
1. Steady-state error was lower
2. Improved stability
3.

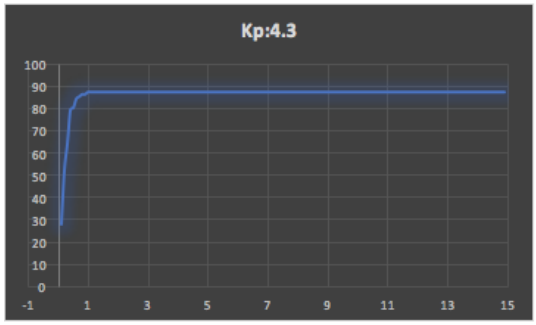
Cons:

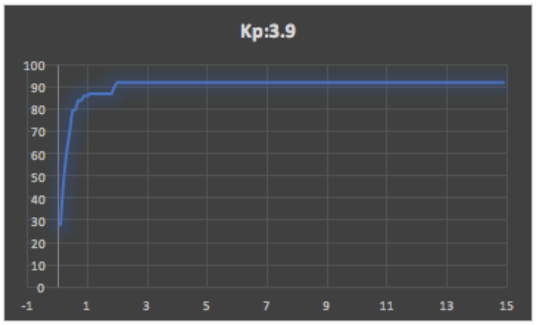
1. The PID controller itself, from our self-tuning experience, is harder to tune it into a steady condition.

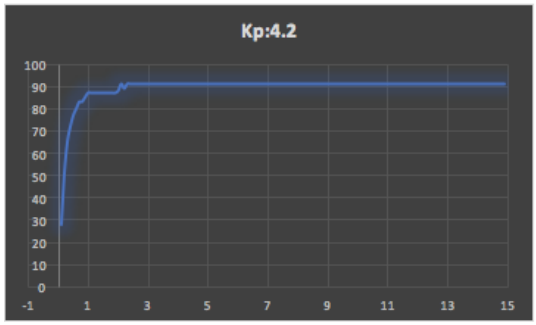
Effect of sampling period

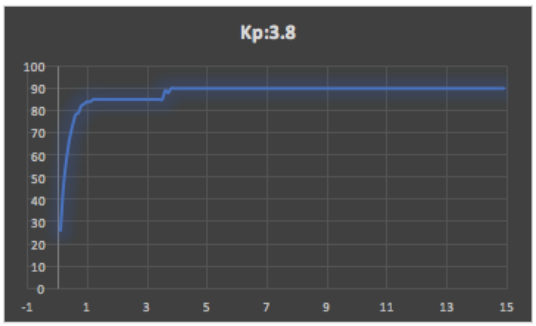


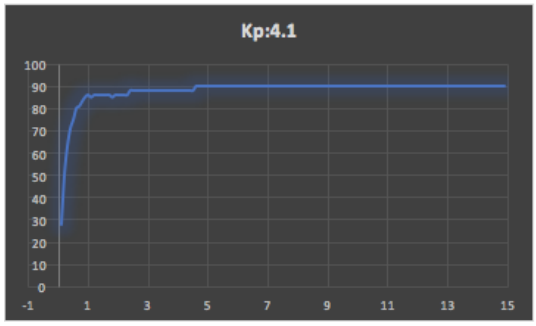


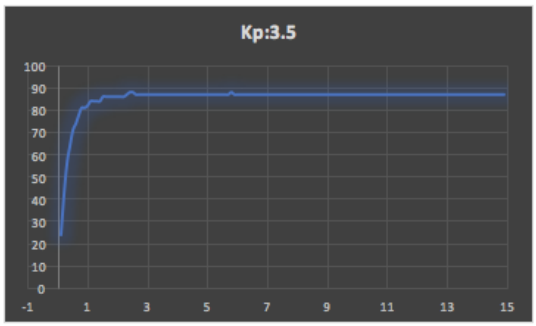


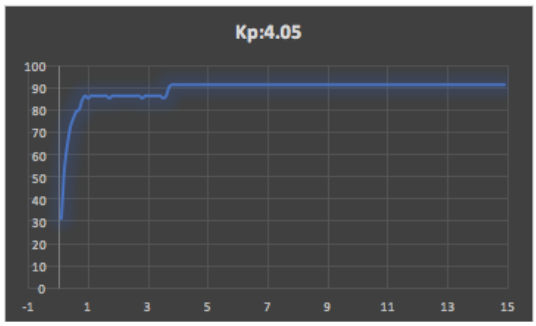












For the experiment above, this setup was the same to the Proportional controller setup except a sampling period of 0.1 seconds was used, . Essentially the lower the sampling period, the less oscillation and quicker it coveraged.

Effect of Actuation Saturation

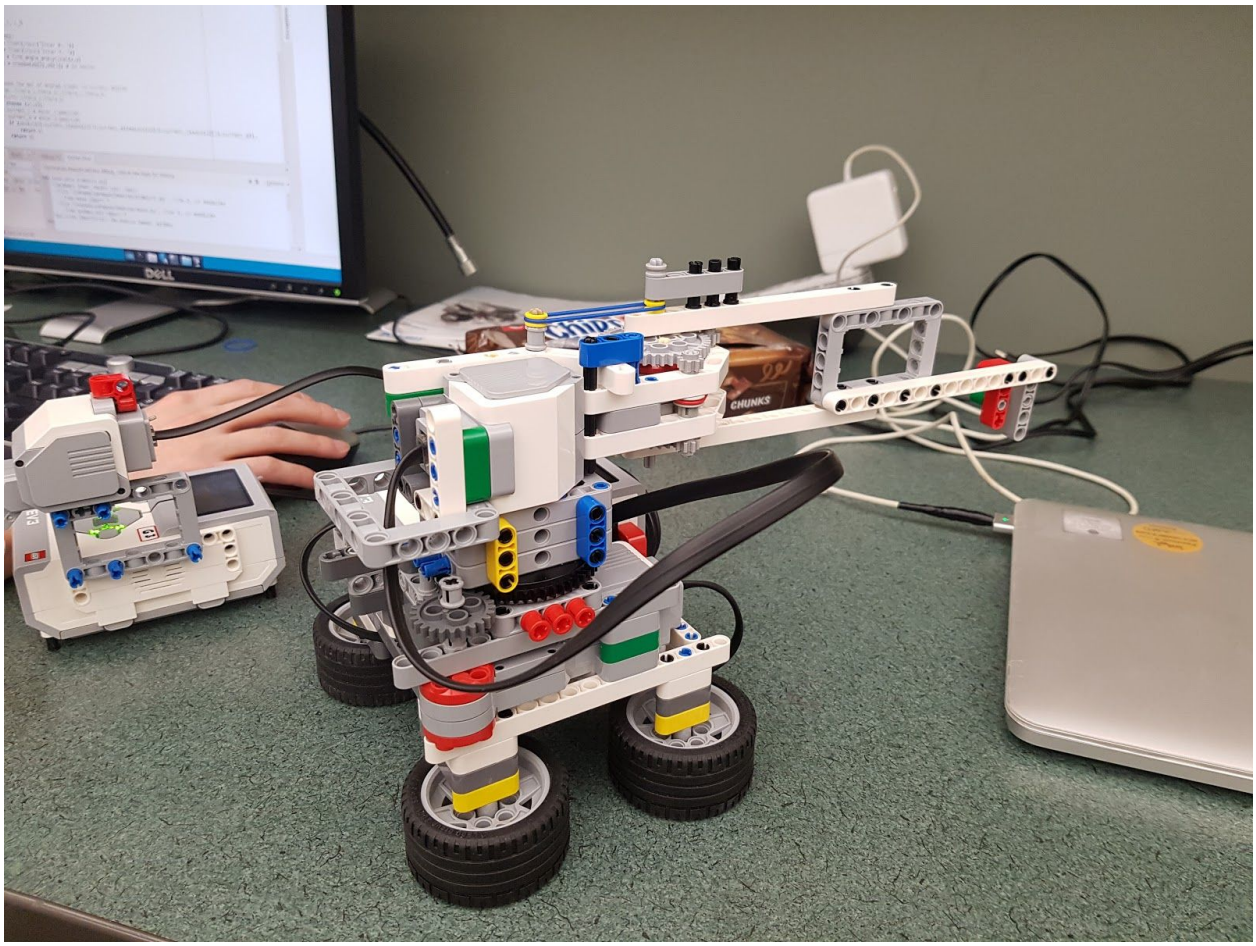
In the program, we had to add an if statement to prevent unhandled termination the output to exceed the motor's maximum capacity. For example if Kp was too big , the Proportional controller will produce a high output that overshoot the well above the value which the system can withstand. Below is a simple solution to the problem:

```
if output > 999:
     output = 999
elif output < -999:
     output = -999
```

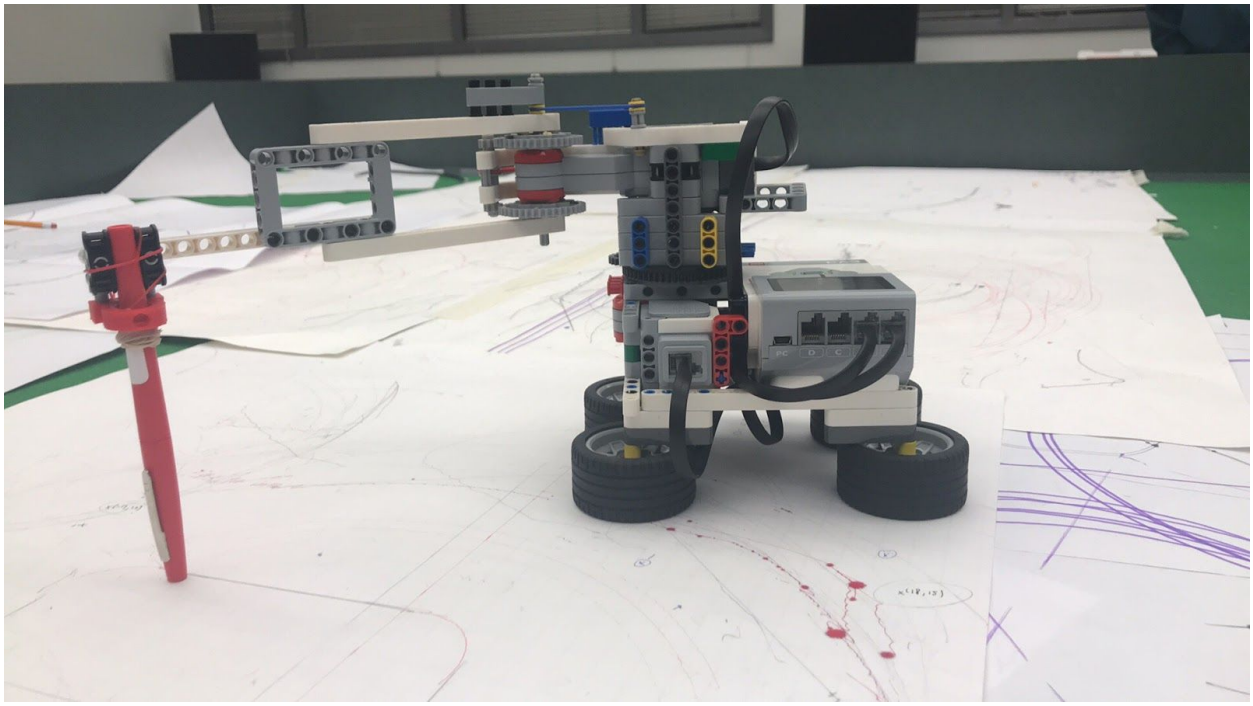
2-DOF Arm

Build of 2-DOF Arm

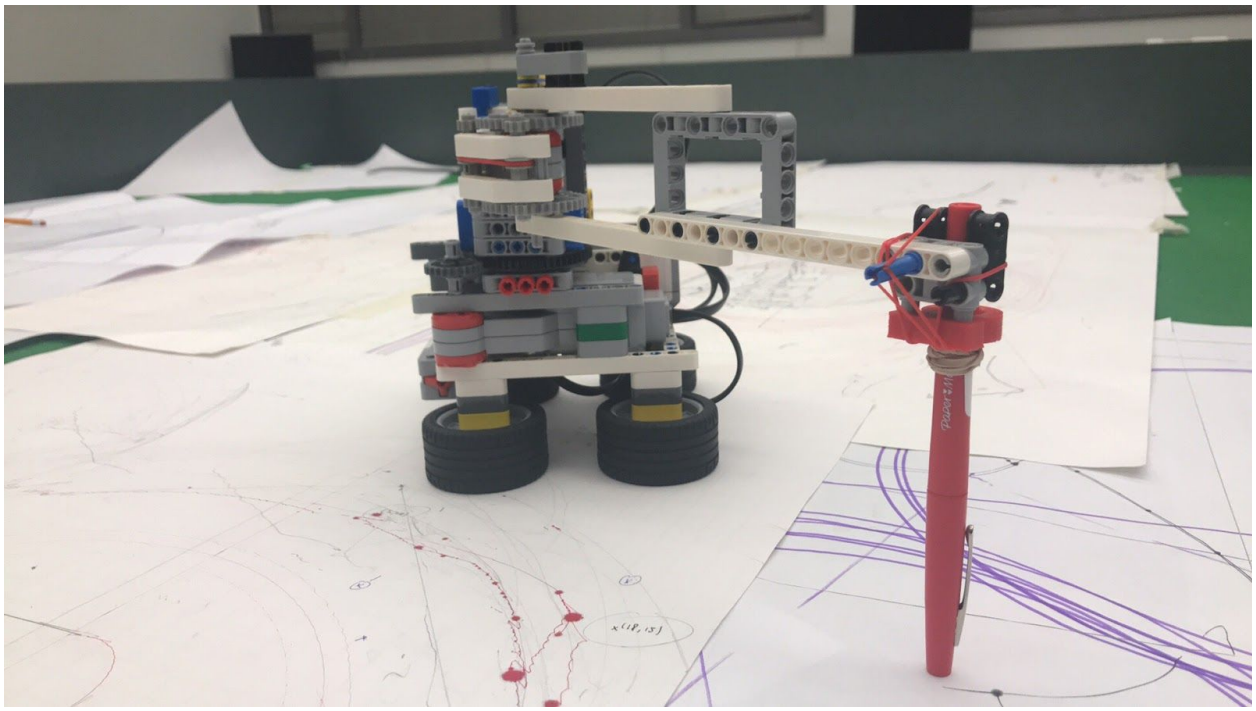
The design of our robot is pictured below:



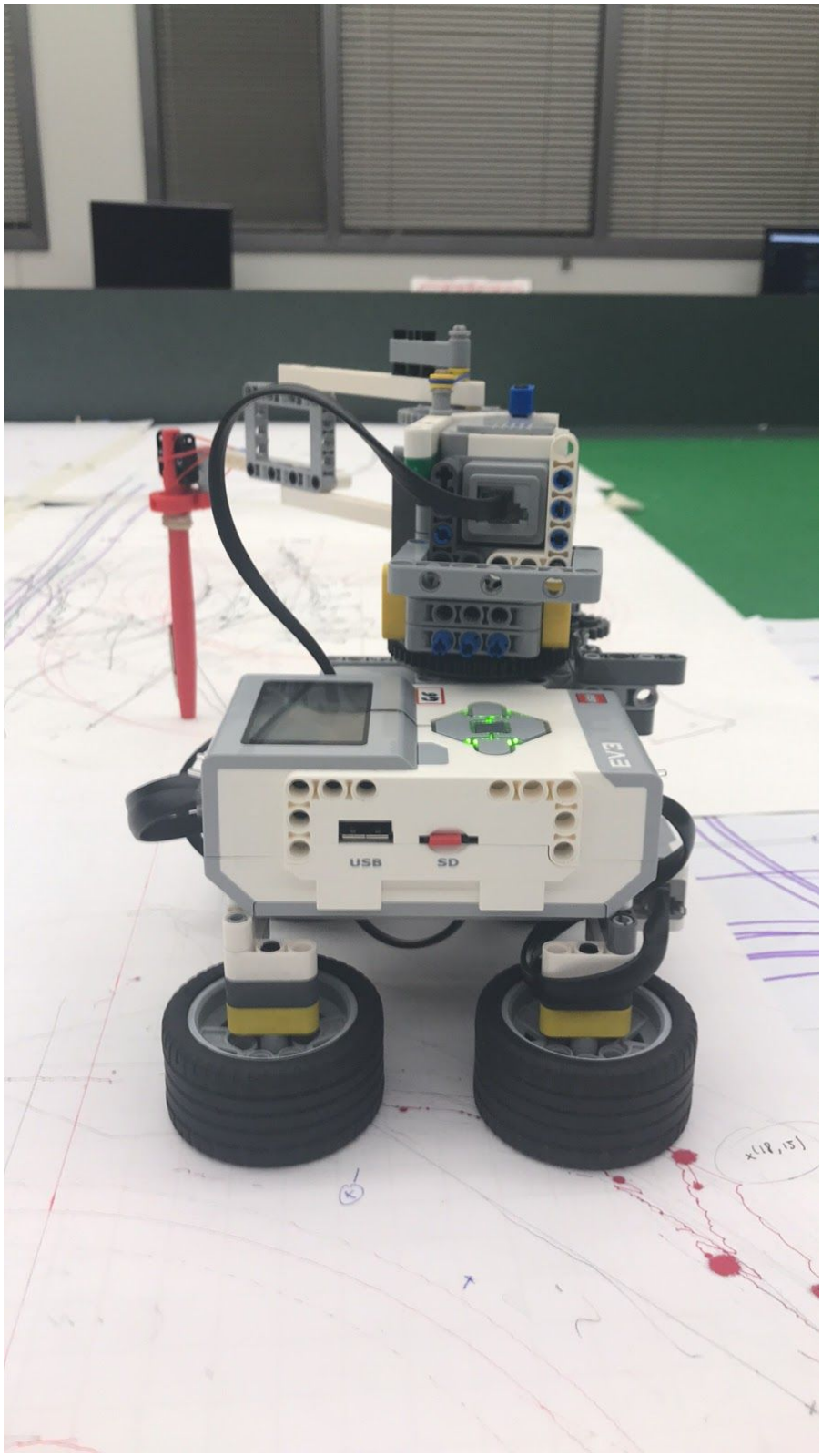
Overview



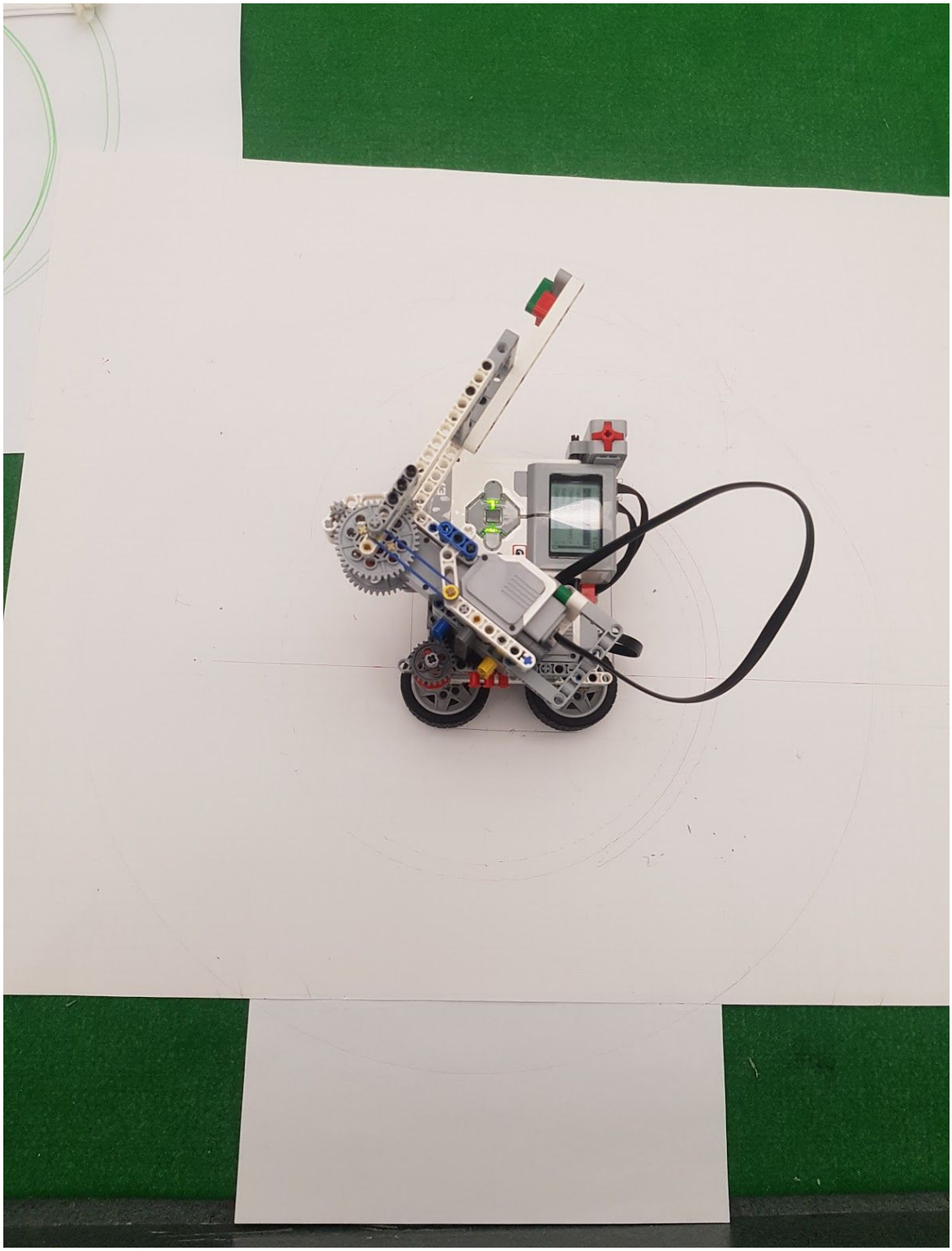
Right side:



Left side:



Back side:



Top view:

<1>. There are different types of linkages that can provide the same motion. The current linkage provides 2 degrees of freedom by two revolute joints in the same parallel space.

The current linkage is to mount the two motors horizontally but not at the same level, the mechanism is to link the two motors sequentially by gears.

1. The pros of this linkage are as follows:
(1) . The bottom motor can be fixed to the bottom base without any movement to the motor itself needed and because of this the cable connected to the bottom motor is easily hidden and it will not affect the movement at all.

(2). The combination of gears provided a much more sturdy construction than link the arm directly to the base-motor (link the arm directly to the base-motor will lead to a plenty of error in the vertical orientation for the reference of motor moving orientation)

(3). The gear mechanism can provide more torque force compared to the design of link the arm directly to the base-motor and it is more precisely when working (which can fix the factory error from the electric motor itself to some degree)

2. Cons of the gear design: Might lead to more errors because the connection between gears is not perfectly linked, it indeed has some loose space. The loose space can lead to errors in addition to the error of the motor itself.

3. Compare to the belt-moving construction, The gears tend to provide more precise movement and tend to be fewer errors compared to the belt design. Since the belt is linked by multiple joints and each linkage between two joints have errors because of the empty space. However, the belt construction can provide the movement which can power the upper arm with its axle located in a different place without moving the motor. Moreover, the belt construction can power two or more arms (in different axle position) in synchronize steps with only one motor.

However, the belt design can provide fewer errors in the horizontal orientation in reference to the motor spinning orientation. Our gear design alters a lot in joints to reduce the errors stated above.

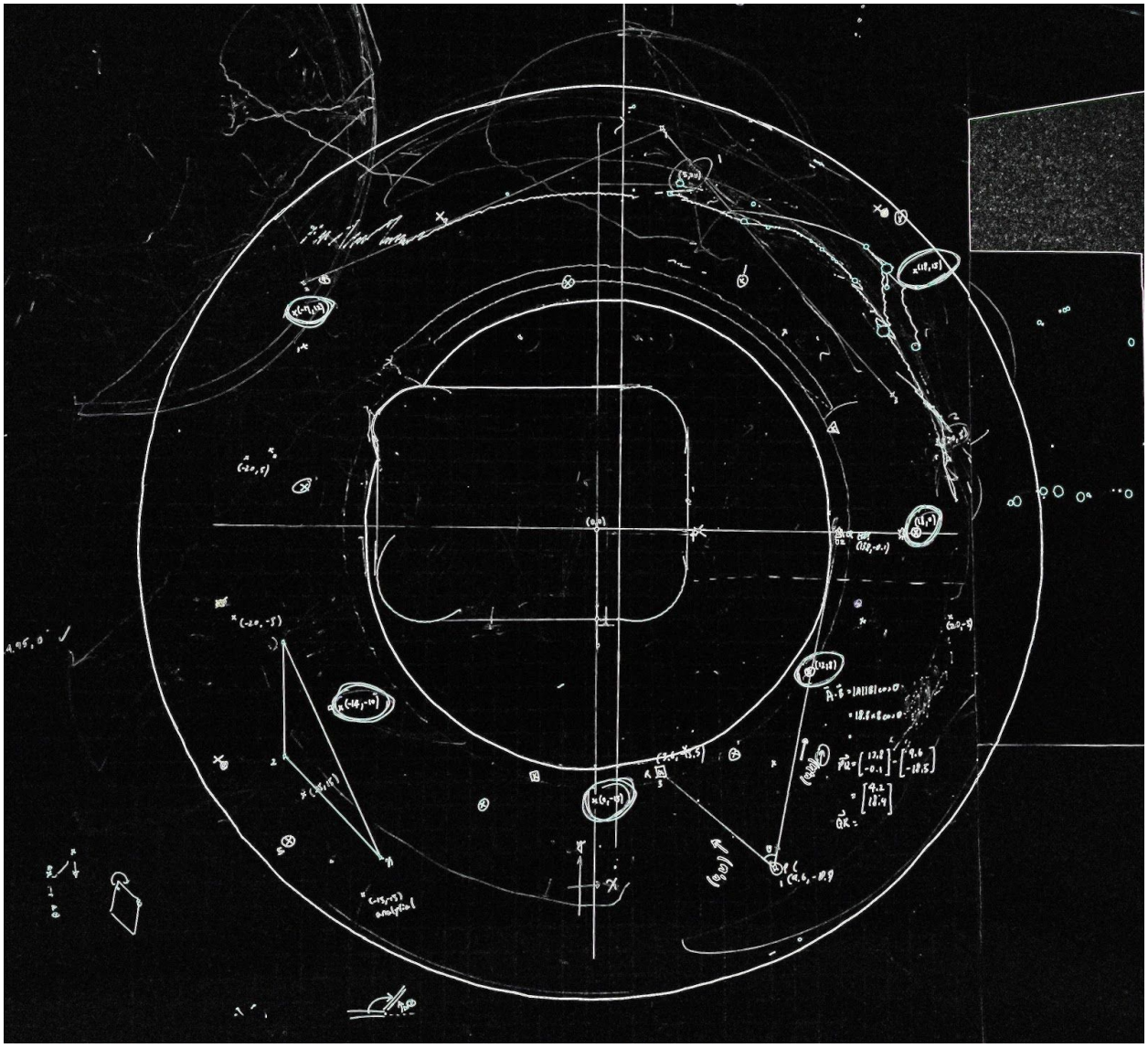
4. Compared to cylindrical arm robot design, the gear design has less flexibility and it has several dead areas which cannot be reached by the top, The gear design also cannot form a cylinder workspace. Moreover, the gear design needs more calculation in order to make the top of the arm to reach its desired cartesian coordination.

However, cylindrical arm robot design needs a powered hydraulic arm or gear racks to power the upper arm. In our lego kits, we do not have the corresponding parts to design the cylindrical arm.

5. Compared to the linkage of putting motors serially in arms, this linkage put the motor for the inner arm at the base, which provides a lightweight and faster movement and a more complex transmission. Just like SCARA which minimizes loading of motors from the weight of other motors.

Work space

<2> The workspace is plotted on the paper using bold black pen as below



Above is the color-inversed picture for the workspace.

The workspace of this robot forms a set of concentric spheres, it has a spherical manipulator, which is similar to Stanford's arm. However, it only has two revolutes (RR).

As we can see from the picture (Top view), the robot is able to draw within the area of a level Annulus shape.
With the center as the center of the large gear (number of teeth is 56) connected to the base motor. The radius of the outer circle is 25.5cm and the radius of the inner circle is 13.1cm.

When the robot is drawing at the outer circle, it stretches the upper arm to the longest position, a.k.a the upper arm is parallel (180 degrees) to the base of the upper motor which powers it.

When the robot is drawing at the inner circle, it stretches the upper arm to the shortest position it could reach to. i.e. the upper motor moves 20 degrees towards the base of the upper motor. when the robot is performing the inner circle, the side close to the port A of the EV3 of the inner circle may yield to the wheel located on the ground which is to fix the robot firmly to the ground for 5 degrees.

The robot is designed to draw on the plain surface, so the side view of its workspace is formed by a line of horizontal base.

Forward Kinematics (part c)

Q1. The robot coordinate position is shown in the figure below. As shown in the figure c1, the starting position for the robot is (24.95,0). Length of the first(inner) arm link is 6.65 and the link of the second(outer) arm link is 18.3

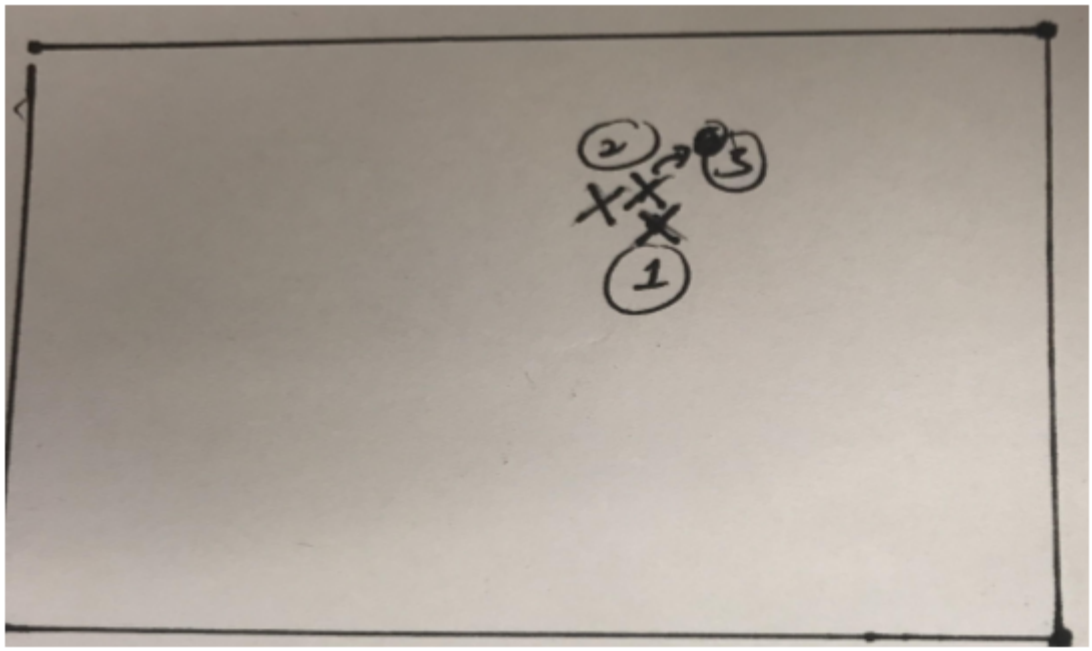
Q2.
Implementations: Name of the function, find_xy.py. This function receives two angles from a user. The first angle being the angle for the inner motor and the second angle for the outer motor. The function then calls move.py which receives the angles of the two motors as parameters. When move.py finishes executing, it moves the both arms to their corresponding joint angles and returns the current rotations of the motors. As shown in figure C2, We used the motor rotations to calculate the end effector.

$$X\_coord = l_1\cos(\theta_1) + l_o\cos(\theta_1 + \theta_o)$$

$$Y\_coord = l_1\sin(\theta_1) + l_o\sin(\theta_1 + \theta_o),$$

Figure C2: Equations for X and Y coordinates

According to table C2_2, the maximum error distance is 2. This error is consistency with the error that we allowed in our implementation of the PID control, which is within +2. Overall, the accuracy of the our robot is pretty good since it's error is small. We also measured the repeatability of our robot. At the beginning of every run, we calibrate our robot to initial position of (24.95,0). We moved the first angle -180 degrees and the second angle -90, then repeated this move for 3 times. We noticed that when we first run, the robot arm moved to (-7.3,18.1). For the second run, the robot moved to (-7.5,18.3) and for the last run the arm moved to (-7.3,18.4). We think that the change in positions was due to a slack on the second arm link.



Repeatability Figure: 1st point (-7.3,18.1), 2nd point (-7.5,18.3) , 3rd point (-7.3,18.4)

Angle_1	Angle_1_moved	Angle_2	Angle_2_moved	X_pos_calc	Y_pos_calc	X_pos_ther	Y_pos_ther	X_pos_measured	Y_pos_measured
225	224.57	100	98	9.79	-15.79	10.29	-15.2	7.5	-19
-180	-179.14	-90	-88	-7.56	18.18	-6.65	18.3	-6.8	16.7
45	44.14	-45	-43	23.07	5	23	4.7	22	5.4
-333	-332.14	79	77	0.88	20.61	1.19	20.8	1	21.5
190	189.85	56	54	-14.62	-17.57	-13.99	-17.87	-16.7	-17.3
-128	-127.71	-68	-66	-21.85	-0.92	-21.68	-0.2	-21.7	-3.7
-200	-199.29	78	76	-16.32	-13.1	-15.95	-13.24	-17.7	-12.1
125	124.29	-85	-83	10	17.57	10.2	17.21	8.5	18
189	188	39	38	-19.26	-14.14	-18.81	-14.64	-20.5	-12.8
30	30.42	30	28	15.74	18.61	14.9	19.17	14.6	19.2

Table C2_1: Data for Find_x_y function

Error for X	Error for Y	Errofor Angles_1	Error for Angle_2
measured_X - cal_X	measured_Y - cal_	measured - cal	measured - cal
2.29	3.21	0.43	2
0.76	1.48	0.86	2
1.07	0.4	0.86	2
0.12	0.89	0.86	2
2.08	0.27	0.15	2
0.15	2.78	0.29	2
1.38	1	0.71	2
1.5	0.43	0.71	2
1.24	1.34	1	1
1.14	0.59	0.42	2

Table C2_2: Errors for data
Note it's calculated using Absolute value(measured - actual(cal))

Q3

Distance

Implementation: Name of the function distance.py. In this function, A user moves the end effector to a first point, clicks a touch sensor for recording this point, then the user moves the end effector to a second point and clicks a touch sensor for recording. After every clicks of the touch sensor, the program reads motor angles and stores in a list. We use this list to calculate the x,y positions of the end effectors for the first and second points. We use the distance formula shown in figure C3, to calculate the distance between the second and first point.

The maximum distance error is 2.48. This deviation is due to the slack on the second arm as the link in this arm is not highly stable. In addition, this deviation was also due to an error in the end effector to the first point as shown the table C3_1 below, the measured x,y position is (13.5,5.9) while the actual x,y position is (11.30,8.06). Thus, the error in x_position is 2.2 and the error in y_position 2.16. Hence for that reason the error is high.

Figure C3: distance formula

Dist_cal	Dist_measured	First_point_x_measured	First_point_y_measu	Second_point_x_measured	Second_point_y_measured	First_point_x_cal	First_point_y_cal	Second_point_x_cal	Second_point_y_cal
23.85	24.4	10.7	11.5	9.9	-12.9	11.27	11.42	10.18	-12.4
5.16	6.12	-16.5	10.1	-18.3	4.2	-16.53	8.72	-18.29	3.87
35.66	37.2	17.2	17.7	9.8	-18.5	16.32	18.84	11.66	-16.51
25.24	23.8	-15.5	13.8	8.2	14.3	-16.11	12.22	9.08	13.92
5.97	7.05	3.6	-13.5	-3.4	-13.7	2.77	-13.6	-3.21	-13.6
8.58	6.1	13.5	5.9	13.8	-0.1	11.3	8.06	13.88	-0.12
15.05	15.2	-14	-10	0	-15	-14.62	-9.28	-0.74	-15.12
33.57	35.3	-17	12	18	15	-16.33	12.76	17.14	15.34
9.64	10	18	0	12	-8	18.19	-1.4	11.95	-8.76
25.37	26.2	12	-8	-14	-10	11.95	-8.76	-13.39	-9.84

Error: Distances
Dist_measured and dist_cal
0.55
0.96
1.54
1.44
1.08
2.48
0.15
1.73
0.36
0.83

Figure C3_1

Angle

Implementations: Name of the Functions angle.py. A user moves the end effector to three points clicking a touch sensor after each point to record the motor angles. The first point is an intersection of a line on the second point and another line on the third point. We calculate the x,y position using equation in figure C2 then store the x,y coordinates on a list. We use this list of coordinates to find out a vector on each line, and the distance between second and the first point as well as the distance between the third and the first point. Using these vectors and distances, we compute the dot product.Therefore, we compute the angle between the two lines.

As shown in the data table below, the maximum error is 2. Overall, the data is consistent with the degree of freedom of error that we allowed in PID control.

First_point_x_cal	First_point_y_cal	Second_point_x_cal	Second_point_y_cal	Third_point_x_cal	Third_point_y_cal
9.9	-17.98	13.87	-0.43	3.57	-13.41
-15	-13.55	-16.57	-6.44	-10.94	-17.49
-15.98	-7.52	-16.23	-12.4	-10.76	-17.48
-10.54	-17.73	-15.71	-13.37	-16.47	-6.69
-1.62	-15.5	-13.9	-9.73	12.39	-7.11
-12.97	-10.21	-0.9	-15.34	11.37	-9.67
-12.97	-10.21	-0.81	-14.9	17.37	1.48
11.37	-9.67	18.23	-0.72	-16.55	11.8
17.6	15.24	11.58	-8.37	17.87	-0.89
18.05	14.71	17.91	14.77	17.78	-0.19
0.71	24.73	-16.5	24.83	18.06	15.19

Figure C3_2: Data for Actual values(i.e calculated x and y)

Theta_cal	Theta_measured	First_point_x_measured	First_point_y_measured	Second_point_x_measured	Second_point_y_measured	Third_point_x_measured	Third_point_y_measured
59.8	64	9.6	-18.5	13.8	-0.1	3.6	-13.5
146.51	138.5	-17	-12.5	-17.2	-6.4	-11.6	-18.1
30.59	23	-17.2	-6.4	-17	-12.5	-11.6	-18.1
21.6	19	-11.6	-18.1	-17	-12.5	-17.2	-6.4
123.9	130	0	-15	-14	-10	12	-8
24.29	26	-14	-10	0	-15	12	-8
141.56	158	12	-8	0	-15	18	0
86.49	85	18	15	18	0	-17	12
15.03	16	18	15	12	-8	18	0
21.64	23	12	-8	18	15	18	0
116.66	116	0.2	14.2	-17	12	18	15

Figure C3_3: Data for Measured value

Error for Theta	Error for First point		Error for 2nd point		Error for 3rd point	
mea_theta - cal_theta	measured_x - x_cal	measured_y - y_cal	measured_x - x_cal	measured_y - y_cal	measured_x - x_cal	measured_y - y_cal
4.2	0.3	0.52	0.07	0.33	0.03	0.09
8.01	2	1.05	0.63	0.04	0.66	0.61
7.59	1.22	1.12	0.77	0.1	0.84	0.62
2.6	1.06	0.37	1.29	0.87	0.73	0.29
6.1	1.62	0.5	0.1	0.27	0.39	0.89
1.71	1.03	0.21	0.9	0.34	0.63	1.67
16.44	24.97	2.21	0.81	0.1	0.63	1.48
1.49	6.63	24.67	0.23	0.72	0.45	0.2
0.97	0.4	0.24	0.42	0.37	0.13	0.89
1.36	6.05	22.71	0.09	0.23	0.22	0.19
0.66	0.51	10.53	0.5	12.83	0.06	0.19

Figure C3_4: Data for the Errors

Inverse Kinematics in 2D (part D)

Two approaches were being implemented for the two-joint robot arm, a numerical solution and the analytical solution. In order to solve for the two angles numerically given the x-y coordinate, Newton's method was used in this specific application and its implementation will be described briefly.

Solving a system of nonlinear equation: Newton's method

Essentially two basic equations were being used here:

$$X = l_i\cos(\theta_i) + l_o\cos(\theta_i + \theta_o), \quad (1)$$

$$Y = l_i\sin(\theta_i) + l_o\sin(\theta_i + \theta_o), \quad (2)$$

where x, y are the desired cartesian coordinate, l_i, l_o are the inner arm and outer arm length and θ_i, θ_o are the inner angle and outer angle of the robot arm correspondingly. Newton's method was fundamentally a root-finding algorithm. Therefore (1), (2) must be rearranged such that the left-hand side of the equation must be equal to zero. In this case, the two functions will be

$$F_x: l_i\cos(\theta_i) + l_o\cos(\theta_i + \theta_o) - X, \quad (3)$$

$$F_y: l_i\sin(\theta_i) + l_o\sin(\theta_i + \theta_o) - y \quad (4)$$

$$F = \begin{bmatrix} F_x \\ F_y \end{bmatrix} \quad (4.1).$$

After that the inverse Jacobian matrix of F_x and F_y will be needed to be calculated. First, the Jacobian matrix can be found by

$$\begin{bmatrix} \partial F_x/\partial\theta_i & \partial F_x/\partial\theta_o \\ \partial F_y/\partial\theta_i & \partial F_y/\partial\theta_o \end{bmatrix} \quad (5),$$

and in this case Jf,

$$\begin{bmatrix} -l_i\sin(\theta_i) - l_o\sin(\theta_i + \theta_o) & -l_o\sin(\theta_i + \theta_o) \\ l_i*\cos(\theta_i) + l_o\cos(\theta_i + \theta_o) & l_o\cos(\theta_i + \theta_o) \end{bmatrix} \quad (6),$$

then the inverse Jacobian matrix Jf^{-1},

$$\begin{bmatrix} l_o\cos(\theta_i + \theta_o)/\det(Jf) & l_o\sin(\theta_i + \theta_o)/\det(Jf) \\ (-l_i*\cos(\theta_i)+l_o\cos(\theta_i + \theta_o))/\det(Jf) & (-l_i\sin(\theta_i)-l_o\sin(\theta_i + \theta_o))/\det(Jf) \end{bmatrix} \quad (7).$$

However, Jf can be can singular is some cases which cause det(Jf) to be zero, and this will obviously create a math error. For this 2-joint robot arm setup, singularities will happen whenever the outer arm is parallel to the inner arm; for instance, the outer arm has an angle of zero with respect to the inner arm. However this problem was overcome by adding a very small value to the θ_o whenever it is equals to $n\pi$, for $-\infty > n > \infty$;

```
if θo%pi==0:
    θo+=0.001.
```

Then Jf^{-1} and F will be substituted into the core equation

$$X_1 = X_0 - Jf^{-1}(\theta_i, \theta_o) F(\theta_i, \theta_o) \quad (8),$$

where X_0 is the initial motor's angles

$$X_0 = \begin{matrix}[\theta_i] \\ [\theta_o]\end{matrix} \quad (8.1).$$

This equation will then be executed recursively,

$$X_1 = X_0 - Jf^{-1}(\theta_i, \theta_o) F(\theta_i, \theta_o)$$

$$X_2 = X_1 - Jf^{-1}(\theta_i, \theta_o) F(\theta_i, \theta_o)$$

$$X_3 = X_2 - Jf^{-1}(\theta_i, \theta_o) F(\theta_i, \theta_o)$$

etc...

until the norm(F(θ_i, θ_o)) <= error. For this application, the error is set to 10^-8. After all experiments on Newton's method, the solutions calculated always converges to the solution that was closer to X_0 (the current angles of the arms).

The analytical approach: Atan2 and Cosine Law

The analytical solution is rather tricky to derive, since redundancy occurs when dealing with arcsin and arccos functions. There were two solutions to reach a specific location in a cartesian space; both solutions will be solved in our program and the solution that is closer to the current arm orientation will be chosen and returned.

Firstly, the outer arm angle will be determined by the arccos function provided by the lecture ppt.,

$$\theta_o1 = \arccos((X^2+Y^2-l_i^2-l_o^2)/(2l_il_o)) \quad (9)$$
$$\theta_o2 = -\arccos((X^2+Y^2-l_i^2-l_o^2)/(2l_il_o)) \quad (10).$$

θ_o1 will be the positive angle and θ_o2 will be the negative angle for the outer arm.

Secondly, the inner arm angle will be needed to be solved using atan2 and the cosine law.

Case 1:

when θ_o1 is positive

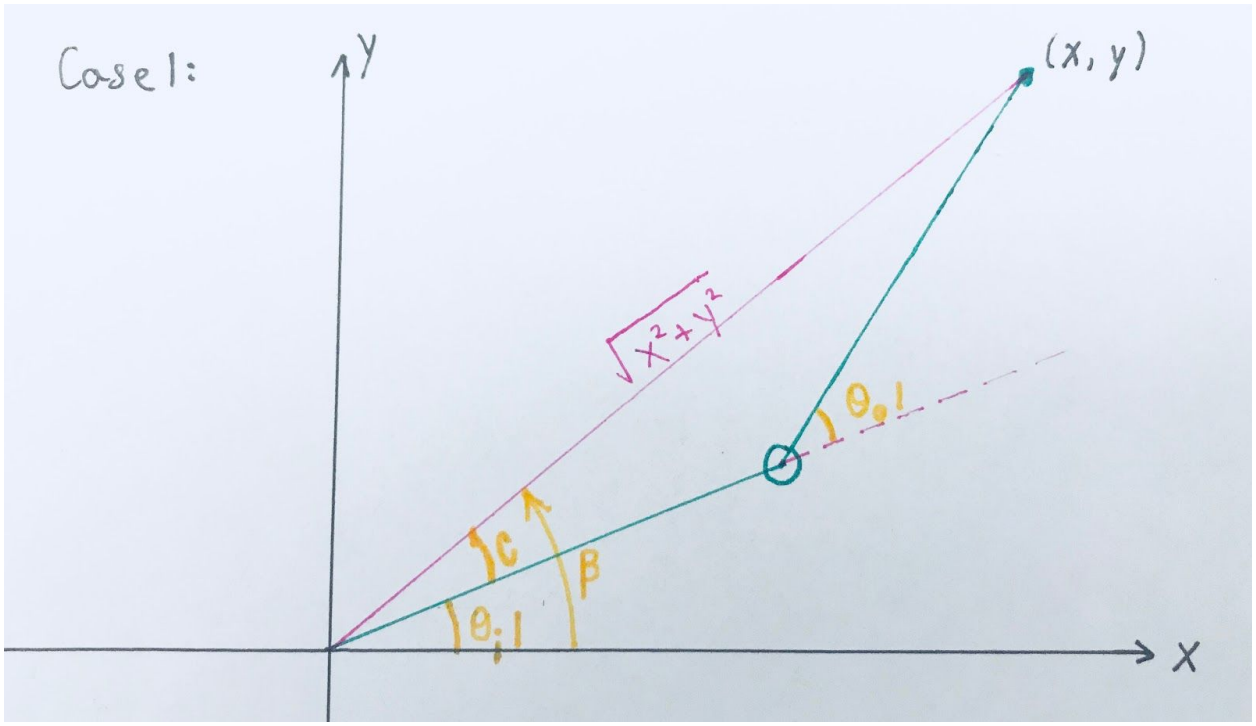


Figure 4.1

In figure 4.1, C is the angle opposite of the outer arm and β is the angle of the line from (0,0) to (x,y) with respect to the positive x-axis measured in counter-clockwise. C can be calculated using the cosine law, since all three sides of the triangle were known. β can be calculated using atan2.
C and β can be determined easily. As a result, θ_i1 can be calculated by subtracting β by C.

$$C = \cos^{-1}((l_i^2+x^2+y^2-l_o^2)/(2l_i\text{sqrt}(x^2+y^2)))$$
$$\beta = \text{atan}(y,x)$$

$$\boldsymbol{\theta_i1 = \beta - C}$$

Path planning (part E)

1. The path from the robot arm which was attached with a green marker to move within two points is as follows:

For straight lines, we made the robot arm to start from the point (18, 15) and made the endpoint as (18, -15).
The intermediate points between the two points were set to 350 for the above path.

2. The path that draws a straight line defined by a point, angle with respect to a horizontal axis and distance is similar to question 1.

3. The path from the robot arm which was attached with a green marker to move within two points is as follows:
For arcs, we made the center of the arc as point (5, 0) and the radius was set to 15, the arc length was set to 20
The intermediate points between the two points were set to 160 for the above path.

4.
(1) For the path of the straight line. According to mathematical calculation, the distance between two points is (15-(-15)) = 30cm and the path from the start point to the endpoint is parallel to the orientation of negative y-axis according to our coordinate.

(2) For the count of intermediate points. From the experiments, we found that if we set the count of intermediate points too low, i.e. 15 points for 30cm, the path will vibrate a lot (the vibration is just like magnify of the path in the picture near the start point and endpoint)

(3) The path in the picture near the start point and endpoint contains some vibrations. We add a magnify output module in our program to enlarge the output to both motors in case that the output is too low according to PID to prevent the motor sticks and stops for a long time when dealing with the conflict between fraction and output. Below is the expression of the magnify module in our program.

```
# adjust output when it is low, so robot can keep moving
if (output <= 22) and (output >= -22):
    output = output*2.5
```

(4) It is inevitable in the implementation because after the tune-up of the magnify output module we set the limitation to magnify the output as the value below 22 to the motor, and the magnifying time is 2.5 times.

Once the output drops below 22 it will trigger the magnify module and a burst to the motor which is 2.5 times of the original value is outputted to the motor. The sudden burst is the reason contribute to the oscillation on the plot. The oscillation is also due to not perfectly tuned the PID controller. The value we used for Kp, Ki and Kd were not perfect enough to keep the path away from the vibrations. Thus the vibrations are inescapability in our plot.