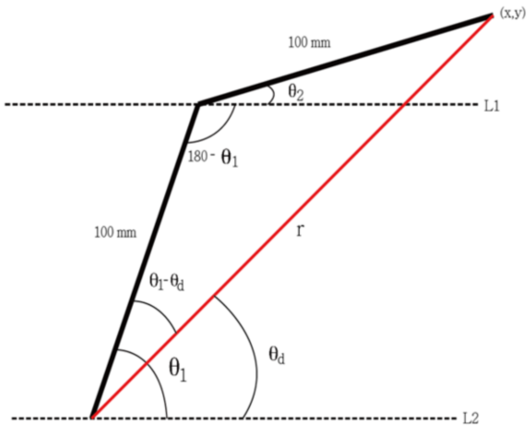
Inverse Kinematic Function




```
function y = InvKin(u)

coordinateX = u(1);
coordinateY = u(2);

r = sqrt(coordinateX^2+coordinateY^2);
theta_d = atand(coordinateY/coordinateX);

theta_one = acosd(r/200)+ theta_d;
theta_two = 2*theta_d-theta_one;

theta_one = theta_one*-3;
theta_two = theta_two*-3;


% Return Default Result
y = [theta_one;theta_two];
```

ELEC 341 PROJECT PART 2

Authors: Brandon Bwanakocha (35366525)
Kingstone Chen (25028549)

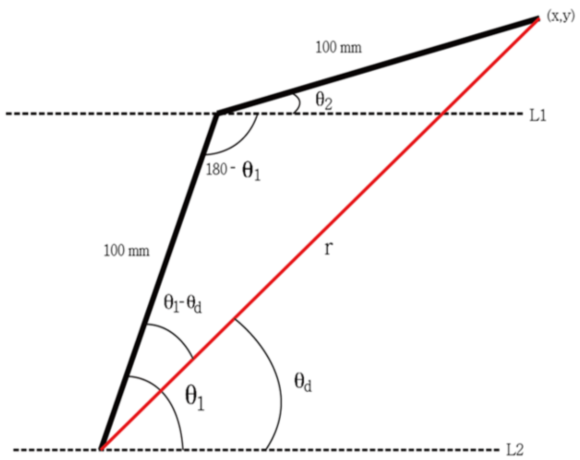
Objective: Determining kinematic functions for the parallel robot, and tuning the PID factors to achieve high accuracy control.

Kinematic Function Determination

- We manipulate the geometry of the arms to determine the outputs of the Inverse Kinematic Function. The outputs are $\theta_1$and θ_2 and the aim is to find them in terms of θ_d and r which we define below as:
- $\theta_d = \tan^{-1}\frac{y}{x}$
- r = $\sqrt{x^2 + y^2}$
- **Using the cosine rule, we get**
- $100^2 = 100^2 + y^2 - 2 \times 100r \times \cos(\theta_1 - \theta_d)$:
- And we solve for θ_1:

-
- $\cos(\theta_1 - \theta_d) = \frac{r^2}{2\times100\times r} = \frac{r}{200}$
- $\theta_1 = \theta_d + \cos^{-1}\frac{r}{200}$
- **Since $\boldsymbol{l_1\ and\ l_2}$ are parallel, it follows that:**
-
- $2\times(\theta_1 - \theta_d) + 180^o - \theta_1 + \theta_2 = 180^o$
-
- $\theta_1 - 2\,\theta_d + \theta_d = 0$
- $\theta_2 = 2\theta_d - \theta_1$

Direct Kinematics Function



Model.m | Inverse Kinematics | Direct Kinematics

```
function y = DirKin(u)

theta_one = u(1);
theta_two = u(2);

theta_one = theta_one/-3;
theta_two = theta_two/-3;

coordinateX = 100*cosd(theta_one) + 100*cosd(theta_two);
coordinateY = 100*sind(theta_one) + 100*sind(theta_two);


% Return Default Result
y = [coordinateX;coordinateY];
```

<u>Direct Kinematics Function Determination</u>

- The outputs of the Direct Kinematics Function are the coordinates (x,y) so we also manipulate the geometry of the arms to determine these:
- **Finding x**
- $x = 100\cos(\theta_1) + 100\cos(\theta_2)$
- **Finding y**

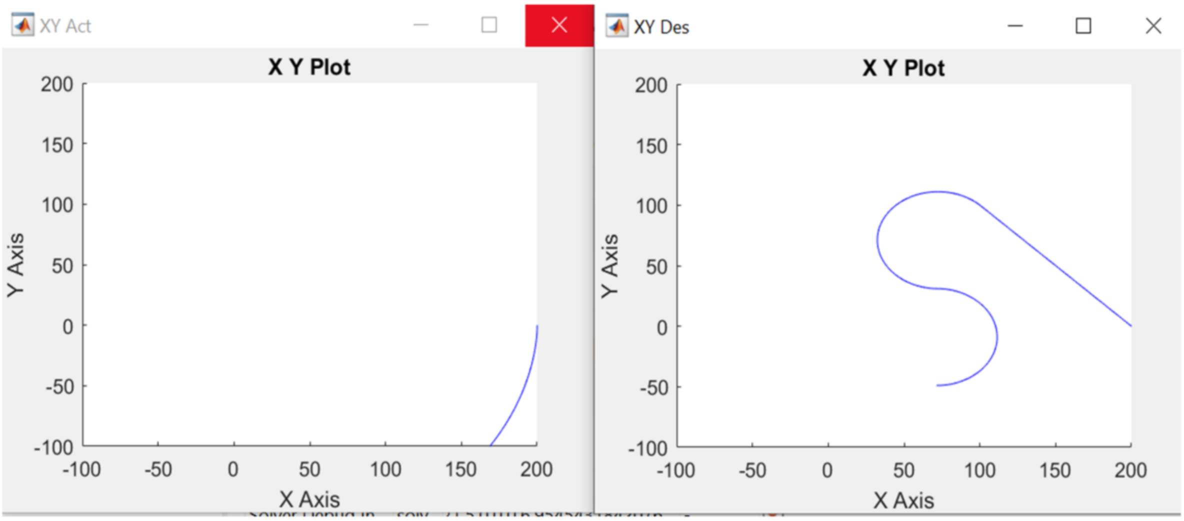
$y = 100\sin(\theta_1) + 100\sin(\theta_2)$

- **Note:** We need to put the gear ratio into consideration for our InvKin and DirKin functions, therefore, we multiply θ_1 and θ_2 by -3 in the InvKin and we divide θ_1 and θ_2 by -3 in the DirKin function

PID Tuning

Actual

Expected



PID Tuning involves changing the value of Kd, Kp and Ki to get PID gains that result in a stable system. We started with our original submitted values for Kp, Ki and Kd from part one which were:

- Kd = 2.93E-6
- Kp = 4.82E-6
- Ki = 8.93E-6
- These values gives a bad trace of the trajectory as shown
- The error was: 250.3
-

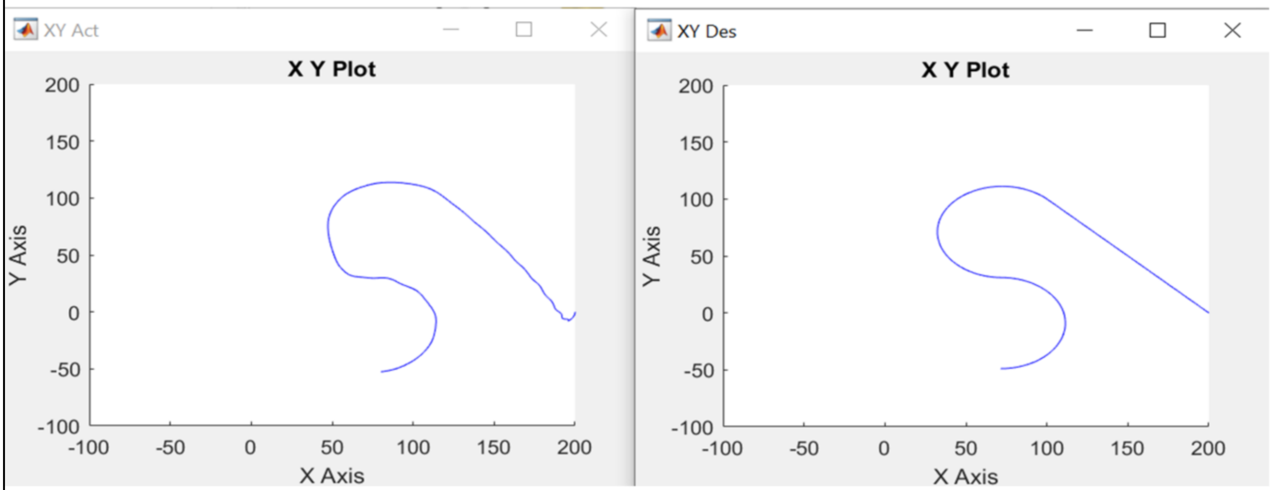
Tuning Procedure:

- We manually tuned the PID controller by following these steps:

- - Set Kd to about half of Ku so that our system has a decent steady state error and remains reasonably
- stable.
- - Increase Ki from zero to a point where our steady state error is minimal/ small.
- - Increase Kd to increase damping to the system.
- - Make small adjustments to these values to get the best graph that we could get.

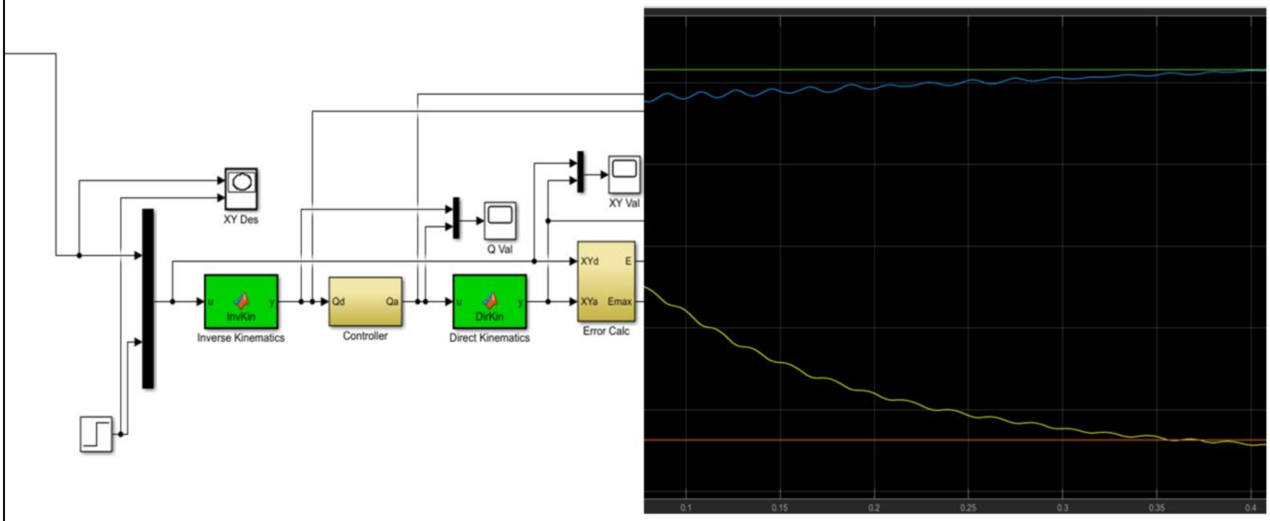
PID Tuning

Actual | Expected

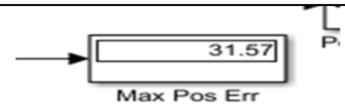


- Our final PID values were found to be:
- Ki = -0.035
- Kp = -0.003
- Kd = -0.0097
- And the error was reduced to 31.57
- The XY plot are really close to the desire path as shown which proves that our PID values are good for the system

Step Response

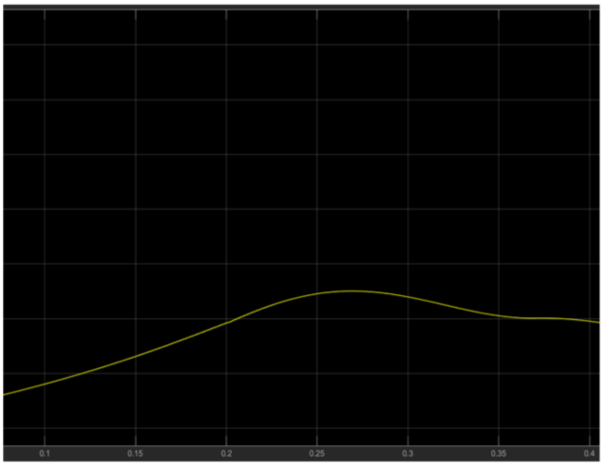


- We let X goes from 200 to 150 while Y goes from 0 to 50 to see how our arm response to the step input. As shown in the graph our system slowly adjust and reached the final value after some time. The blue line and green line are of interest to us. We see than the blue line approaches the green line and then settles on it showing that our system eventually reaches the desired path. This graph was our optimal after we tuned the value of Kd to reduce the time taken to settle on the green line.

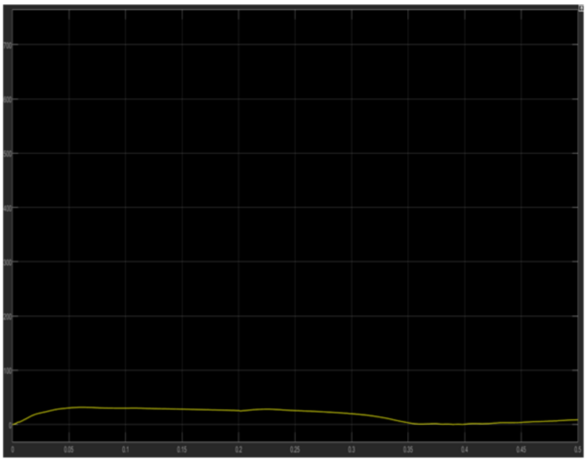


Error

Before tuning



After tuning



- We noticed that as we increase the value of Kp the error gets smaller, but at the same time the overshoot becomes higher and the graph looks less accurate (System is Destabilized). After considering Kp Ki and Kd and manually putting in different values, the smallest error was found to be 31.57 which shows that the arms responded accurately to the desire path.

- The above graph shows the error profile for out tuned PID and an untuned PID controller.

Conclusion

Throughout this project we learned:

- 1. How to use simulationX along with Matlab
- 2. Linearizing system
- 3. Finding transfer function
- 4. Finding zeros and poles
- 5. PID control and tuning to determine Kp, Ki, and Kd

We successfully developed a controller for the parallel arm robot by following the guidelines taught in the course. This experience will be extremely applicable for us as we proceed.