Control System Design for Virtual Quality Control Robot

White-Paper Updated – April 14th, 2021

Yinlong Gao, ECE, University of BC, Vancouver, BC, Canada

Quinlan Cheng, ECE University of BC, Vancouver, BC, Canada

Eric Wu, ECE, University of BC, Vancouver, BC, Canada

Abstract

This report describes the control system of the virtual quality control robot. In this report, the procedures and results of the design will be illustrated. The report contains five parts, section 1: linear model development, section 2: PID tuning, section 3: robot kinematics, section 4: path planning, and section 5: co-simulation.

Nomenclature

ISR:	interrupt service routine
MA1:	motor attaching to the first arm
MA2:	motor attaching to the second arm
MG1:	motor attaching to the gripper base
MG2:	motor attaching to the gripper
DD:	desired displacement
DA:	desired angle
T1:	arm1 angular displacement
T2:	arm2 angular displacement
T3:	gripper angular displacement

1. Control System Development

1.1. Linear models of motors and motor driver circuits.

Linear models of motors and motor driver circuits are developed in electrical and mechanical reports. *Figure 1.1.1* and *Figure 1.1.2* illustrate the linear models of motors and amplifiers, and they are used for building a control system model. Motor parameters are provided in Appendix.

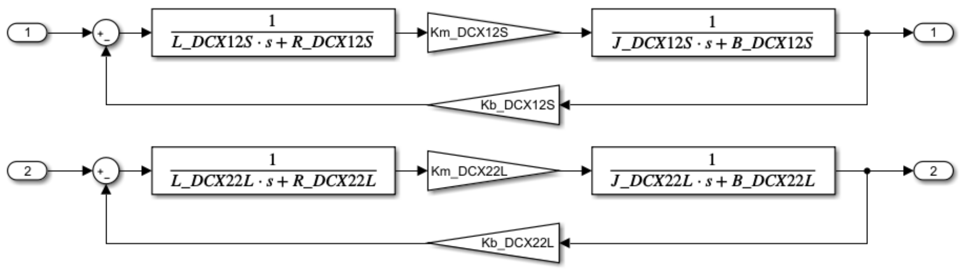

1
$\frac{1}{L\_DCX12S \cdot s + R\_DCX12S}$
Km_DCX12S
$\frac{1}{J\_DCX12S \cdot s + B\_DCX12S}$
1
Kb_DCX12S
2
$\frac{1}{L\_DCX22L \cdot s + R\_DCX22L}$
Km_DCX22L
$\frac{1}{J\_DCX22L \cdot s + B\_DCX22L}$
2
Kb_DCX22L

Figure 1.1.1: DCX22L and DCX12S linear models

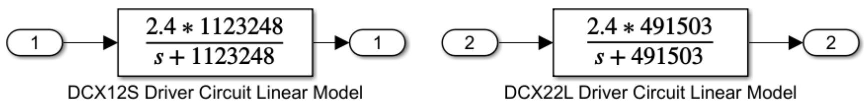

1
$\frac{2.4 * 1123248}{s + 1123248}$
1
DCX12S Driver Circuit Linear Model
2
$\frac{2.4 * 491503}{s + 491503}$
2
DCX22L Driver Circuit Linear Model

Figure 1.1.2: DCX22L driver circuit and DCX12S driver circuit linear model

1.2. Linear model of encoders

The encoders used have a counts per turn of 1024, resulting in 4096 states per turn, which in turn gives a resolution of $\frac{2*\pi}{16384}$ which is used for the system feedback. The linear model of the encoder is represented by an integrator and a quantizer, and the quantizer has a quantization interval the resolution of the encoder, $\frac{2*\pi}{16384}$. The linear model is demonstrated in *Figure 1.2.1*.

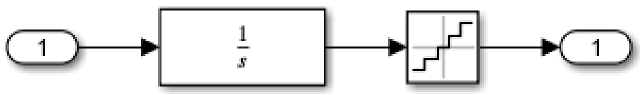


Figure 1.2.1: encoder linear model

1.3. PID controller development

The C code of PID is implemented showing in Appendix. The ISR takes 631 clock cycles per routine, and the Arduino has a frequency of 20MHz[1]. That means the ISR takes 31.55us for every routine. 10 is used for the N-hat of the derivative calculation. Since Unity Gain Filter has a pole of $\frac{4}{N-hat*dt}$, where dt is the ISR rate, a pole of 21918 for Unity Gain Filter is obtained. It ends up an ideal PID transfer function of $Kp + \frac{KI}{s} + KD * \frac{21918*s}{s+21918}$. A MATLAB user defined function is implemented to achieve the same performance, and the code is provided in Appendix. *Figure 1.3.1.* shows the PID user defined function block.

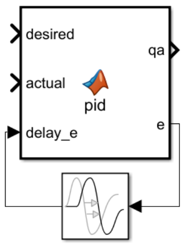


Figure 1.3.1: PID user defined function block

The same step response is fed to the user defined PID block and ideal PID block as shown in Figure 1.3.2. They have the same output as shown in Figure 1.3.3.

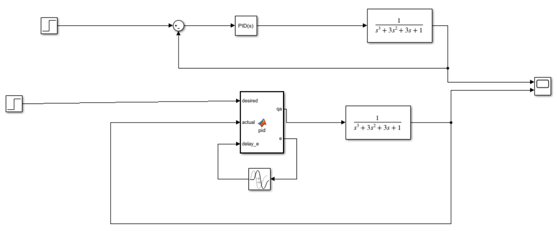


Figure 1.3.2: PID user defined function block and PID block comparison circuit

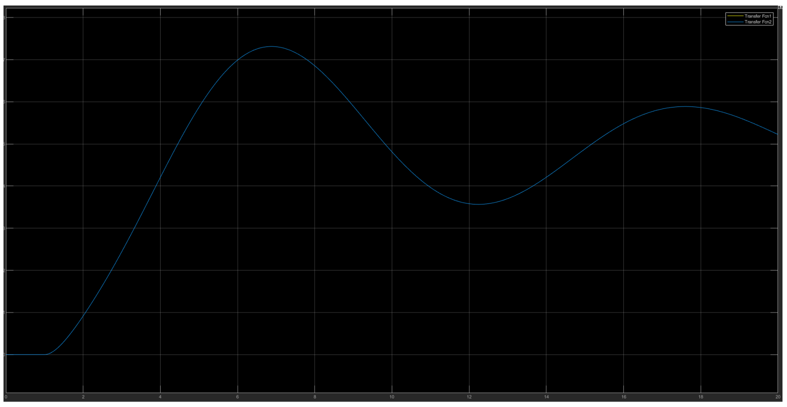


Figure 1.3.3: PID user defined function block and PID block output comparison

1.4. Linear model of the control system

After all the linear models are developed, they are combined together to provide a control system linear model for motor DCX22L and DCX12S respectively. Saturation blocks are added, since the Arduino can only output -5 volts to 5 volts and the motor driver circuit can only output -12 volts to 12 volts. *Figure 1.4.1* demonstrates the control system linear model for DCX22L and DCX12S, and the model does not include arm and gripper attachments (no external inertia included).

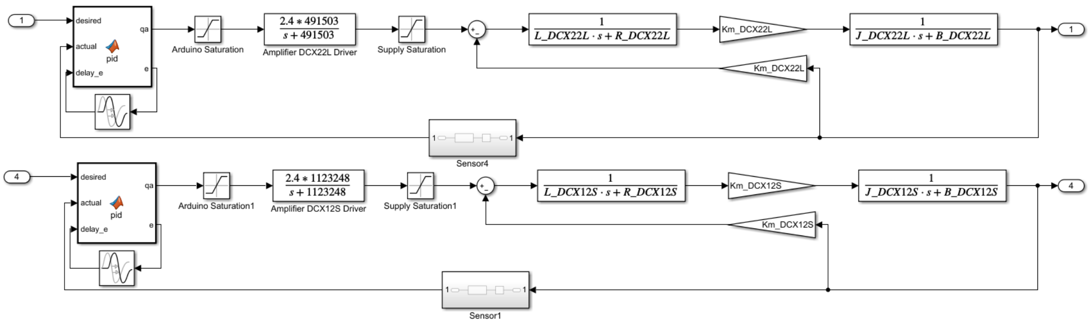
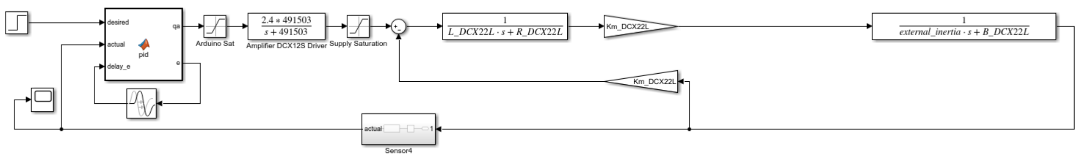


Figure 1.4.1: Linear model of the system

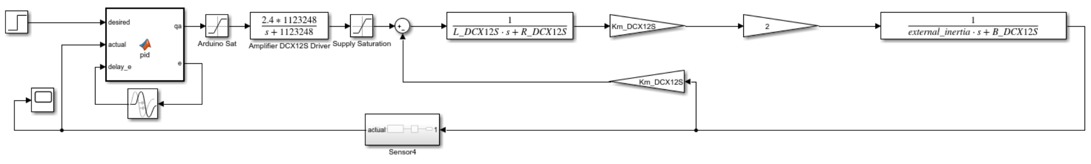
2. PID tuning

In the system, there are two DCX22L motors and two DCX12S motors. DCX22L motors are used for two arms, and DCX12S motors are used for the gripper and turning the gripper. Because they have different external inertia, they are tuned separately. Since the external inertia is significantly larger than motor inertia, motor inertia has been neglected. *Figure 2.1.* shows the MA1 and MA2 control system linear model. *Figure 2.2.* shows the MG1 and control system linear model, and *Figure 2.3.* shows the MG2 control system linear model.



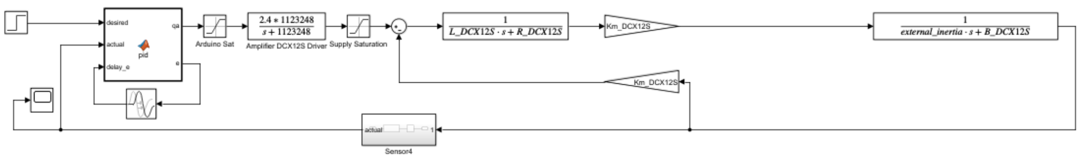
desired
actual
delay_e
qa
e
pid
Arduino Sat
$\frac{2.4 * 491503}{s + 491503}$
Amplifier DCX12S Driver
Supply Saturation
$\frac{1}{L\_DCX22L \cdot s + R\_DCX22L}$
Km_DCX22L
$\frac{1}{external\_inertia \cdot s + B\_DCX22L}$
Km_DCX22L
actual
1
Sensor4

Figure 2.1. MA1 and MA2 control system linear model



desired
actual
delay_e
qa
e
pid
Arduino Sat
$\frac{2.4 * 1123248}{s + 1123248}$
Amplifier DCX12S Driver
Supply Saturation
$\frac{1}{L\_DCX12S \cdot s + R\_DCX12S}$
Km_DCX12S
2
$\frac{1}{external\_inertia \cdot s + B\_DCX12S}$
Km_DCX12S
actual
1
Sensor4

Figure 2.1. MG1 control system linear model



desired
actual
delay_e
qa
e
pid
Arduino Sat
$\frac{2.4 * 1123248}{s + 1123248}$
Amplifier DCX12S Driver
Supply Saturation
$\frac{1}{L\_DCX12S \cdot s + R\_DCX12S}$
Km_DCX12S
$\frac{1}{external\_inertia \cdot s + B\_DCX12S}$
Km_DCX12S
actual
1
Sensor4

Figure 2.3. MG2 control system linear model

Since we have a gear set ratio of 1:2, the output torque is doubled. The gain of 2 represents the fact that gear set generates a double torque (This will be applied for MA1, MA2, MG1 PID tuning)

2.1. MA1 tuning

The external inertia for MA1 is 0.0075 $kg * m^2$. The open loop transfer function is:

```
>> zpk(sys)

ans =

             3.7518e10 (s+9583) (s+0.0001171)
  ---------------------------------------------------
  s (s+4.915e05) (s+9583)^2 (s+0.0763) (s+0.0001171)
```

The Bode plot is obtained in *Figure 2.1.1.*

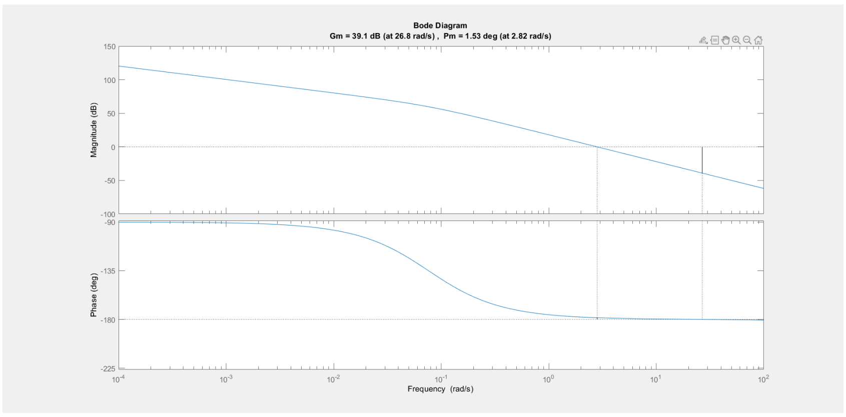


Figure 2.1.1. Bode plot of MA1 system

$$Ku = 10^{\frac{Gm}{20}} = 90$$

By applying *PID-10* [2], K, zero1 and zero2 are set to be 9000, (0.04-0.04*j), and (0.04-0.04*j). K is large because a quicker response is desired. Kp, Ki, and Kd are obtained from *PID-10*[1]. This is the starting point, and the PID parameters are tuned based on the website [3], [4].

Kp, Kd, and Ki are 0.48, 0.6, and 0.000016 respectively after tuning the PID parameter. *Figure 2.1.2.* shows the step response of MA1 after tuning the PID parameters, and *Table 2.1.1.* shows the step response details.

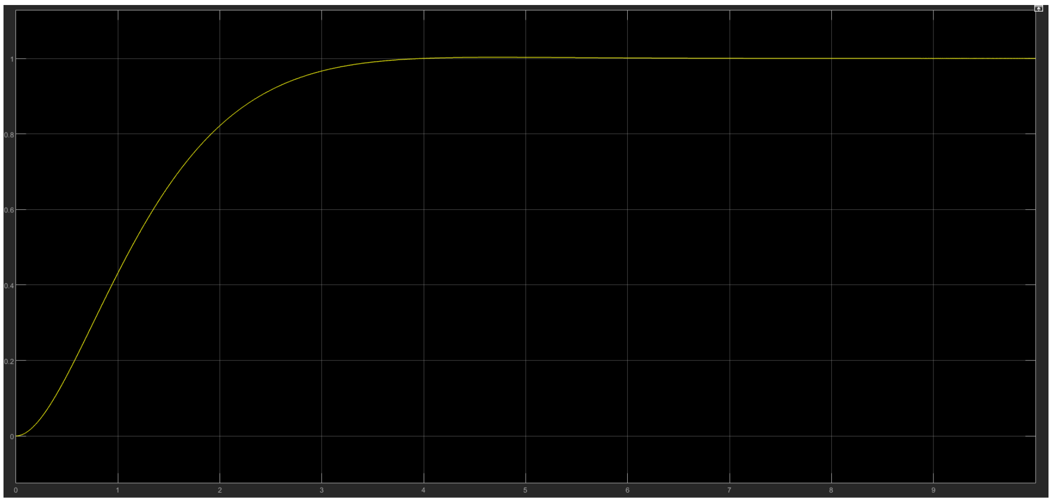


Figure 2.1.2. Step response of MA1 system

Rise time	3.5s
Settle time	4s
Overshoot	0
Steady state error	0

Table 2.1.1. Step response details of MA1 system

2.2.MA1 tuning

The external inertia for MA1 is 0.0039 $kg * m^2$. The open loop transfer function is:

```
>> zpk(sys)

ans =

                 7.215e10 (s+9583) (s+0.0002252)
  --------------------------------------------------------
  s (s+4.915e05) (s+9583) (s+9583) (s+0.1467) (s+0.0002252)
```

The Bode plot is obtained in *Figure 2.2.1.*

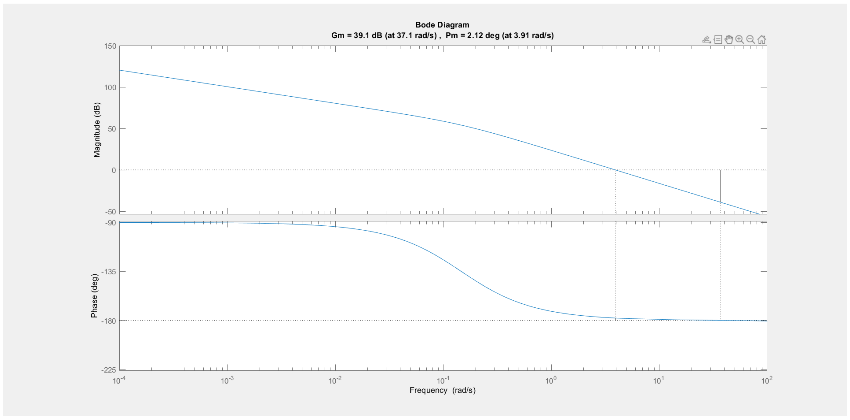


Figure 2.2.1. Bode plot of MA2 system

$$Ku = 10^{\frac{Gm}{20}} = 90$$

By using the same method of tuning the MA1 system, K is initially set to 9000. Zero1 and zero2 are set to (0.07+0.07*j) and (0.07-0.07*j). *PID-10* [1] is used to calculate starting point PID parameters. Kp, Kd, and Ki are tuned to 0.8, 0.7, 0.00002 respectively. The step response is shown in *Figure 2.2.2.*

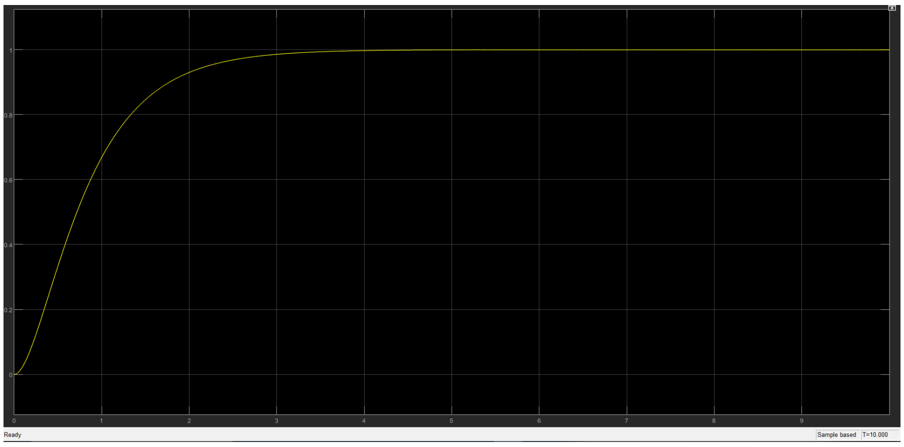


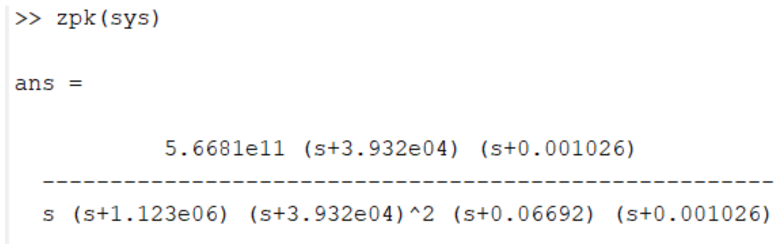
Figure 2.2.2. Step response of MA1 system

Rise time	3.8s
Settle time	4.3s
Overshoot	0
Steady state error	0

Table 2.2.1. Step response details of MA1 system

2.3.MG1 tuning

The external inertia for MG1 is 0.00001 $kg * m^2$, and the open loop transfer function is:



```
>> zpk(sys)

ans =

             5.6681e11 (s+3.932e04) (s+0.001026)
  -----------------------------------------------------
  s (s+1.123e06) (s+3.932e04)^2 (s+0.06692) (s+0.001026)
```

The Bode plot is obtained in *Figure 2.3.1.*

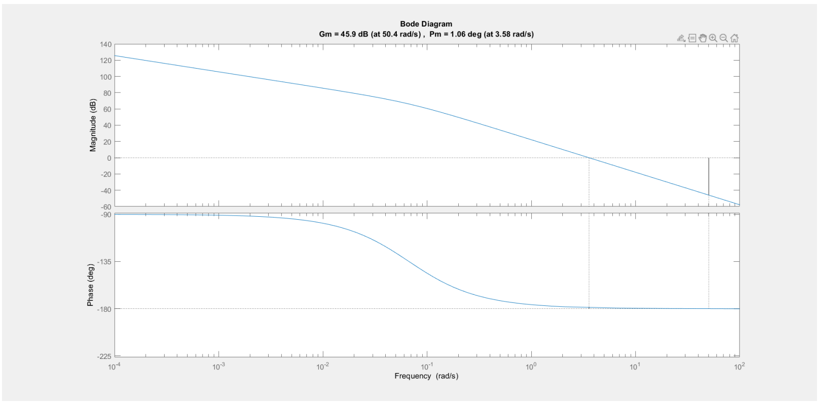


Figure 2.3.1. Bode plot of MG1 system

$$Ku = 10^{\frac{Gm}{20}} = 197$$

K is set to 90. Two zeros are set to (0.033+0.033*j) and (0.033-0.033*j) initially. By using the same method, Kp, Kd, and Ki are tuned to 1.2, 0.9, 0.00003. Figure 2.3.2 illustrates the step response.

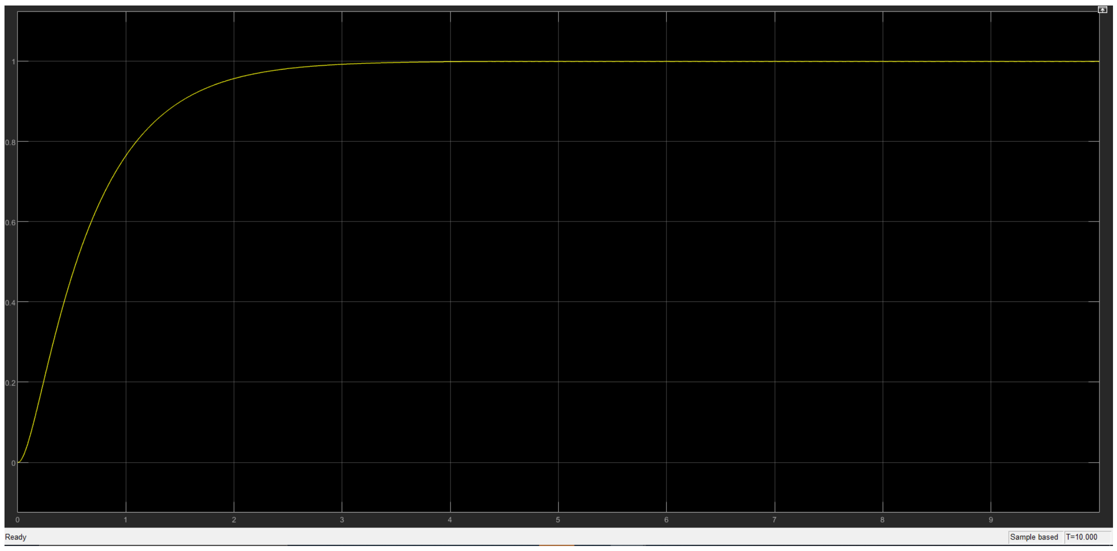


Figure 2.3.2. Step response of MG1 system

Rise time	3.6s
Settle time	4.2s
Overshoot	0
Steady state error	0

Table 2.2.1. Step response details of MG1 system

2.4.MG2 tuning

The external inertia for MG2 is 0.00001 $kg * m^2$, and the open loop transfer function is:

```
>> zpk(sys)

ans =

             8.2444e10 (s+9583) (s+0.0002252)
  ---------------------------------------------------------
  s (s+1.123e06) (s+9583)^2 (s+0.03955) (s+0.0002252)
```

The Bode plot is obtained in *Figure 2.3.1*.

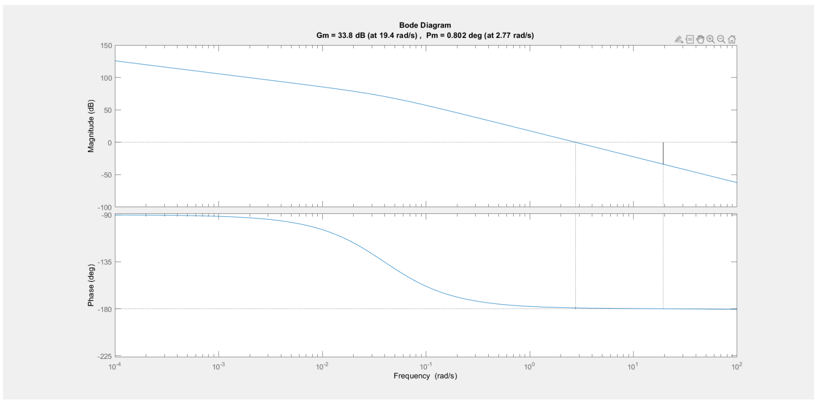


Figure 2.1.1. Bode plot of MG2 system

$$Ku = 10^{\frac{Gm}{20}} = 49$$

K is set to 25. Two zeros are set to (0.02+0.02*j) and (0.02-0.02*j) initially. By using the same method, Kp, Kd, and Ki are tuned to 0.32, 0.39, 0.00000003. Figure 2.3.2 illustrates the step response.

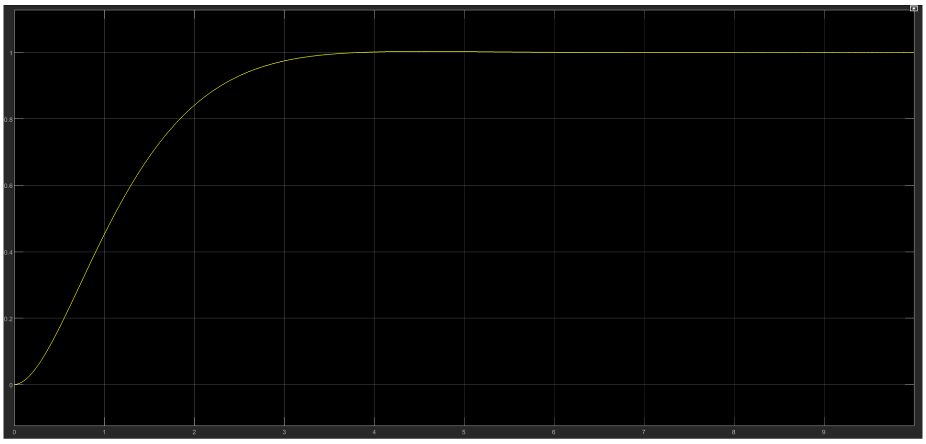


Figure 2.3.2. Step response of MG1 system

Rise time	3.6s
Settle time	3.8s
Overshoot	0
Steady state error	0

Table 2.2.1. Step response details of MG2 system

3. Robot kinematics

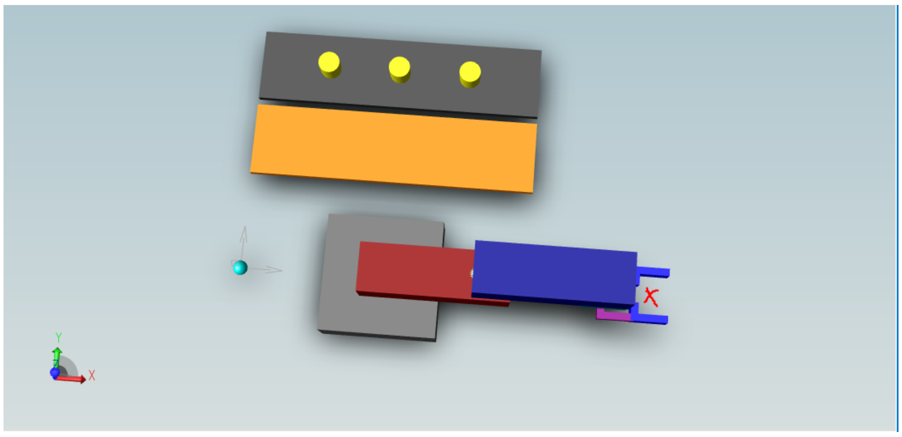


Figure 3.1. System overview

In this project, inverse kinematics is used to calculate and control the position of the robot. The desired X axis displacement and desired Y axis displacement are input into the algorithm, and it outputs corresponding angles as to be used for the PID control system..

Assumption

In *Figure 3.1.*, a system overview is given. X axis and Y axis will be the same as in *Figure3.1.* for the coming discussion. The origin is at the joint between red arm and gray base. Red "X" mark is the place where the gripper can grip a marshmallow, and it will be moved to desired location. In the figure, two arms and the gripper have an angular displacement of 0 initially. Counter clockwise rotation is defined as positive.

Strategy

Firstly, the system can be simplified as shown in *Figure 3.2.*

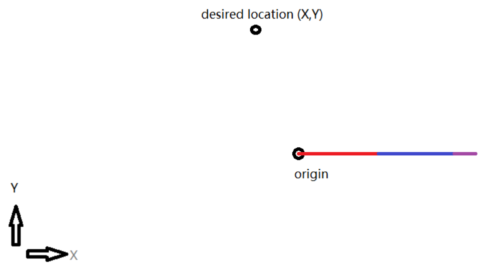


Figure 3.2. simplified system drawing

Red line, blue line, and purple line represents red arm, blue arm and gripper respectively. Two arms have the length of 15 cm, and a gripper has a length of 5cm. The strategy is to make the gripper parallel to the line between desired location and origin. Because two arms have the same length, we make two arms always form an isosceles triangle. The graphic explanation is shown in *Figure 3.3*.

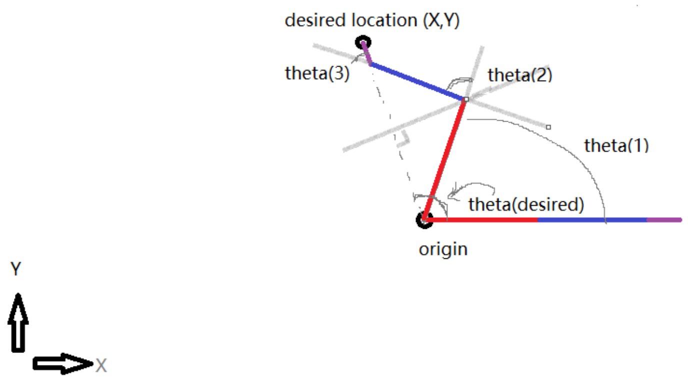


Figure 3.3. graphic explanation

Calculation

$$DD = \sqrt{X^2 + Y^2}$$

$$DA = \arctan\ arctan\left(\frac{Y}{X}\right) \qquad if\,(X\ is\ postive)$$

$$= 180 + \arctan\ arctan\left(\frac{Y}{X}\right) \quad if\ (X\ is\ negative)$$

$$T1 = DT - arcos\left(\frac{\left(\frac{DD-5}{2}\right)}{(15)}\right)$$

$$T2 = [180 - 2 * \left(90 - arcos\left(\frac{\left(\frac{DD-5}{2}\right)}{(15)}\right)\right)]$$

$$T3 =- (DA - T1)$$

4. Path Planning

The robot has an original location at (0, 12.68). Three marshmallows are located at (0,22.68), (10,22.68), and (-10,22.68). The robot will move to the first object, and grip the first marshmallow. The gripper will rotate 180 degrees, and drop the marshmallow into the garbage chute. To avoid collision with other marshmallows. The arm will first move backwards, then it will go to the next location of marshmallow. The robot sequence of operations will be, go to the position of the marshmallow, grip the marshmallow, rotate the gripper, drop the marshmallow, and reposition the gripper, for every marshmallow.

5. Co-simulation

Figure 5.1. shows the Simulink circuit used for co-simulation. Four repeating sequential inputs are used, and they are representing the data calculated by robot kinematics function. The performance of the Co-simulation is discussed and demonstrated in the demonstration video.

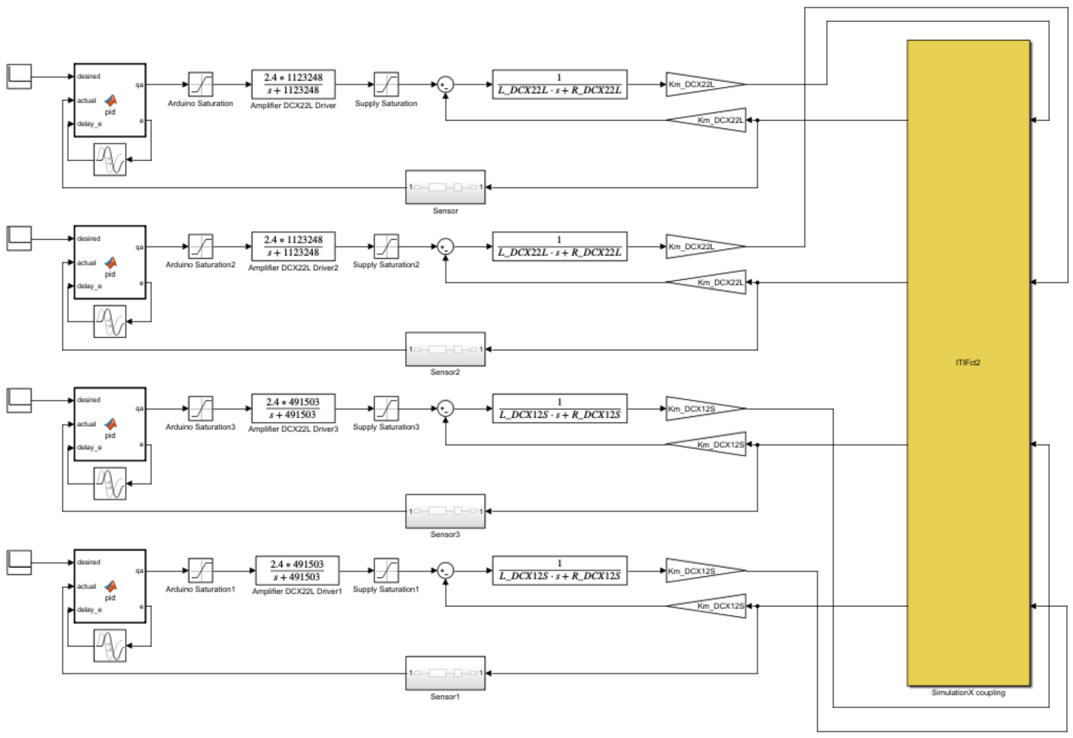


Figure 5.1. Co-Simulation Simulink model.

Figure 5.2, 5.3 5.4, and 5.5. show the desired positions (blue lines) and actual positions (yellow lines) of four motors from co-simulation. There are some overshoots and errors, but they only happen in transition steps (during arm moving or gripper base rotating). Before the gripper grips, there are 0 overshoot and 0 steady state error.

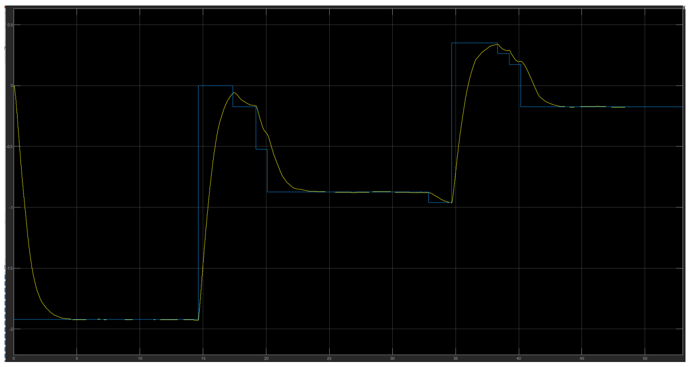


Figure 5.2. MA1 actual values and desired values comparison during co-simulation

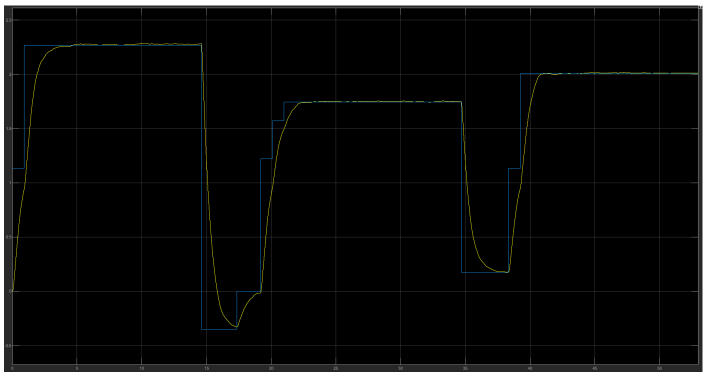


Figure 5.3. MA2 actual values and desired values comparison during co-simulation

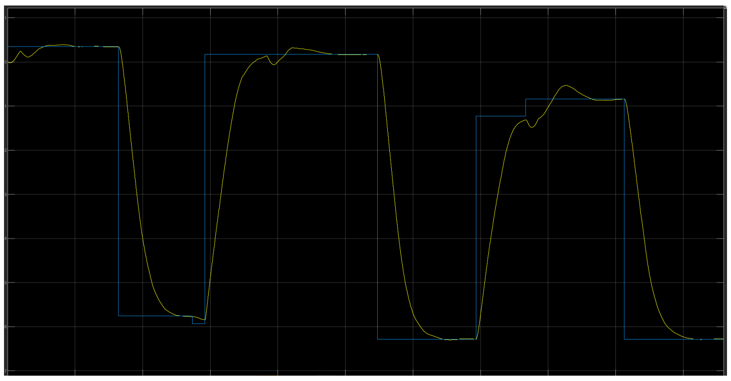


Figure 5.4. MG1 actual values and desired values comparison during co-simulation

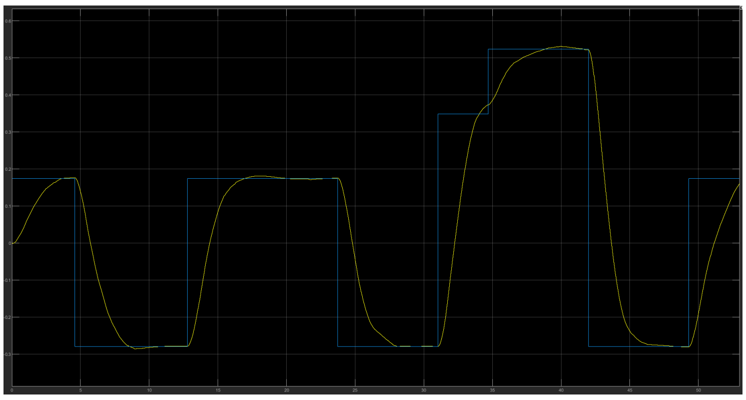


Figure 5.5. MG2 actual values and desired values comparison during co-simulation

Reference

[1]:https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[2]: PID-10 on Canvas

[3]: https://blog.yokogawa.com/blog/how-to-loop-tuning

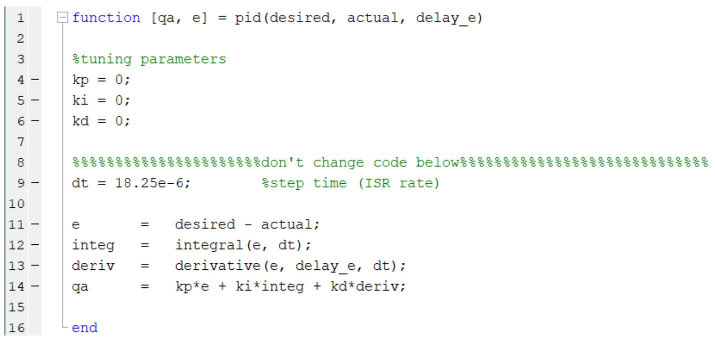
[4]: https://en.wikipedia.org/wiki/PID_controller

Appendix

motor parameters

Parameters	Symbol	DCX 12S	DCX 22L
Resistance (Ω)	R	46	1.84
Inductance (H)	L	1.17 e-3	0.192 e-3
Torque constant (Nm/A)	Km	1.23 e-3	22.9 e-3
Back emf constant (V*s/rad)	Kb	12.3 e-3	23.0 e-3
Motor damping coefficient (rad/s*N*m)	B	10.3 e-9	878 e-9
Rotor inertia (kg*m^2)	J	27.5 e-9	900 e-9

PID Matlab user defined function



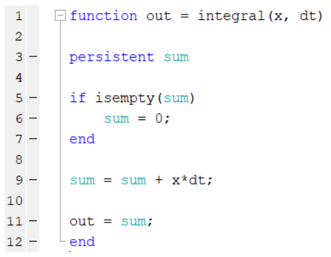
```matlab
function [qa, e] = pid(desired, actual, delay_e)

%tuning parameters
kp = 0;
ki = 0;
kd = 0;

%%%%%%%%%%%%%%%%%%%%%%%don't change code below%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dt = 18.25e-6;        %step time (ISR rate)

e       =   desired - actual;
integ   =   integral(e, dt);
deriv   =   derivative(e, delay_e, dt);
qa      =   kp*e + ki*integ + kd*deriv;

end
```



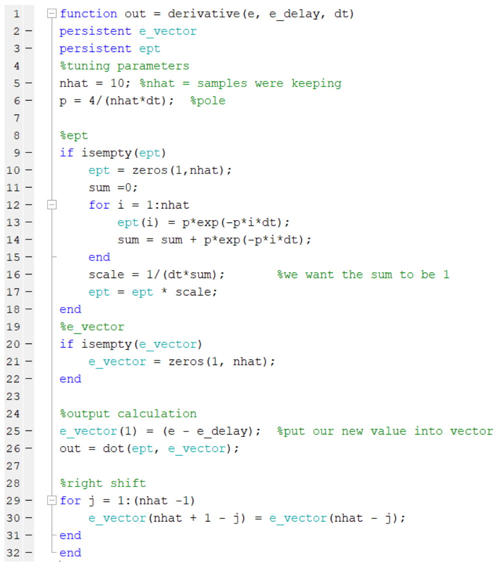
```matlab
function out = integral(x, dt)

persistent sum

if isempty(sum)
    sum = 0;
end

sum = sum + x*dt;

out = sum;
end
```



```matlab
function out = derivative(e, e_delay, dt)
persistent e_vector
persistent ept
%tuning parameters
nhat = 10; %nhat = samples were keeping
p = 4/(nhat*dt);  %pole

%ept
if isempty(ept)
    ept = zeros(1,nhat);
    sum =0;
    for i = 1:nhat
        ept(i) = p*exp(-p*i*dt);
        sum = sum + p*exp(-p*i*dt);
    end
    scale = 1/(dt*sum);       %we want the sum to be 1
    ept = ept * scale;
end
%e_vector
if isempty(e_vector)
    e_vector = zeros(1, nhat);
end

%output calculation
e_vector(1) = (e - e_delay);  %put our new value into vector
out = dot(ept, e_vector);

%right shift
for j = 1:(nhat -1)
    e_vector(nhat + 1 - j) = e_vector(nhat - j);
end
end
```

PID C code

```c
double kp = 0;
double ki = 0;
double kd = 0;
double n = 0;
double sensor, setpoint, output;
double currentT, prevT;
double dt, integ, deriv, p, sum, e, e_delay, multi, sum2;
double e_array[10]; //size is nhat
double ept_array[10]; //size is nhat
```

```
void setup() {
      pinMode(A0, INPUT);     //sensor
      pinMode(A1, INPUT);     //desired value
      pinMode(A2, OUTPUT);    //output

}

 double pid(double in){
        currentT = millis();  //get running time
        dt = currentT - prevT;  //ISR rate

        //integral calculation
        e = setpoint - in;
        integ += e * dt;

        //derivative calculation
        //ept array
        sum = 0;
        p = 4/(10*dt);
        for (int i; i<11; i++)
        {
         ept_array[i] = p*exp(-p*i*dt);
         sum = sum +p*exp(-p*i*dt);
        }
        for (int k; k<11; k++)
        {
        ept_array[k] = ept_array[k] * 1/(dt*sum);
        }

        //e_array
        e_array[1] = (e - e_delay);
        sum2 = 0;
        for (int s; s<11; s++)
        {
         multi = ept_array[s]*e_array[s];
         sum2 = sum2 + multi;
         }
        deriv = sum2;

        //final output
        double out = kp*e +ki*integ - kd*deriv*n/(deriv+n*integ);

        //keeping values for next calculation + e_array shifting
        for (int j; j<10 ;j++)
        {
         e_array[11-j] = e_array[10-j];
        }
        e_delay = e;
        prevT = currentT;

        return out;
 }
```