Robotics & Control of SCARA Robot

White Paper
Muhamad Izzul Syahmi - Student Number 45814472
Team Triumph

Nomenclature

SCARA -	Selective Compliance Articulated Robot Arm.
DOF -	Degrees of Freedom.
TF -	Transfer Function

1. SYSTEM OVERVIEW

The SCARA robot aims to move in a 2D plane to grab three cylindrical objects of negligible mass and release them into the bin. The system is designed to accept coordinates in order to perform the action. The system consists of:

- **Coordinates (X,Y, theta)** as input signals
- **Inverse Kinematic** module to convert the coordinates into desired motor angles
- **Controller** module to handle the system response and controller
- **Direct Kinematic** to convert the motor angles to the coordinates for error calculation

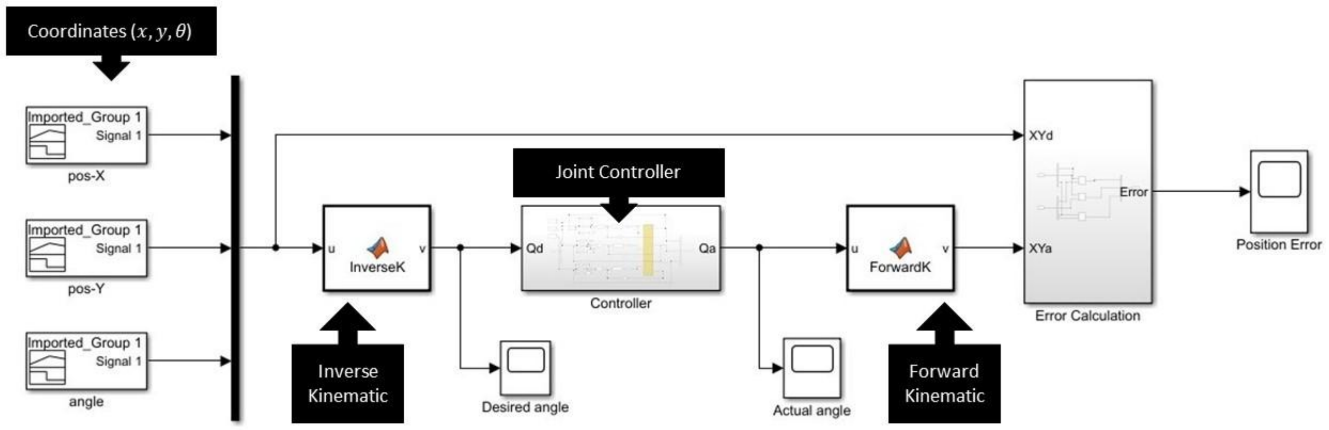


Figure 1: System Overview

2. ROBOT KINEMATICS

2.1 Robot Arm Structure

The robotic arm is a combination of links and joints that move coordinately. It is made up of 3 links – shoulder, elbow and wrist - and 3 joints attached to the links. The joints represent the DC motors which exert torques to allow the movement of the links. In addition to that, the robotic arm also consists of a gripper which has a fixed position at the wrist. Hence, it has **3.5 DOF**. The overall structure of the robot arm in 2D plane is shown in Figure 2 below.

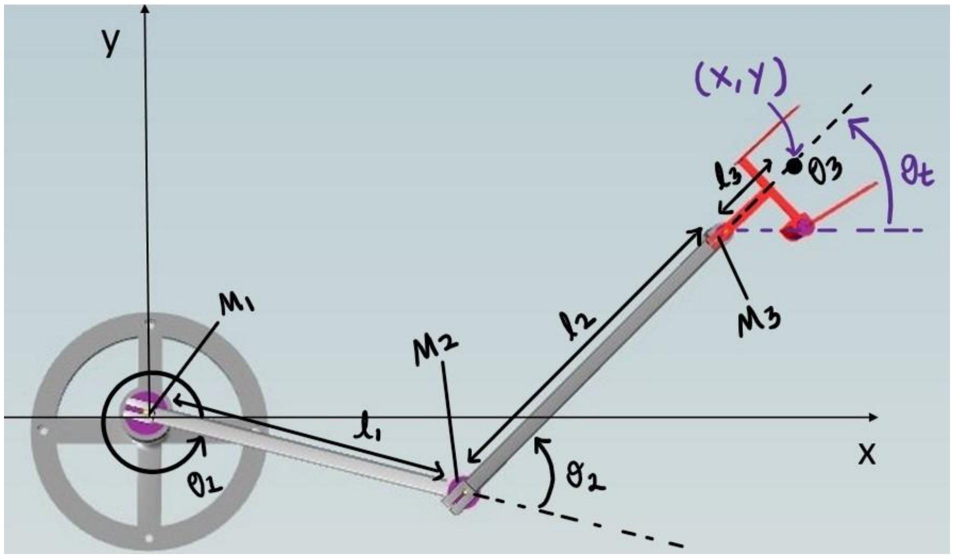


Figure 2: Robotic Arm Structure in 2D plane

2.2 Forward Kinematics

To calculate the kinematics, we made an assumption that the gripper is fixed without any movement. Thus, the kinematics are based on a 3-DOF articulated manipulator. From Figure 2 above, the forward kinematics are as shown:

Inputs: θ_1 , θ_2 , θ_3 Outputs: $x,\ y,\ \theta_t$

$$x = l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) + l_3 cos(\theta_1 + \theta_2 + \theta_3)$$

$$y = l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) + l_3 sin(\theta_1 + \theta_2 + \theta_3)$$

$$\theta_t = \theta_1 + \theta_2 + \theta_3$$

Matlab Implementation:

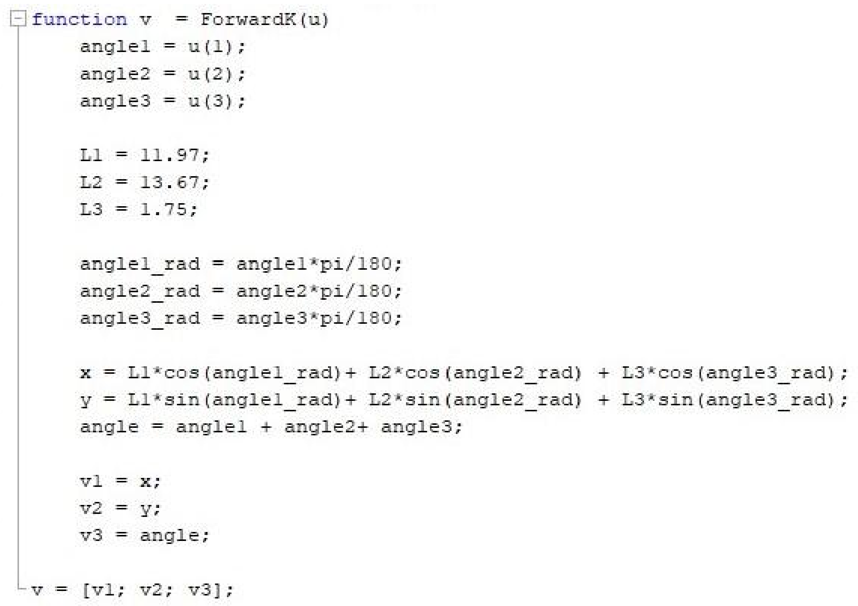

```
function v  = ForwardK(u)
    angle1 = u(1);
    angle2 = u(2);
    angle3 = u(3);

    L1 = 11.97;
    L2 = 13.67;
    L3 = 1.75;

    angle1_rad = angle1*pi/180;
    angle2_rad = angle2*pi/180;
    angle3_rad = angle3*pi/180;

    x = L1*cos(angle1_rad)+ L2*cos(angle2_rad) + L3*cos(angle3_rad);
    y = L1*sin(angle1_rad)+ L2*sin(angle2_rad) + L3*sin(angle3_rad);
    angle = angle1 + angle2+ angle3;

    v1 = x;
    v2 = y;
    v3 = angle;

v = [v1; v2; v3];
```

Figure 3:Matlab Implementation of Forward Kinematics

2.2 Inverse Kinematics

Inverse kinematics of this robot arm are as shown:

Inputs: $x,\ y,\ \theta_t$ Outputs: θ_1 , θ_2 , θ_3

$$x' = x - l_3 cos(\theta_t)$$

$$y' = y - l_3 sin(\theta_t)$$

$$r = \sqrt{(x')^2 + (y')^2}$$

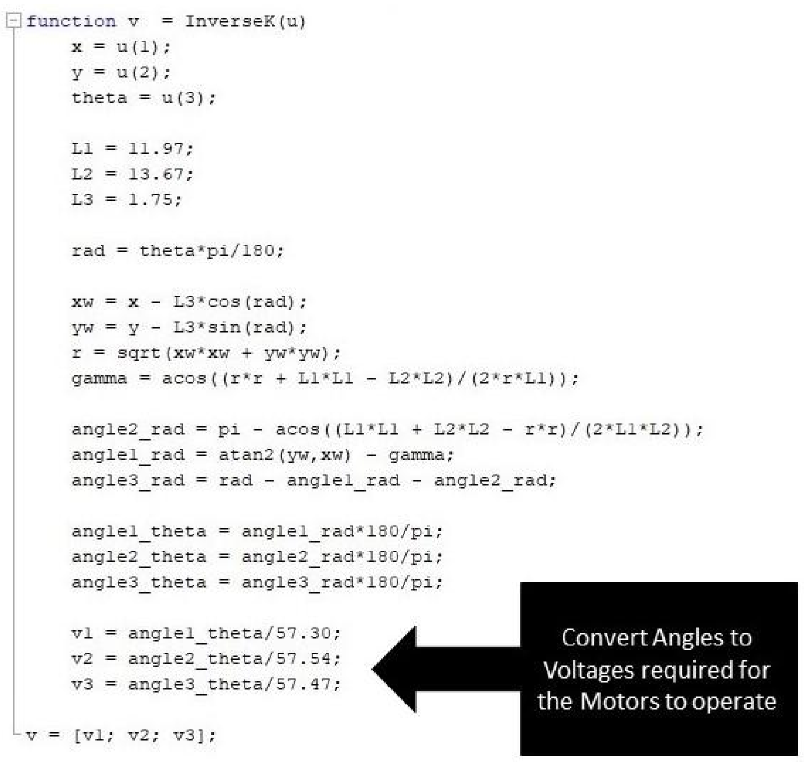
$$\lambda = arccos\left(\frac{r^2 + l_1^{\ 2} - l_2^{\ 2}}{2 l_1 l_2}\right)$$

$$\theta_1 = arctan(y', x') - \lambda$$

$$\theta_2 = \pi - arccos\left(\frac{l_1^{\ 2} + l_2^{\ 2} - r^2}{2 l_1 l_2}\right)$$

$$\theta_3 = \theta_t - \theta_1 - \theta_2$$

Matlab Implementation:



```
function v  = InverseK(u)
    x = u(1);
    y = u(2);
    theta = u(3);

    L1 = 11.97;
    L2 = 13.67;
    L3 = 1.75;

    rad = theta*pi/180;

    xw = x - L3*cos(rad);
    yw = y - L3*sin(rad);
    r = sqrt(xw*xw + yw*yw);
    gamma = acos((r*r + L1*L1 - L2*L2)/(2*r*L1));

    angle2_rad = pi - acos((L1*L1 + L2*L2 - r*r)/(2*L1*L2));
    angle1_rad = atan2(yw,xw) - gamma;
    angle3_rad = rad - angle1_rad - angle2_rad;

    angle1_theta = angle1_rad*180/pi;
    angle2_theta = angle2_rad*180/pi;
    angle3_theta = angle3_rad*180/pi;

    v1 = angle1_theta/57.30;
    v2 = angle2_theta/57.54;
    v3 = angle3_theta/57.47;

v = [v1; v2; v3];
```

Convert Angles to Voltages required for the Motors to operate

Figure 4: Matlab Implementation of Inverse Kinematics

3. PATH PLANNING

The robotic arm is operated based on the path planning which acts as an input to the system in the form of coordinates x,y and angle. Path planning requires some considerations.

1) Avoid obstacles:
   Obstacles of this project are the cylindrical objects. From the requirement of this project, the robot arm should not tip or push the objects.

2) Coordinates limitation:
   There is a range of coordinates where the robot arm could not reach given the length of the links L1,L2 and L3. Designing a path planning that includes these coordinates will result in inaccuracy. Based on the simulation, the range of the coordinates are x:(-5:5), y:(-5:5) that must be avoided.

3) Reduce overshoot to maintain accuracy:
   Since the robot arm consists of connected links and joints, small overshoot of the motors can cause substantial inaccuracy. This issue is solved by designing a path planning algorithm that allows the motor to reach steady-state error before proceeding with the next coordinate.

4) Use nominal voltage to operate:
   Each motor has its own nominal voltage where the motor must be operated at. Designing a path planning algorithm that requires higher voltage than the nominal is not ideal as it can cause inconsistency. This constraint is solved by increasing the time for the robot arm to move from one coordinate to another. For instance, for the motor to move from x = 10 to x = 20 in 1 seconds would require a much higher torque compared to in 5 seconds of duration.

3.1 Path planning design:

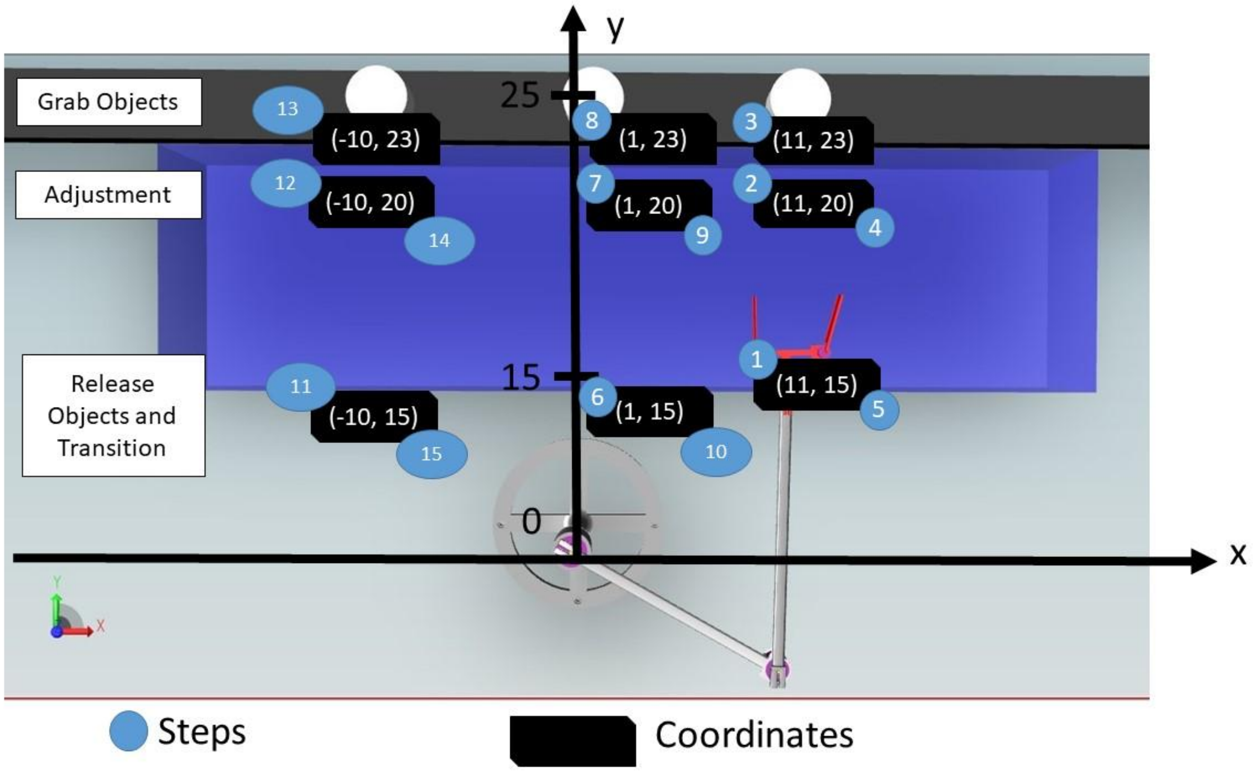


Figure 5: Path planning design

From figure 5, there are 3 stages - Transition, Adjustment, Grab Object - that the robot arm requires to follow. These stages are designed to ensure consistency and accuracy when the robot arm operates.

Stages:

1) **Transition** - Position the gripper on the x-coordinate of the cylindrical objects, and 10cm below the y-coordinate. These coordinates are also where the robot arm will release the objects into the bin

2) **Adjustment** - This stage is required to ensure the gripper is accurately aligned with the objects before the process of grabbing the objects as it allows the robot arm to reach steady-state error prior to the next stage.

3) **Grab object** - The motor gripper closes to grab the object.

3.2 Path planning signals (x,y)

Below are the path planning coordinates signals that act as inputs for the system. The angle is kept constant at 90 degree angle.These signals are then converted to desired angles for each motor joint through Inverse Kinematics algorithm.

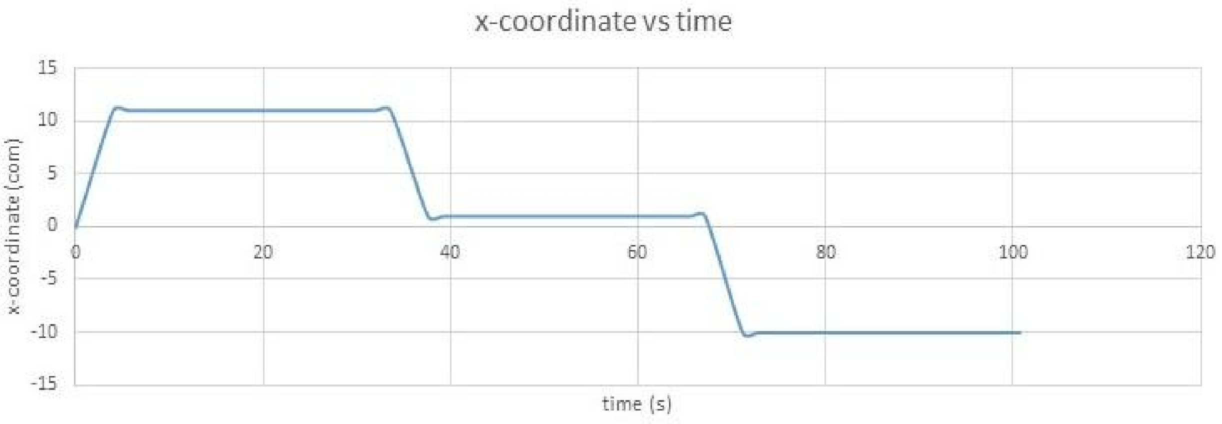


Figure 6: x-coordinate of path planning

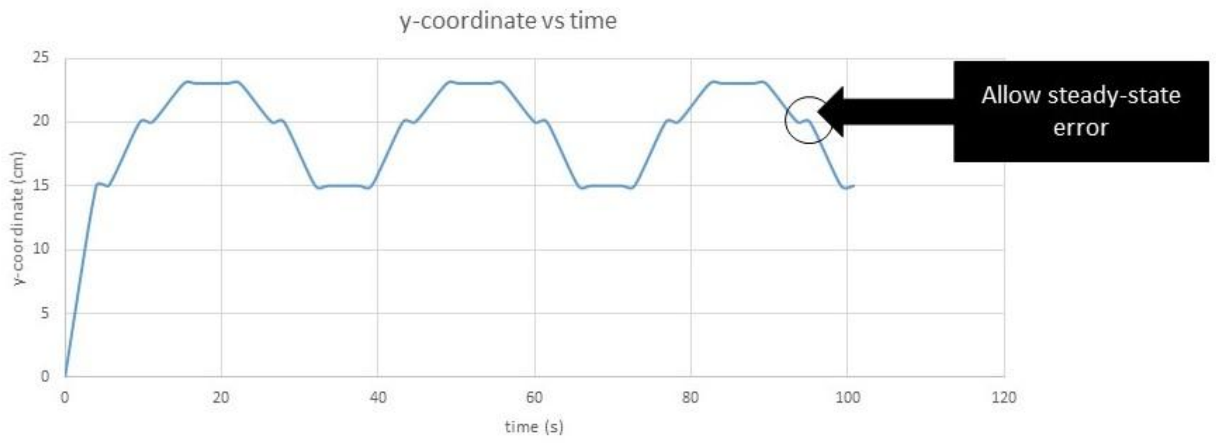


Figure 7: y-coordinate of path planning

4. CONTROLLER

4.1 Controller Overview:

Model system and actual system as shown in Figure 8 and 9 are designed to analyse and simulate the robot arm. The model system consists of: **Zero-order hold**, **Filter**, **PID**, **Amplifier Model**, **Electrical Motor Model** and **Mechanical Motor Model**.

<u>Model System:</u>

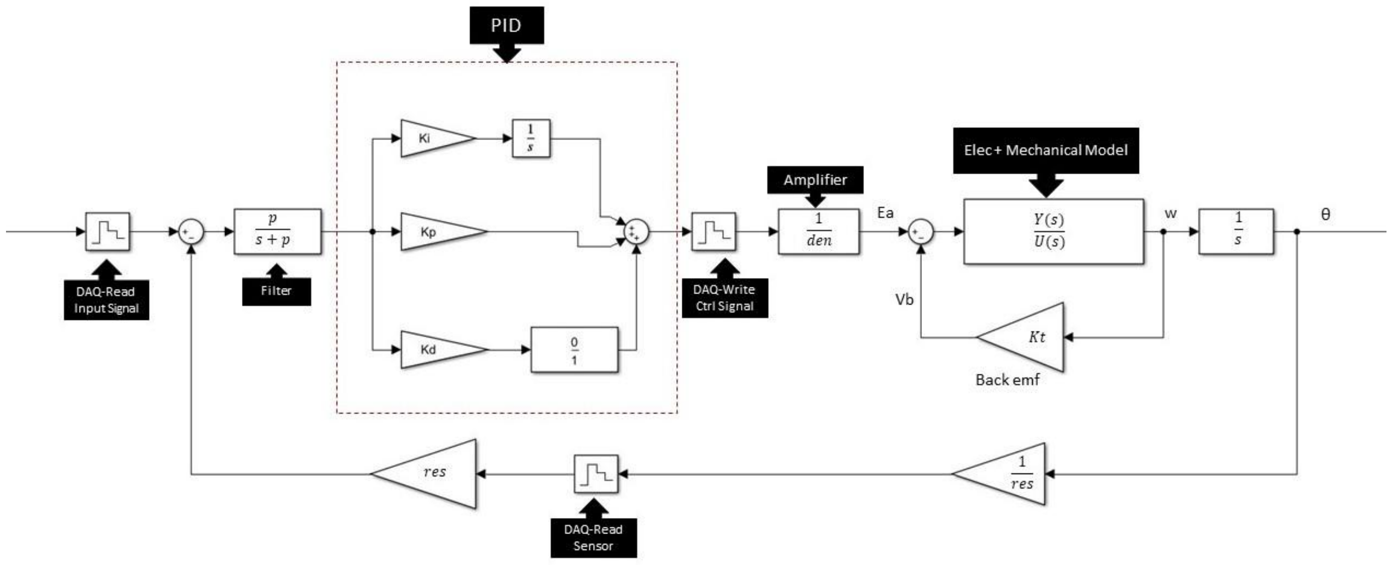


Figure 8: Model system for one motor

Actual System for SimulationX:

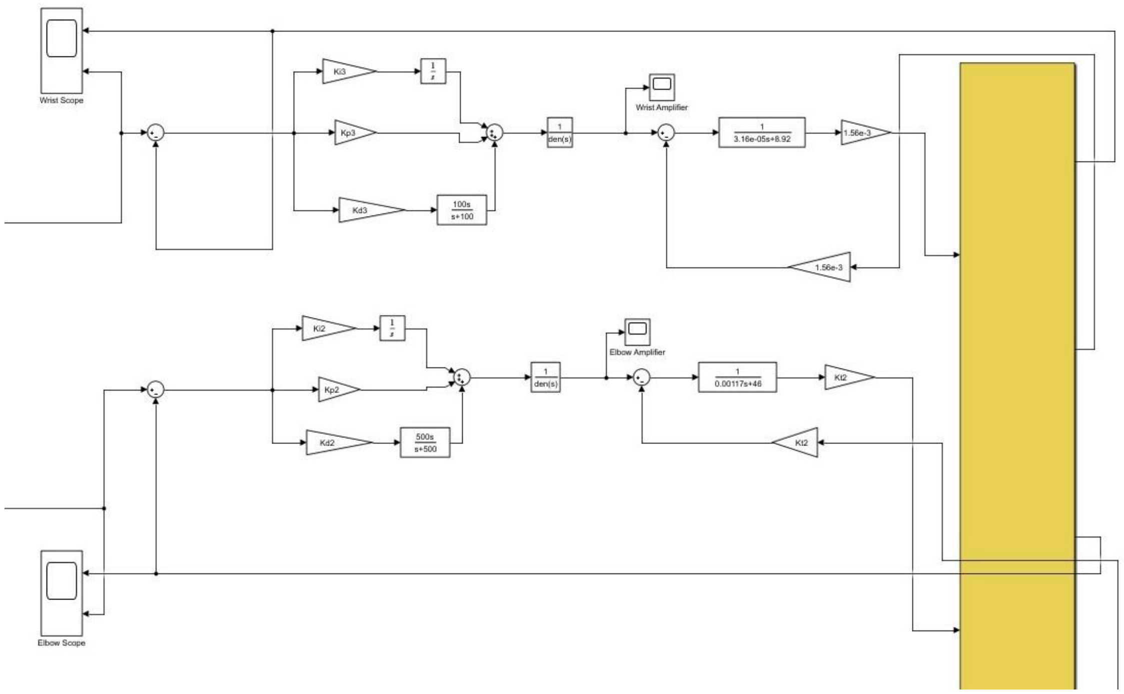


Figure 9: Actual system for SimulationX

4.2 PID Controller:

4.2.1 PID Controller

PID control is based on 3 aspects, Proportional (Kp), Integral (Ki), and Derivative (Kd). The values of the PID controller parameters are presented below. Further analysis on how these parameters are obtained is in part **5: Tuning Strategy**

$$\frac{U(s)}{E(s)} = Kp + \frac{Ki}{s} + Kd\frac{as}{s+a}$$

Parameters	Shoulder Motor	Elbow Motor	Wrist Motor	Gripper Motor
Kp	10.365	0.2004	0.03351	0.14
Ki	0.24132	0.000234	0.01046	0
Kd	247.002	123.4744	0.97041	0
a	400	200	100	0

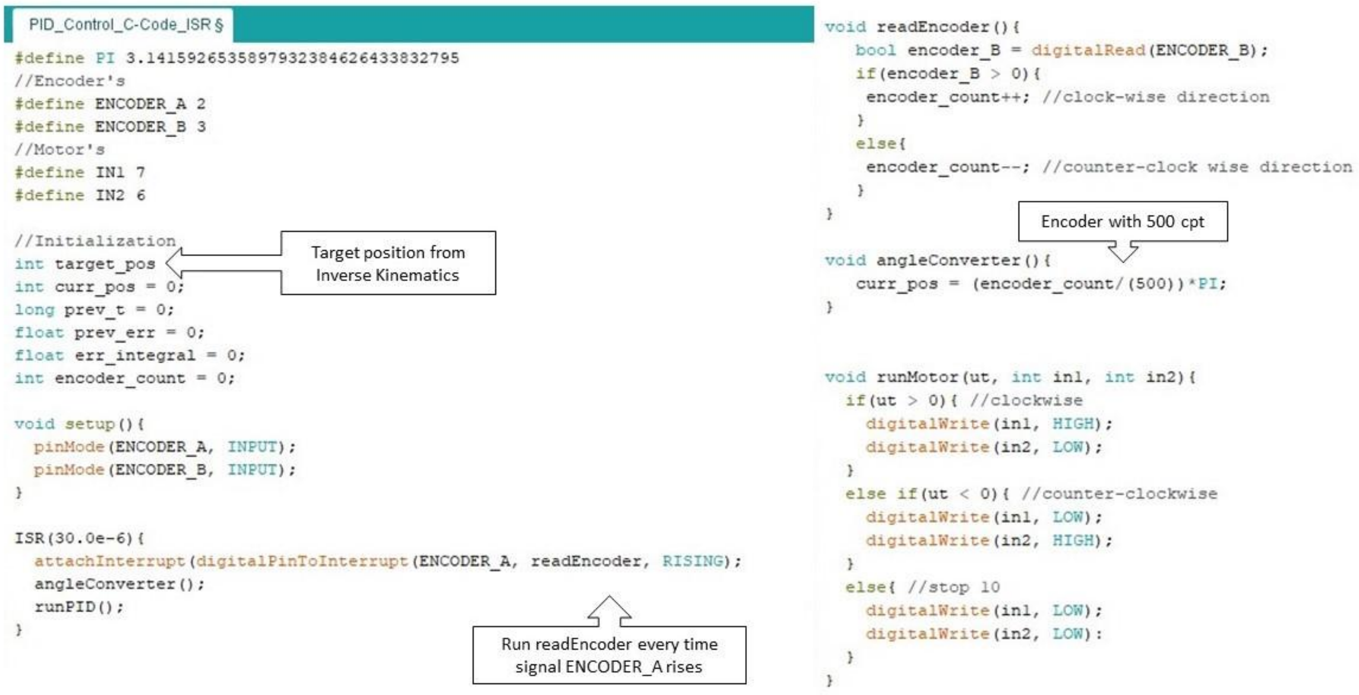
Figure 10: PID Controller parameters

4.2.2 C-Code of PID Control

In this project, we chose ATmega32U4 as the microcontroller. It has a clock speed of 16MHz. Below is the estimated clock-rate for each functions :

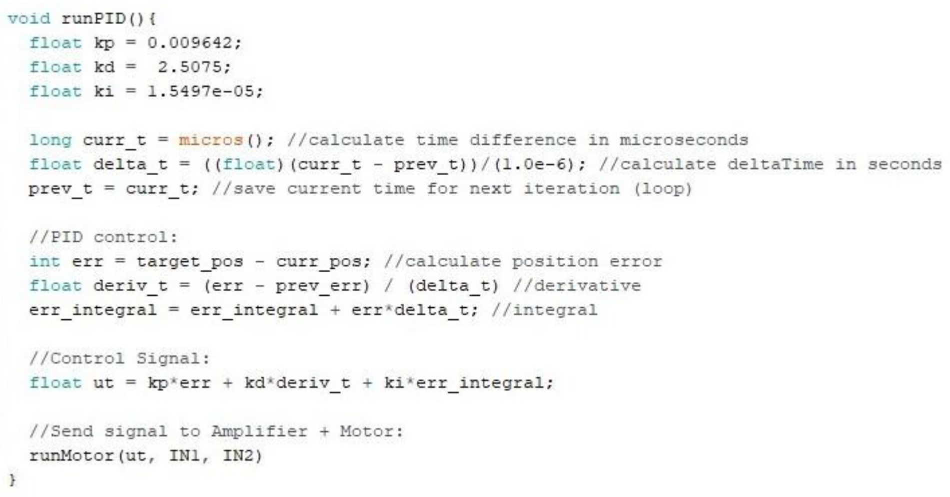
- attachInterrupt = 6 clock cycles
- readEncoder() = 47 clock cycles
- angleConverter() = 38 clock cycles
- runMotor(ut, in1, in2) = 49 clock cycles
- runPID() = 235 clock cycles (including runMotor function)

With the estimations above, we can approximate the ISR clock-rate: (6 + 47 + 38 + 235)/16MHz = 20.375 microseconds. Hence, to ensure the ISR avoids interrupting itself, we have set the interrupt time = 30 microseconds.


PID_Control_C-Code_ISR §
Target position from Inverse Kinematics
Run readEncoder every time signal ENCODER_A rises
Encoder with 500 cpt

```
#define PI 3.14159265358979323846264338327950
//Encoder's
#define ENCODER_A 2
#define ENCODER_B 3
//Motor's
#define IN1 7
#define IN2 6

//Initialization
int target_pos
int curr_pos = 0;
long prev_t = 0;
float prev_err = 0;
float err_integral = 0;
int encoder_count = 0;

void setup(){
  pinMode(ENCODER_A, INPUT);
  pinMode(ENCODER_B, INPUT);
}

ISR(30.0e-6){
  attachInterrupt(digitalPinToInterrupt(ENCODER_A, readEncoder, RISING);
  angleConverter();
  runPID();
}
```

```
void readEncoder(){
   bool encoder_B = digitalRead(ENCODER_B);
   if(encoder_B > 0){
    encoder_count++; //clock-wise direction
   }
   else{
    encoder_count--; //counter-clock wise direction
   }
}

void angleConverter(){
   curr_pos = (encoder_count/(500))*PI;
}


void runMotor(ut, int in1, int in2){
  if(ut > 0){ //clockwise
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if(ut < 0){ //counter-clockwise
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else{ //stop 10
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW):
  }
}
```



```
void runPID(){
  float kp = 0.009642;
  float kd =  2.5075;
  float ki = 1.5497e-05;

  long curr_t = micros(); //calculate time difference in microseconds
  float delta_t = ((float)(curr_t - prev_t))/(1.0e-6); //calculate deltaTime in seconds
  prev_t = curr_t; //save current time for next iteration (loop)

  //PID control:
  int err = target_pos - curr_pos; //calculate position error
  float deriv_t = (err - prev_err) / (delta_t) //derivative
  err_integral = err_integral + err*delta_t; //integral

  //Control Signal:
  float ut = kp*err + kd*deriv_t + ki*err_integral;

  //Send signal to Amplifier + Motor:
  runMotor(ut, IN1, IN2)
}
```

Figure 11: C-Code for PID Control of a single motor

5. TUNING STRATEGY

5.1 Wrist Motor PID Control

The transfer function (TF) of the Wrist Motor is shown below with its root locus presented in Figure 12.

$$TF_{wrist} = \frac{0.00156}{(1.053e{-}11)s^4+(2.974e{-}6)s^3+(0.0001488)s^2+(4.855e{-}6)s}$$

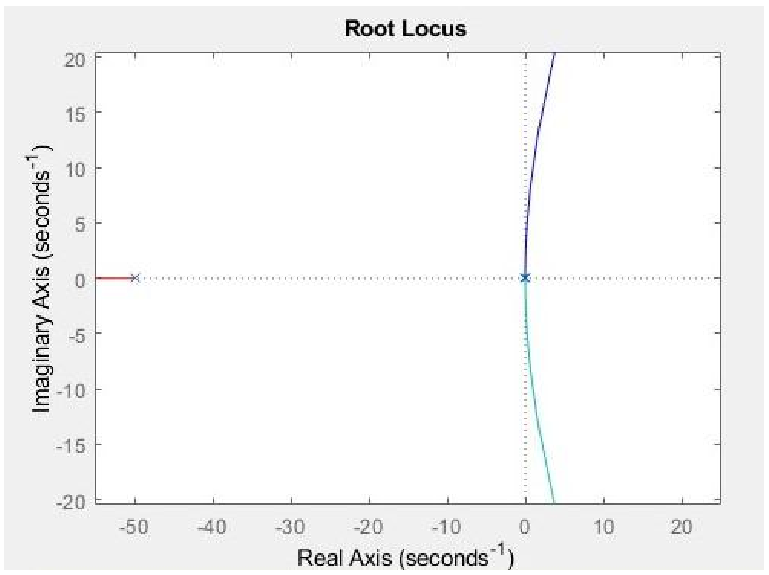


Figure 11: Root Locus of Wrist Motor TF

Since there are two poles located near the imaginary axis, the margin is too small before the system goes unstable. Hence, a controller is designed to increase the margin by adding poles and zeros at specific locations. Below are ideal poles, zeros and gain for the PID controller:

poles = -100, 0   complex zeros = -0.017 ∓ 0.103j,   K = 97.075

$$C(s) = \frac{97.075(s^2+0.0346s + 0.1076)}{s(s+100)}$$

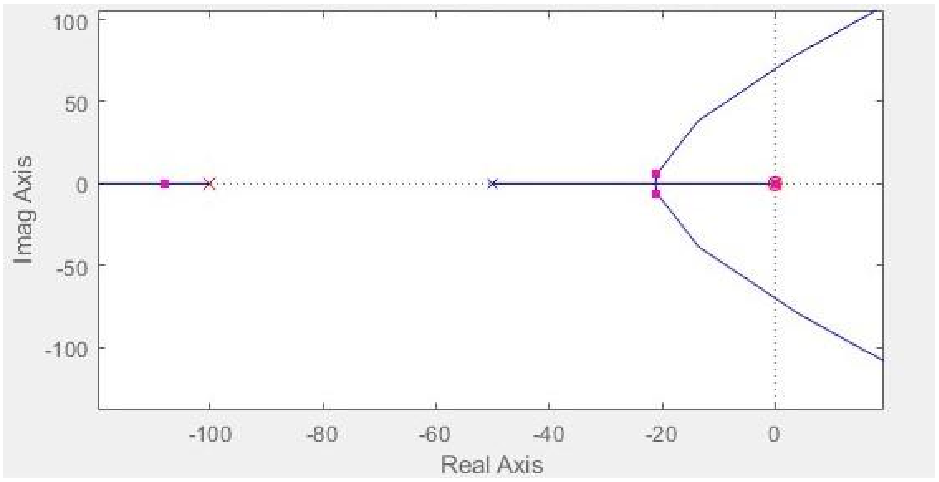


Figure 12: Root Locus of Wrist Motor TF

To gather the Kp, Kd, Ki from C(s), we use the calculation below:

$$C(s) = \frac{\alpha s^2+\beta s + \gamma}{s(s+\delta)}$$

$$a = \delta$$

$$K_p = \frac{(-\gamma+\beta\delta)}{\delta^2}$$

$$K_i = \frac{\gamma}{\delta}$$

$$K_d = \frac{(\gamma-\beta\delta+\alpha\delta^2)}{\delta^3}$$

Thus, based on the calculation, the PID controller gain for Wrist system are:

Kp = 0.033513, Kd = 0.97041, Ki = 0.01046, a = 100

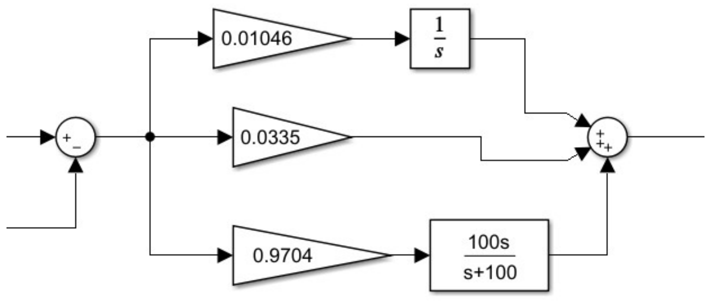


Figure 13: PID Gain for Wrist System

The step response of the system with the controller is shown in figure 14:

- Rise time = 0.149s
- Settling time = 0.26s
- Overshoot = 0.0025%

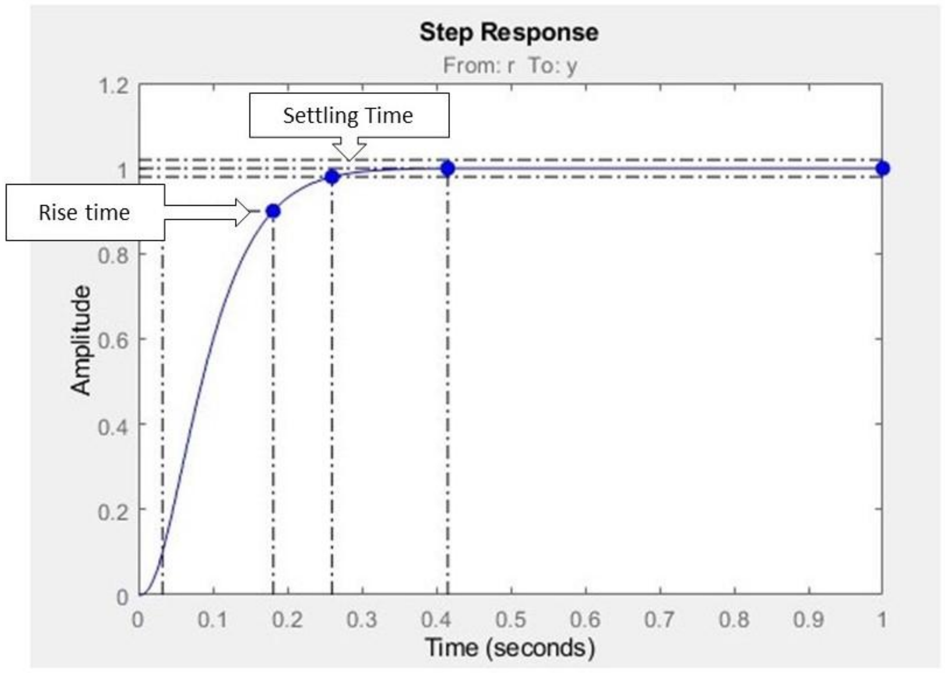


Figure 14: Step response of Wrist Motor with Controller

5.2 Elbow Motor PID Control

The TF of the Elbow Motor is shown below with its root locus presented in Figure 14.

$$TF_{elbow} = \frac{0.0123}{(1.099e-7)s^4+(0.004325)s^3+(0.216)s^2+(0.0003032)s}$$

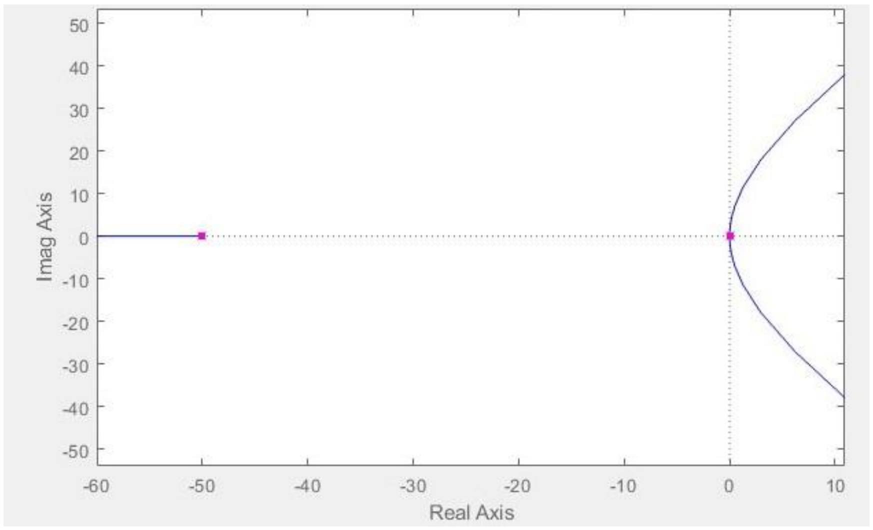


Figure 15: Root Locus of Elbow Motor TF

PID Control:

poles = -200, 0   complex zeros = -0.0008114 ∓ 0.00111j,   K = 24695

$$C(s) = \frac{24695(s^2+0.001623s + 1.894e{-}6)}{s(s+200)}$$

Kp =0.200398, Kd = 123.4744, Ki = 0.000234, a = 200

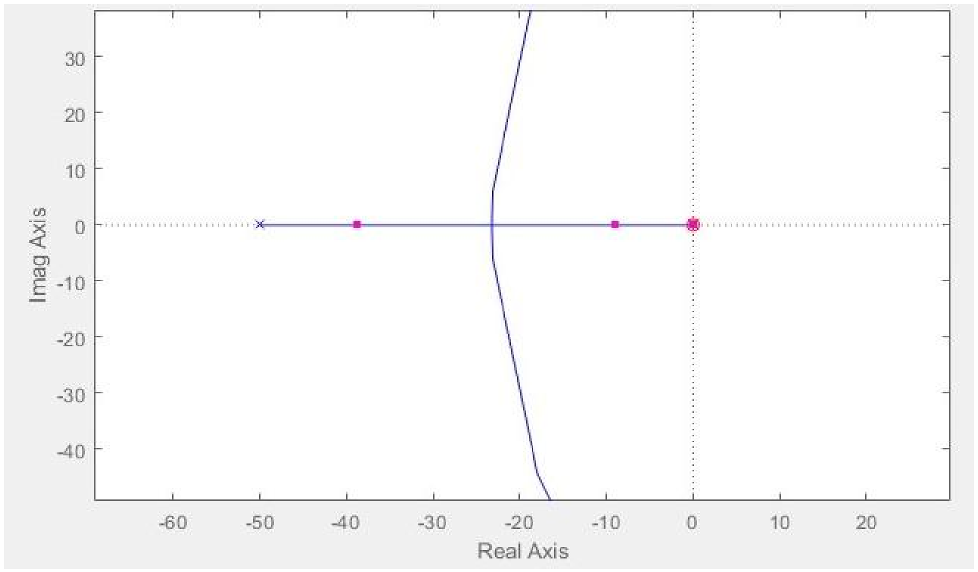


Figure 16: Root Locus of Elbow Motor TF with PID control

The step response of Elbow Motor with PID Control:

- Rise time = 0.256s
- Settling time = 0.47s
- Overshoot = 0%

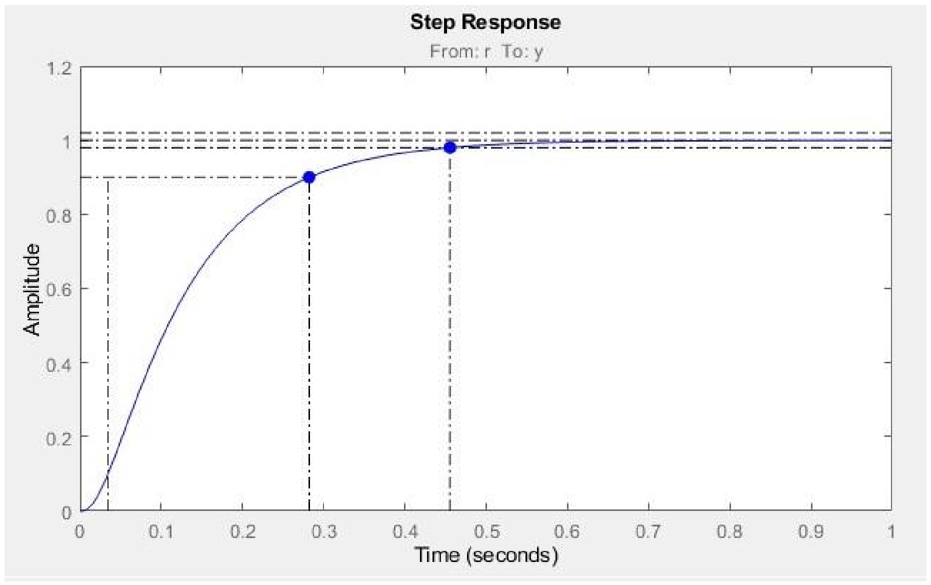


Figure 17: Step response of Elbow Motor with Controller

5.3 Shoulder Motor PID Control

The TF of the Shoulder Motor is shown below with its root locus:

$$TF_{shoulder} = \frac{0.036}{(6.053e-7)s^4+(0.02171)s^3+(1.084)s^2+(0.002589)s}$$

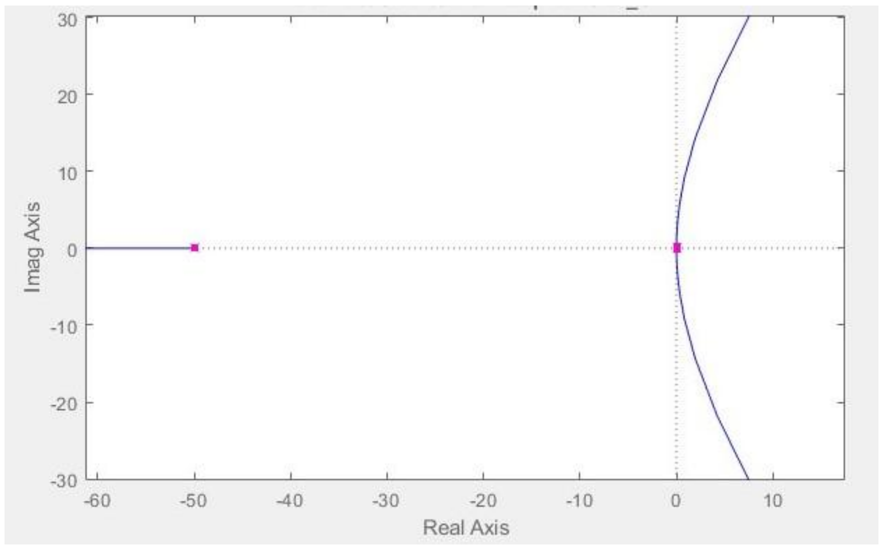


Figure 18: Root Locus of Shoulder Motor TF

PID Control:

poles = -400, 0 complex zeros = -0.021 ∓ 0.0232j, K = 98811

$$C(s) = \frac{98811(s^2+0.04196s + 0.0009769)}{s(s+400)}$$

Kp =10.365, Kd = 247.002, Ki = 0.24132, a = 400

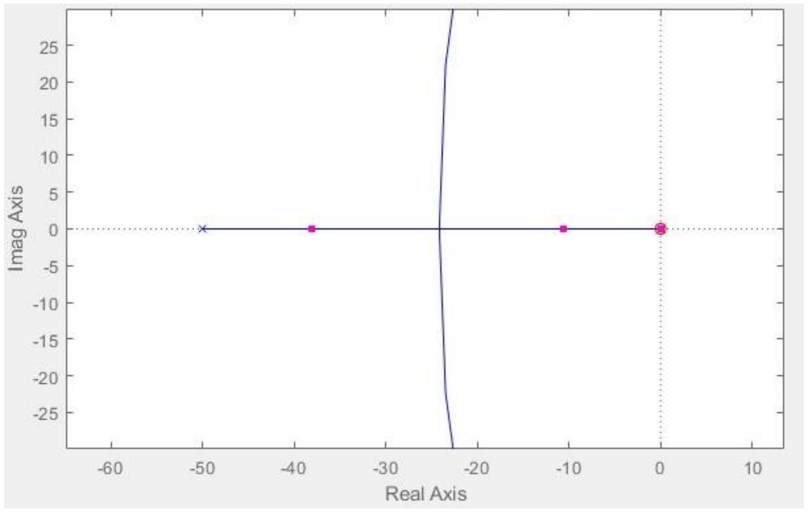


Figure 19: Root Locus of Shoulder Motor TF with PID Control

The step response of Elbow Motor with PID Control:

- Rise time = 0.215s
- Settling time = 0.38s
- Overshoot = 0.303%

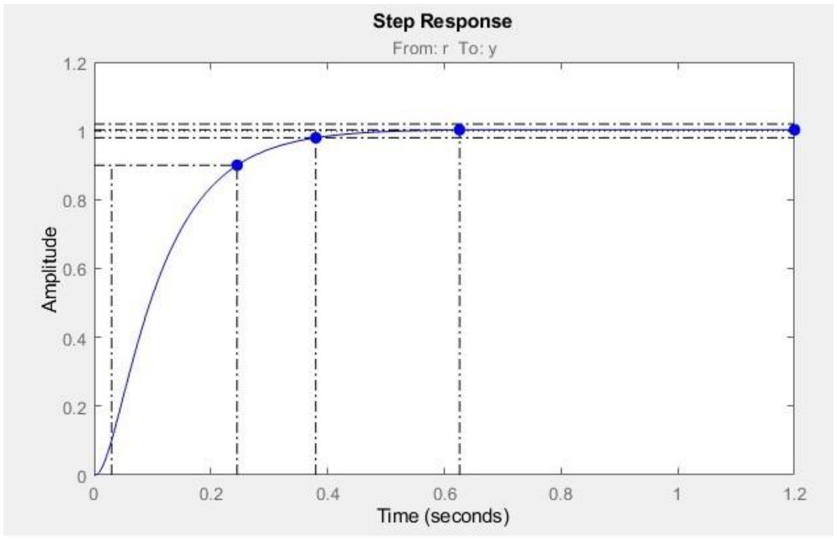


Figure 20: Step response of Shoulder Motor with Controller

5.3 Robot Arm Performance:
Based on the path planning in part 3, the cycle time for the robot arm to move the objects for 10cm and discard them is around **140 seconds**. The path planning and lower % of overshoot from the motors avoid the robot arm to tip or push the objects. However, due to the weight of the links, the shoulder and elbow motors still require higher voltage than nominal in order to create enough torque to operate. In conclusion, the robot arm satisfies every requirement but fails one constraint which is to operate at or below nominal voltage.

<u>Performance Result:</u>
Below are the desired angles (yellow line) and actual angles (blue line) signals from the simulation. As we can see, due to the path planning and the PID control, the motors are able to follow exactly the signals of the desired angles with minimal error margin. As for the gripper, since it only relies on P-Control, the error is the most significant.

Wrist Motor:

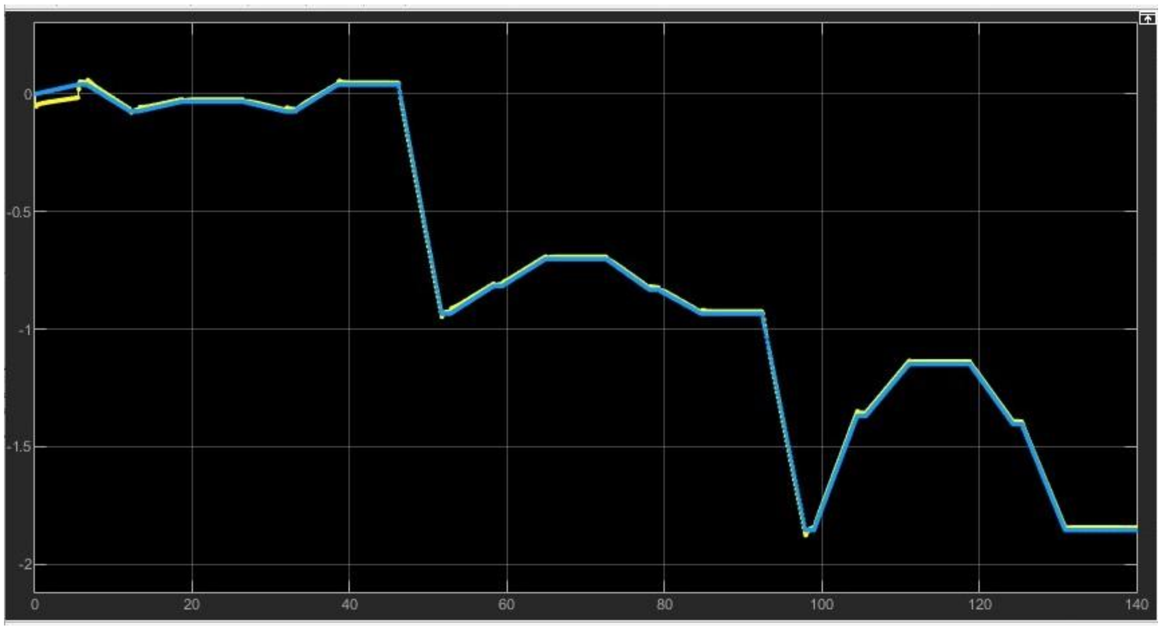


Figure 21: Desired angle (yellow) and Actual angle (angle) of Wrist Motor

Elbow Motor:

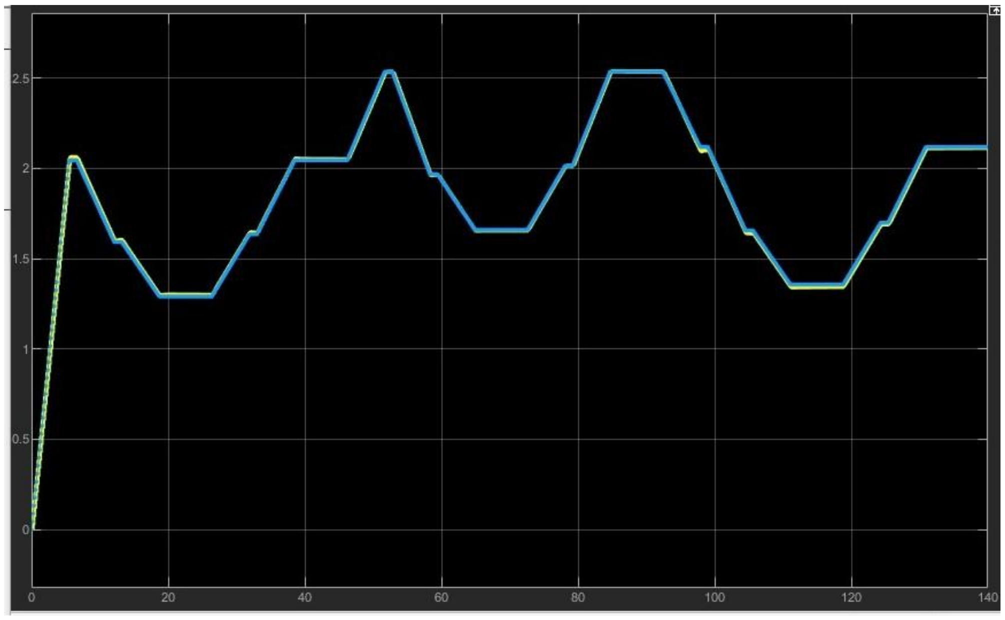


Figure 22: Desired angle (yellow) and Actual angle (angle) of Elbow Motor

Shoulder Motor:

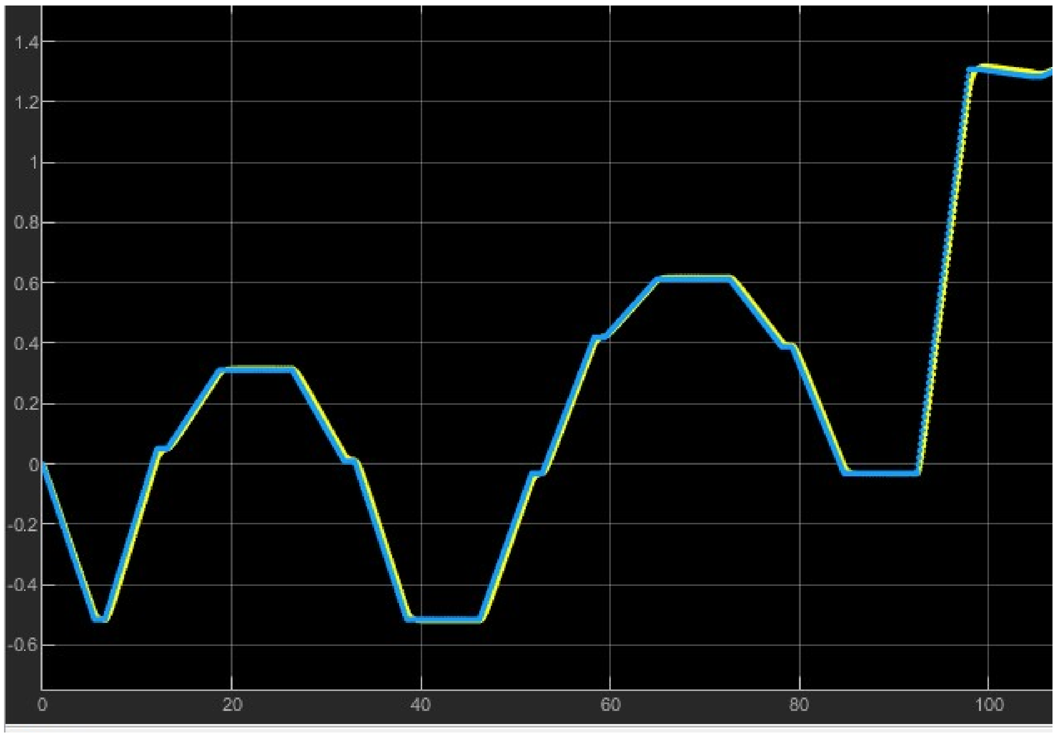


Figure 23: Desired angle (yellow) and Actual angle (angle) of Shoulder Motor

Gripper Motor:

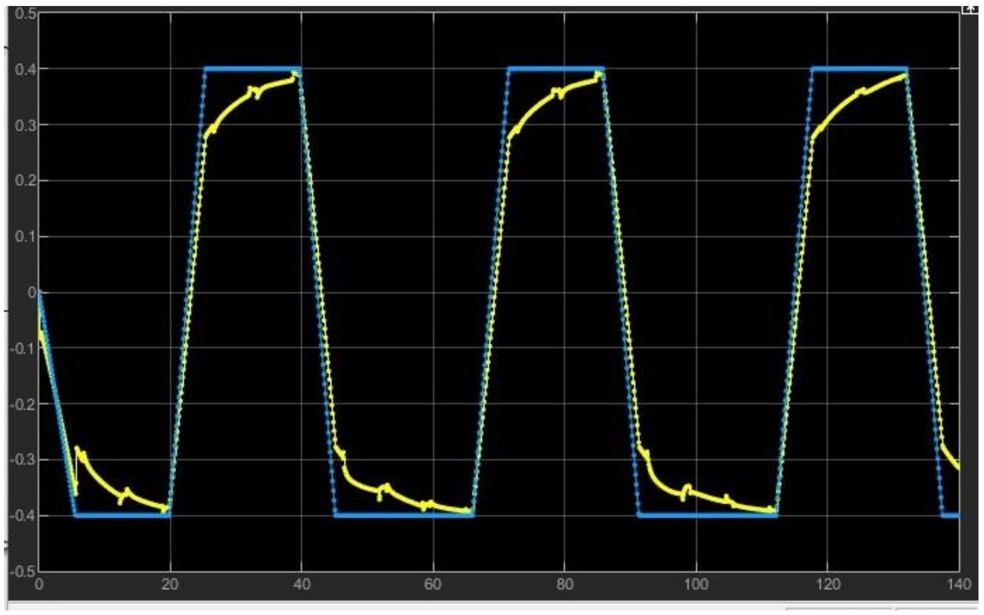


Figure 24: Desired angle (yellow) and Actual angle (angle) of Gripper Motor

<u>Input voltage from Amplifiers:</u>

Shoulder motor should operate around 24V however our robot arm requires a maximum 32V. For the elbow, the nominal voltage is 12V however it operates a maximum 20V. Wrist Motor operates below the nominal voltage of 3V.

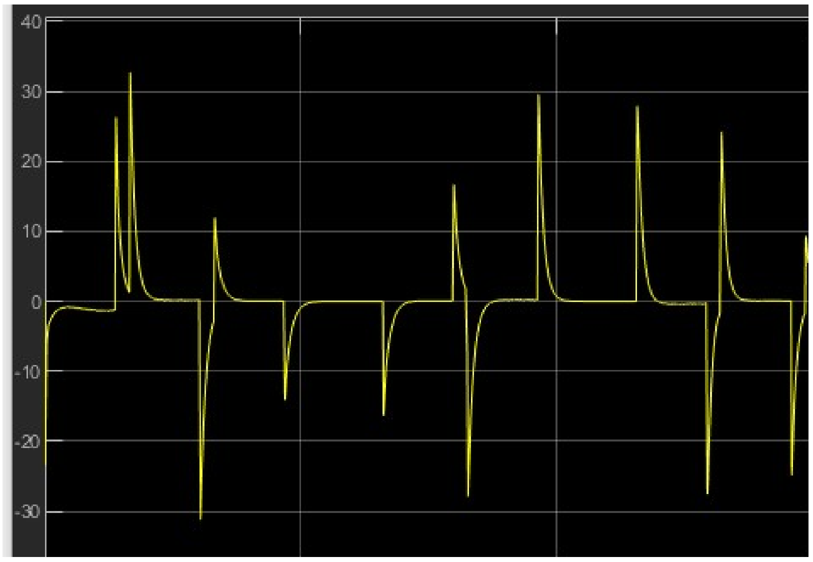


Figure 25: Input signal of Shoulder Motor

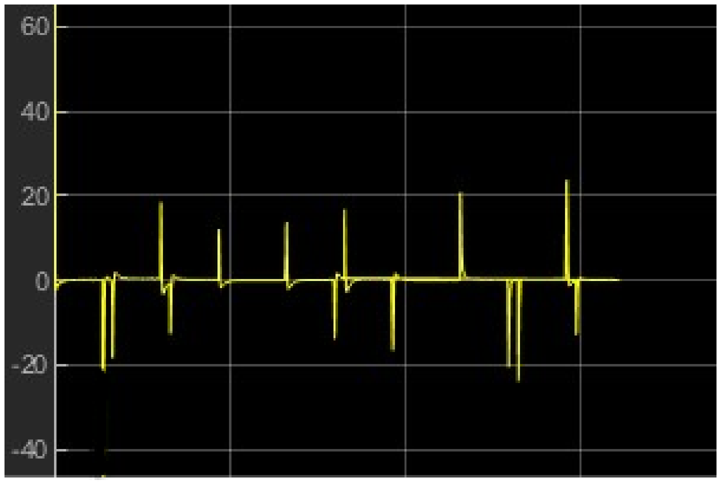


Figure 26: Input signal of Elbow Motor

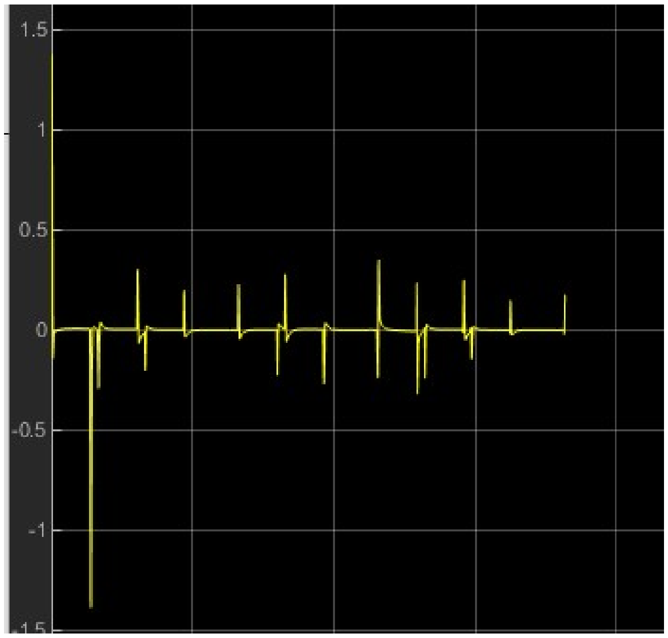
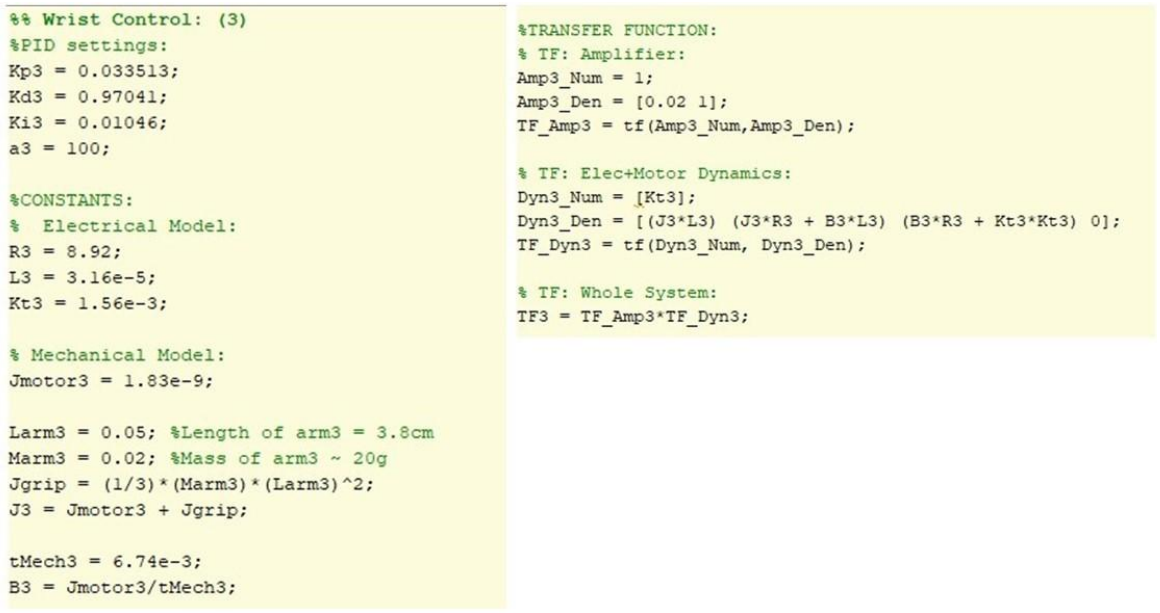


Figure 27: Input signal of Wrist Motor

5.4 Matlab Function:

Wrist System:

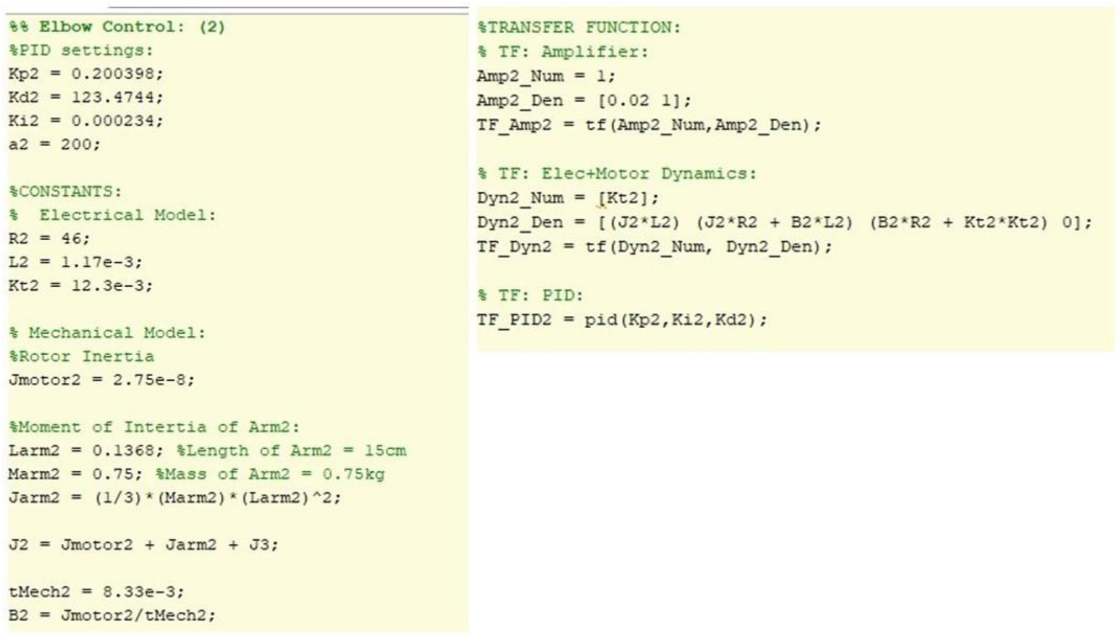


```
%% Wrist Control: (3)
%PID settings:
Kp3 = 0.033513;
Kd3 = 0.97041;
Ki3 = 0.01046;
a3 = 100;

%CONSTANTS:
%  Electrical Model:
R3 = 8.92;
L3 = 3.16e-5;
Kt3 = 1.56e-3;

% Mechanical Model:
Jmotor3 = 1.83e-9;

Larm3 = 0.05; %Length of arm3 = 3.8cm
Marm3 = 0.02; %Mass of arm3 ~ 20g
Jgrip = (1/3)*(Marm3)*(Larm3)^2;
J3 = Jmotor3 + Jgrip;

tMech3 = 6.74e-3;
B3 = Jmotor3/tMech3;
```

```
%TRANSFER FUNCTION:
% TF: Amplifier:
Amp3_Num = 1;
Amp3_Den = [0.02 1];
TF_Amp3 = tf(Amp3_Num,Amp3_Den);

% TF: Elec+Motor Dynamics:
Dyn3_Num = [Kt3];
Dyn3_Den = [(J3*L3) (J3*R3 + B3*L3) (B3*R3 + Kt3*Kt3) 0];
TF_Dyn3 = tf(Dyn3_Num, Dyn3_Den);

% TF: Whole System:
TF3 = TF_Amp3*TF_Dyn3;
```

Figure 28: Matlab Function for Wrist Motor System

Elbow System:

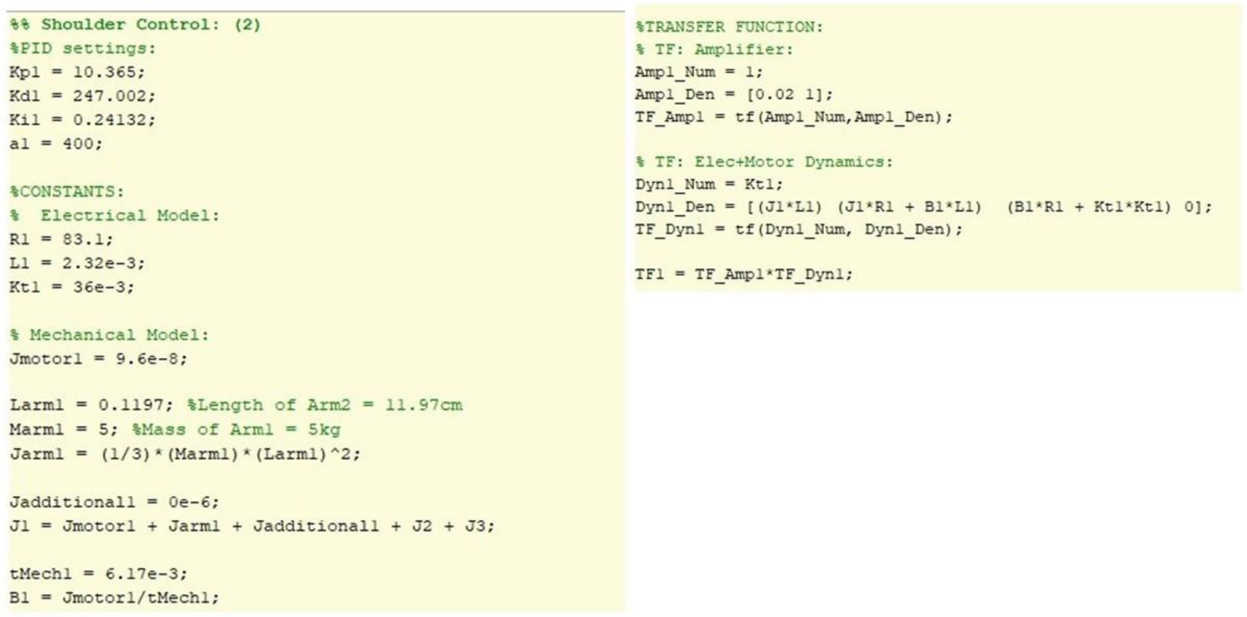


```
%% Elbow Control: (2)
%PID settings:
Kp2 = 0.200398;
Kd2 = 123.4744;
Ki2 = 0.000234;
a2 = 200;

%CONSTANTS:
%  Electrical Model:
R2 = 46;
L2 = 1.17e-3;
Kt2 = 12.3e-3;

% Mechanical Model:
%Rotor Inertia
Jmotor2 = 2.75e-8;

%Moment of Intertia of Arm2:
Larm2 = 0.1368; %Length of Arm2 = 15cm
Marm2 = 0.75; %Mass of Arm2 = 0.75kg
Jarm2 = (1/3)*(Marm2)*(Larm2)^2;

J2 = Jmotor2 + Jarm2 + J3;

tMech2 = 8.33e-3;
B2 = Jmotor2/tMech2;
```

```
%TRANSFER FUNCTION:
% TF: Amplifier:
Amp2_Num = 1;
Amp2_Den = [0.02 1];
TF_Amp2 = tf(Amp2_Num,Amp2_Den);

% TF: Elec+Motor Dynamics:
Dyn2_Num = [Kt2];
Dyn2_Den = [(J2*L2) (J2*R2 + B2*L2) (B2*R2 + Kt2*Kt2) 0];
TF_Dyn2 = tf(Dyn2_Num, Dyn2_Den);

% TF: PID:
TF_PID2 = pid(Kp2,Ki2,Kd2);
```

Figure 29: Matlab Function for Elbow Motor System

Shoulder System:



```
%% Shoulder Control: (2)
%PID settings:
Kp1 = 10.365;
Kd1 = 247.002;
Ki1 = 0.24132;
a1 = 400;

%CONSTANTS:
%  Electrical Model:
R1 = 83.1;
L1 = 2.32e-3;
Kt1 = 36e-3;

% Mechanical Model:
Jmotor1 = 9.6e-8;

Larm1 = 0.1197; %Length of Arm2 = 11.97cm
Marm1 = 5; %Mass of Arm1 = 5kg
Jarm1 = (1/3)*(Marm1)*(Larm1)^2;

Jadditional1 = 0e-6;
J1 = Jmotor1 + Jarm1 + Jadditional1 + J2 + J3;

tMech1 = 6.17e-3;
B1 = Jmotor1/tMech1;
```

```
%TRANSFER FUNCTION:
% TF: Amplifier:
Amp1_Num = 1;
Amp1_Den = [0.02 1];
TF_Amp1 = tf(Amp1_Num,Amp1_Den);

% TF: Elec+Motor Dynamics:
Dyn1_Num = Kt1;
Dyn1_Den = [(J1*L1) (J1*R1 + B1*L1)  (B1*R1 + Kt1*Kt1) 0];
TF_Dyn1 = tf(Dyn1_Num, Dyn1_Den);

TF1 = TF_Amp1*TF_Dyn1;
```

Figure 30: Matlab Function for Shoulder Motor System