ME132: DYNAMIC SYSTEMS AND FEEDBACK EV3 FINAL LAB WRITE-UP

LAB SECTION 001 (TUE 5-6PM)

December 15, 2019

SUBMISSION BY

Mingjun WU

EDMUND NARH

1 Motor Identification via Step-Response (Part I in Lab 1)

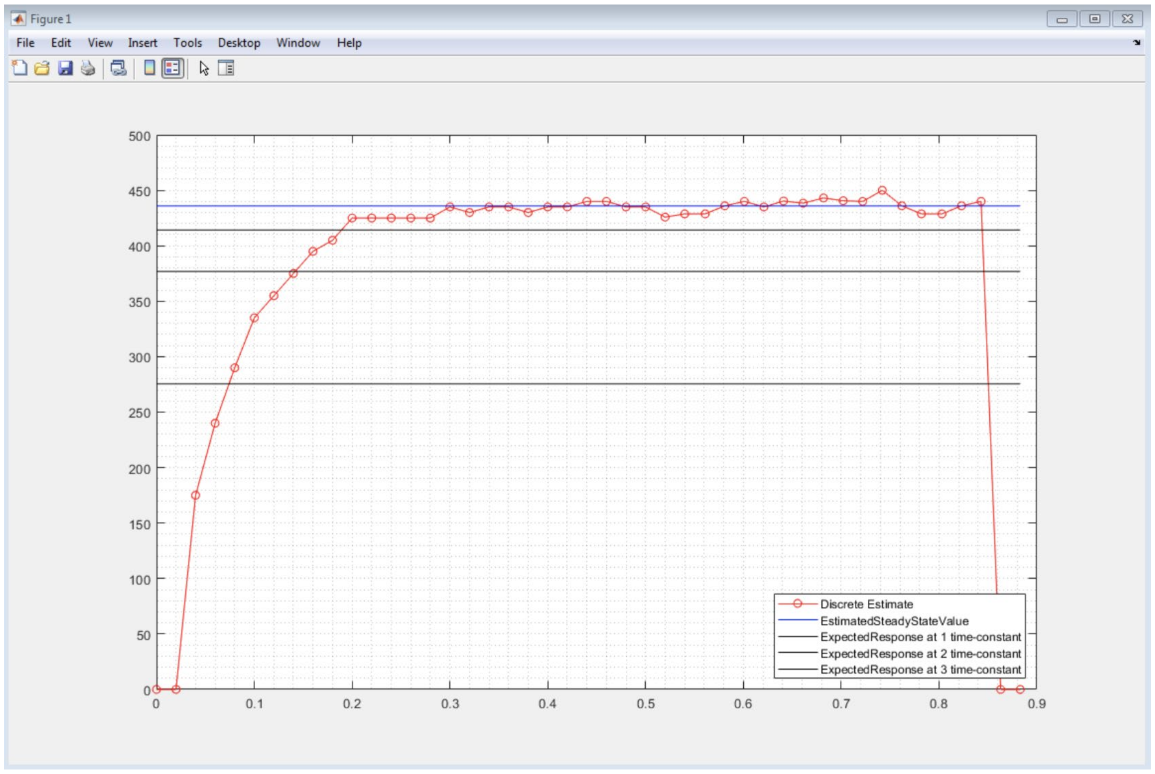
1. Which motor did you use? (large or medium)

- Large.

2. c-code (cut/paste, cleaned up)

```
#pragma config(Motor,  motorA,   mA,  tmotorEV3_Large, openLoop, encoder)
task main()
{
        int thetaD = 0;
        float Vbar = 50;
        int ExperimentLength = 900;
        int SamplePeriod = 20;
        clearTimer(T1);
        clearTimer(T2);
        resetMotorEncoder(mA);
        writeDebugStreamLine("Mdata = [%i, %i",time1(T1), getMotorEncoder(mA));
        motor[mA] = Vbar;
        while(time1[T1]<=ExperimentLength)
        {
                if(time1[T2] >= SamplePeriod)
                {
                        clearTimer(T2);
                        thetaD = getMotorEncoder(mA);
                        writeDebugStreamLine("%i, %i", time1(T1), thetaD);
                }
        }
        writeDebugStreamLine("];");
        writeDebugStreamLine("Vdata = %f;", Vbar);
}
```

3. one graph from Matlab post-processing



4. estimates for (τ_m,γ_m).

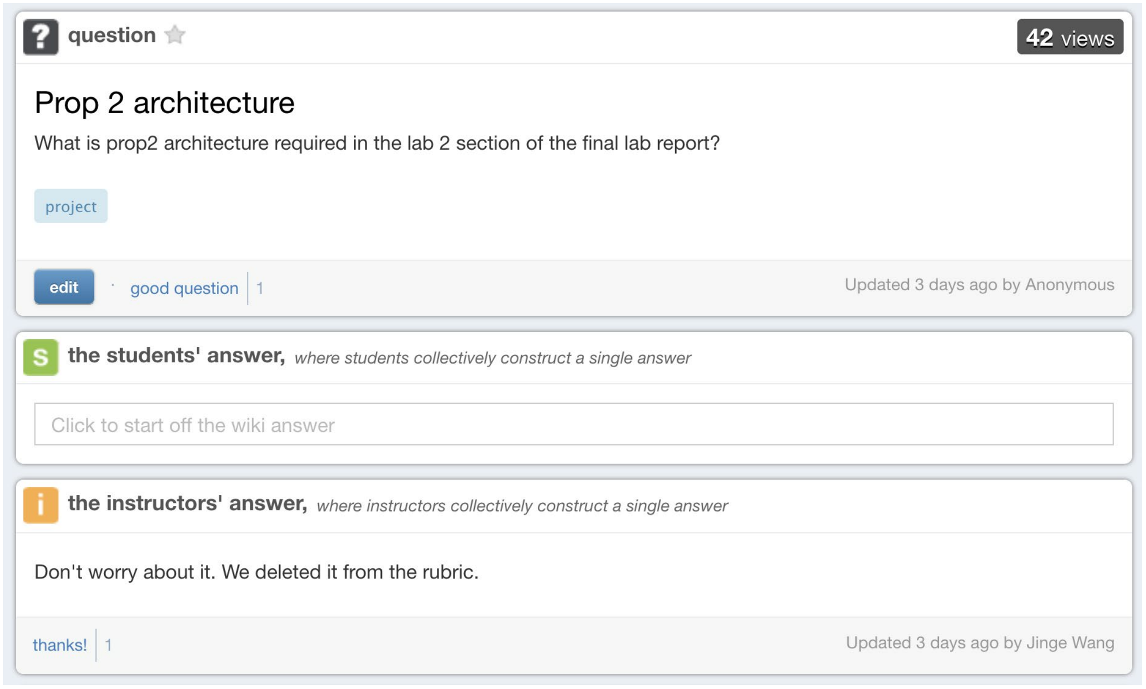
- $\tau_m = 0.067$ seconds
- $\gamma_m = 8.7408$

5. 1-2 sentence summary (no longer)

The motor begins to accelerate when there is a step voltage input, and the motor eventually reaches a constant velocity. The motor is powered by the voltage applied to it and certain voltages correspond to certain velocities.

2 Proportional Control of Speed (Part II in Lab 1) (also some overlap with Lab 2)

1. Use Prop2 architecture


question 42 views

Prop 2 architecture

What is prop2 architecture required in the lab 2 section of the final lab report?

project

edit · good question | 1 Updated 3 days ago by Anonymous

the students' answer, *where students collectively construct a single answer*

Click to start off the wiki answer

the instructors' answer, *where instructors collectively construct a single answer*

Don't worry about it. We deleted it from the rubric.

thanks! | 1 Updated 3 days ago by Jinge Wang

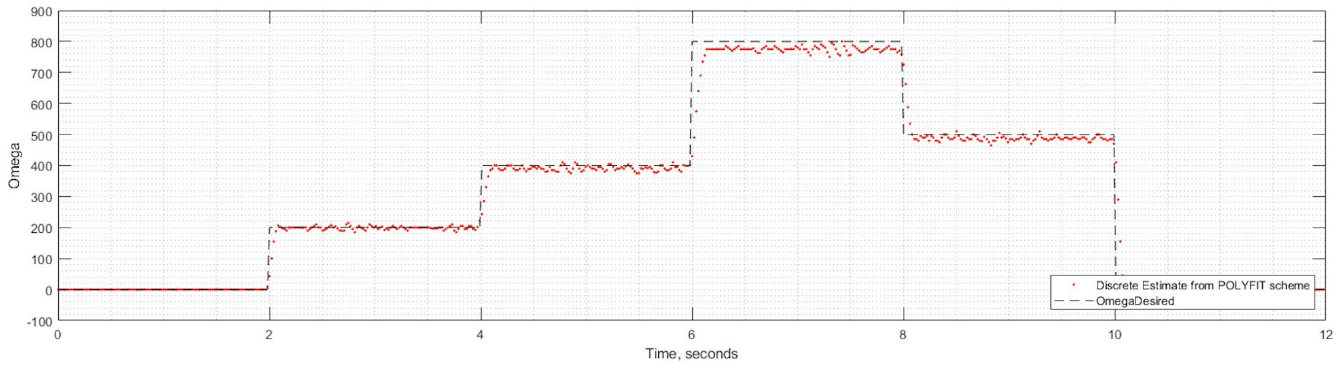
2. c-code (cut/paste, cleaned up)

```
#pragma config(Motor, motorA, mA, tmotorEV3_Large, openLoop, encoder)
task main()
{
        int thetaDeg, thetaDegPrevious = 0;
        float voltage, OmegaEst, OmegaDes;
        int ExperimentLength = 12000;
        int SamplePeriod = 20;
        float SampleTime = SamplePeriod*0.001;
        float K1, K2, ii;
        float gammaM = 8.7408;
        float tauM = 0.067;
        K2 = 1/gammaM;
        K1 = 1/gammaM + K2;
        int OmegaDesArray[] = {200, 400, 800, 500,0};
        setMotorBrakeMode(mA, motorCoast);
        clearTimer(T1);
        clearTimer(T2);
        clearTimer(T3);
        resetMotorEncoder(mA);
        OmegaEst = 0;
        ii = 0;

        OmegaDes = 0;
        motor[mA] = voltage;
        writeDebugStreamLine("Mdata = [0,0,%f, %f", voltage, OmegaDes);
        while(time1[T1]<=ExperimentLength)
```

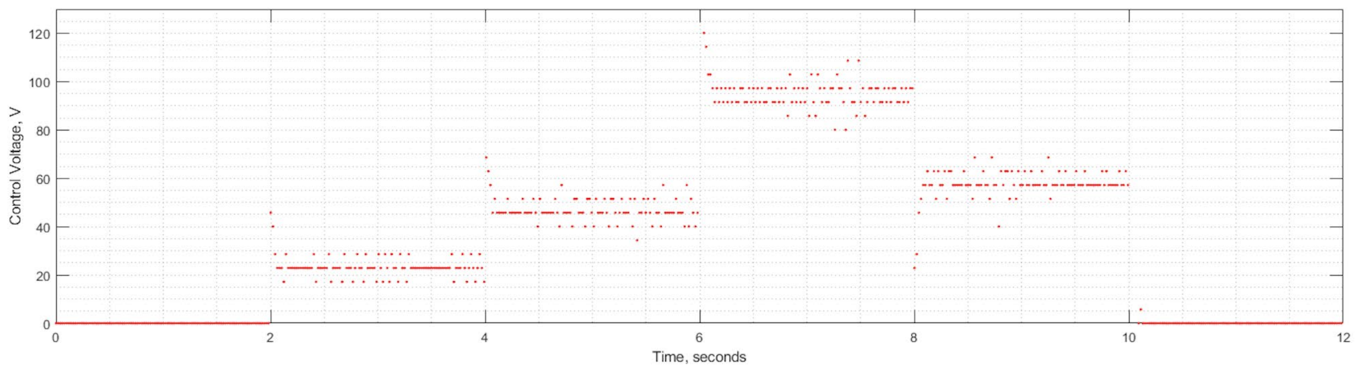
```
	{
		if (time1[T3] >= 2000)
		{
			clearTimer(T3);
			OmegaDes = OmegaDesArray[ii];
			ii = ii+1;
		}

		if (ii >= 5)
		{
			ii = 0;
		}

		if (time1[T2] >= SamplePeriod)
		{
			clearTimer(T2);
			thetaDeg = getMotorEncoder(mA);
			OmegaEst = (thetaDeg - thetaDegPrevious)/SampleTime; // use current and last theta
			voltage = K1*OmegaDes - K2*OmegaEst;
			motor[mA] = voltage;
			thetaDegPrevious = thetaDeg; // save theta for next time through
			writeDebugStreamLine("%i, %i, %f, %f", time1(T1), thetaDeg, voltage, OmegaDes);
		}
	}
	writeDebugStreamLine("];");
}
```

3. one interesting reference trajectory (r, representing $\Omega_{desired}$)

(a) one graph (from Matlab post-processing) of r and Ω versus t



(b) one graph showing V (voltage) versus t.



4. 1-2 sentence summary (no longer)

The motor adjusted its velocity very quickly to the desired velocity when proportional control was applied. From of the plots above, we can therefore conclude that the velocity of the motor increases with increasing voltage while the voltage is constant, the velocity is also constant (directly proportional).

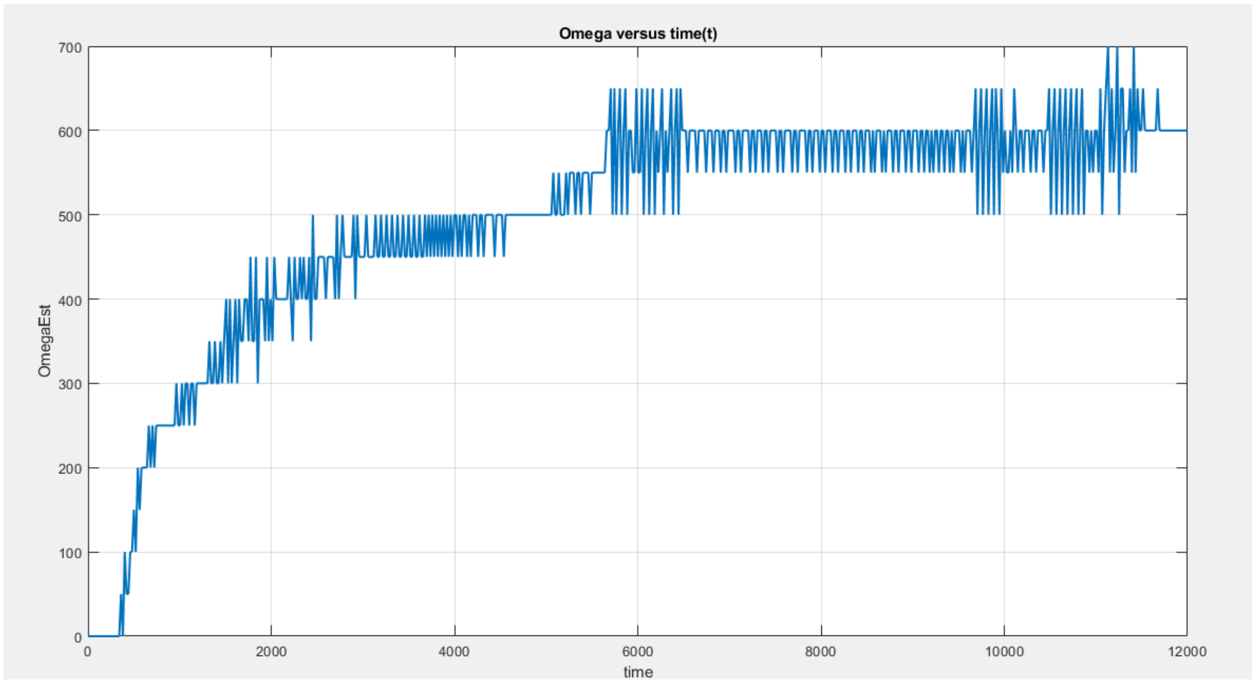
3 Frictionless and spring-like behavior (Lab 2)

1. Cut/paste of c-code (cleaned up) for frictionless behavior

```
#pragma config(Motor, motorA, mA, tmotorEV3_Large, openLoop, encoder)
task main()
{
        int thetaDeg, thetaDegPrevious = 0;
        float voltage;
        float OmegaEst = 0;
        int ExperimentLength = 12000;
        int SamplePeriod = 20;
        float SampleTime = SamplePeriod*0.001
        float K1, K2;
        float gammaM = 9;
        K2 = 1/gammaM;

        K1 = 1/gammaM + K2;
        setMotorBrakeMode(mA, motorCoast);
        clearTimer(T1);
        clearTimer(T2);
        resetMotorEncoder(mA);
        motor[mA] = voltage;
        writeDebugStreamLine("Mdata = [0,0,%f, %f", voltage, OmegaEst);
        while(time1[T1]<=ExperimentLength)
        {
                if (time1[T2] >= SamplePeriod)
                {
                        clearTimer(T2);
                        thetaDeg = getMotorEncoder(mA);
                        OmegaEst = (thetaDeg - thetaDegPrevious)/SampleTime; // use current and last theta
                        voltage = (1/gammaM)*OmegaEst;
```

```
                    motor[mA] = voltage;
                    thetaDegPrevious = thetaDeg; // save theta for next time through
                    writeDebugStreamLine("%i, %i, %f, %f", time1(T1), thetaDeg, voltage, OmegaEst);
            }


        }
        writeDebugStreamLine("];");
        writeDebugStreamLine("OmegaDesValue = %f;",OmegaEst);
}
```

2. Graph of Ω versus t (c-code running, by hand you give it an initial spin, and hopefully it (kind of) rotates at a constant speed)



3. 1-2 sentence summary (no longer) of frictionless behavior

When we apply the strategy $V(\mathrm{t}) = \alpha\Omega_{meas}(\mathrm{t})$, the voltage applied on the motor increases when motor speed increases, which is caused by the application of an initial velocity. We could give the motor speed by hand, in which the momentum that is produced from that is able to compensate the friction momentum on the motor that would then make the motor rotate at a constant velocity.

4. Cut/paste of c-code (cleaned up) for spring-like behavior

```
#pragma config(Motor, motorA, mA, tmotorEV3_Large, openLoop, encoder)
task main()
{
        int thetaDeg, thetaDegPrevious = 0;
        float voltage,
        float OmegaEst = 0;
        int ExperimentLength = 12000;
        int SamplePeriod = 20;
        float SampleTime = SamplePeriod*0.001;
        float K1, K2;
        float gammaM = 9;
        float Ks = 4;
        K2 = 1/gammaM;
        K1 = 1/gammaM + K2;
        setMotorBrakeMode(mA, motorCoast);
        clearTimer(T1);
        clearTimer(T2);
        resetMotorEncoder(mA);
        motor[mA] = voltage;
        writeDebugStreamLine("Mdata = [0,0,%f, %f", voltage, OmegaEst);
        while(time1[T1]<=ExperimentLength)
        {
                if (time1[T2] >= SamplePeriod)
                {
                        clearTimer(T2);
                        thetaDeg = getMotorEncoder(mA);
                        OmegaEst = (thetaDeg - thetaDegPrevious)/SampleTime;
                        voltage = (1/gammaM)*OmegaEst - Ks;
                        motor[mA] = voltage;
                        thetaDegPrevious = thetaDeg;
                        writeDebugStreamLine("%i, %i, %f, %f", time1(T1), thetaDeg, voltage, OmegaEst);
                }
        }
        writeDebugStreamLine("];");
        writeDebugStreamLine("OmegaDesValue = %f;",OmegaEst);
}
```

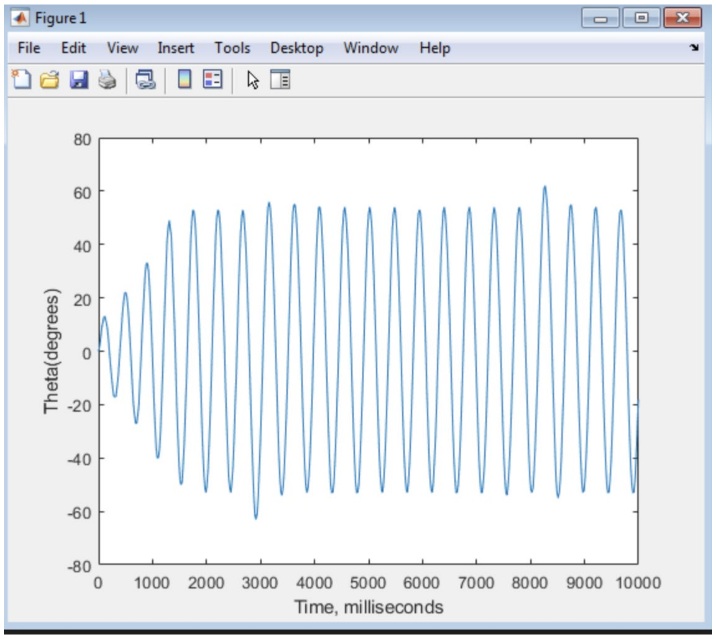
5. Estimate of oscillation frequency, based on (your estimated) motor parameters, and the "spring"-gain you chose in code

- Ks = 4

The oscillation frequency is $\frac{1}{2\pi}\sqrt{\frac{\gamma Ks}{'\Upsilon\tau}}$

- Frequency = 3. 183 Hz

6. Graph of Θ versus t (c-code running, by hand you give it an initial rotation, and hopefully it oscillates back-and-forth)



7. 1-2 sentence summary (no longer) of spring-like behavior

When we applied the spring-like strategy, the motor began to rotate back and forth like a spring, and in the plot above we see sinusoidal oscillations. The frequency of this oscillation was 3.183 Hz with a period of 0.314 seconds, which does not match the period from the plot above, and it is assumed to be caused by friction.

4 PI control of speed, no Anti-Windup logic (start of Lab 3....)

1. Choice of (ξ,ω_n)

- $\xi = 0.85$
- $\omega_n = 8$

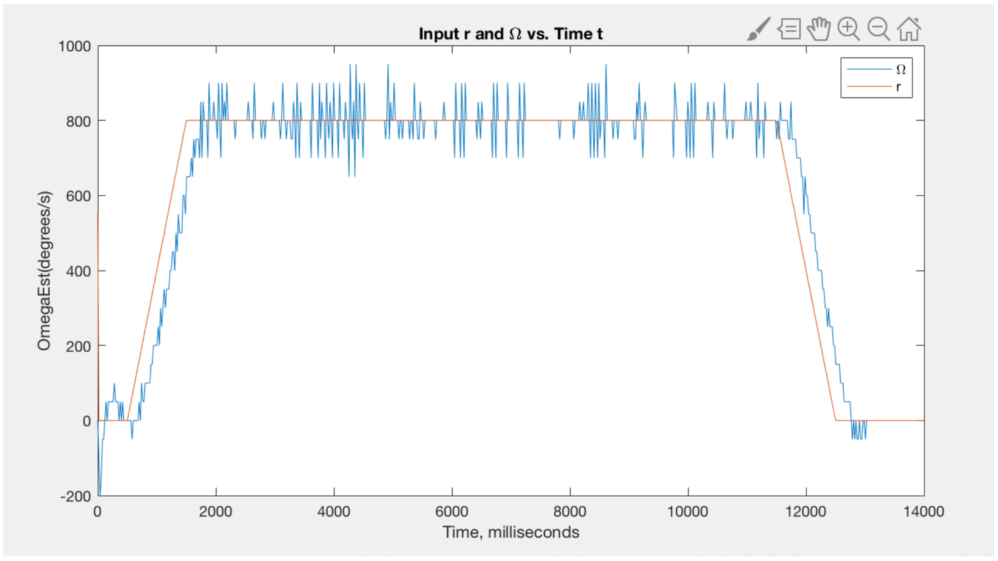
2. Cut/paste of c-code (cleaned up).

```
#pragma config(Motor, motorA, mA, tmotorEV3_Large, openLoop, encoder)
task main( )
{
        float q;
        q=0;
```

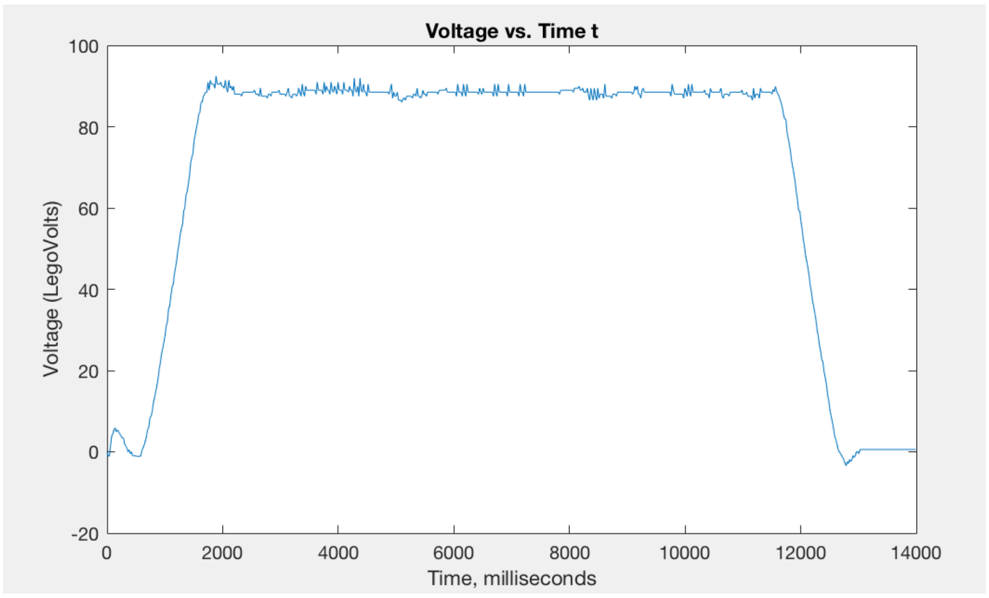
```
int T1Val;
int thetaDeg, thetaDegPrevious = 0;
float voltage, OmegaEst, OmegaDes;
int ExperimentLength = 14000; // 2.25 seconds
int SamplePeriod = 20; // 20 milliseconds = 0.02 seconds
float SampleTime = SamplePeriod*0.001; // in Seconds

float Zta = 0.85;
float OmegaN = 8;
float A = -14.92;
float B2 = 130.459;//129.493,130.46
float C = 1;

float KP, KI, KF;
KP = (2*Zta*OmegaN+A)/(B2*C);
KI = pow(OmegaN,2)/(B2*C) ;
KF = KP;
OmegaDes = 550;
setMotorBrakeMode(mA, motorCoast);
clearTimer(T1);
clearTimer(T2);
resetMotorEncoder(mA);

OmegaEst = 0;
voltage = (a*OmegaEst)-(thetaDeg*Ks)+ OmegaDes*tauM/gammaM;
motor[mA] = voltage;
//writeDebugStreamLine("Mdata = [0,0,%f", voltage);
writeDebugStreamLine("Mdata = [%i, %f, %f, %f, %f", time1(T1), OmegaEst, OmegaDes, voltage, q);
while(time1[T1]<=ExperimentLength)
{

        T1Val = time1[T1];

        if (T1Val<=500){ OmegaDes = 0;}
        else if (T1Val<=1500){OmegaDes = 800*(T1Val-500)/1000;}
        else if (T1Val<=11500){OmegaDes = 800;}
        else if (T1Val<=12500){OmegaDes = 800-800*(T1Val-11500)/1000;}
        else if (T1Val<=14000){OmegaDes = 0;}

        if (time1[T2] >= SamplePeriod)
        {
                clearTimer(T2);
                thetaDeg = getMotorEncoder(mA);
                OmegaEst = (thetaDeg - thetaDegPrevious)/SampleTime;
                voltage = KI*q + KF*OmegaDes - KP*OmegaEst;
                motor[mA] = voltage;
                thetaDegPrevious = thetaDeg;

                q=q + SampleTime*(OmegaDes-OmegaEst);
                writeDebugStreamLine("%i, %f, %f, %f, %f", time1(T1), OmegaEst, OmegaDes, voltage, q);
        }
        }
        writeDebugStreamLine("];");
        writeDebugStreamLine("OmegaDesValue = %f;",OmegaDes);
}
```

3. one interesting reference trajectory that is gentle enough so that control V stays within ±100 limits

(a) Graph of r and Ω versus t



(b) separate graph of V versus t

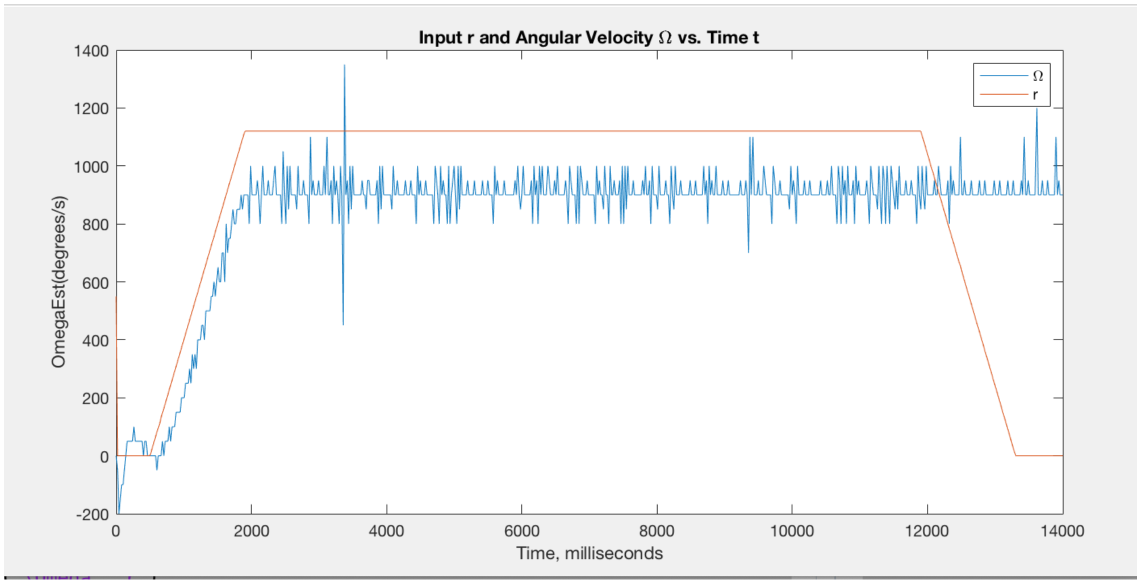


4. 1-2 sentence summary (no longer)

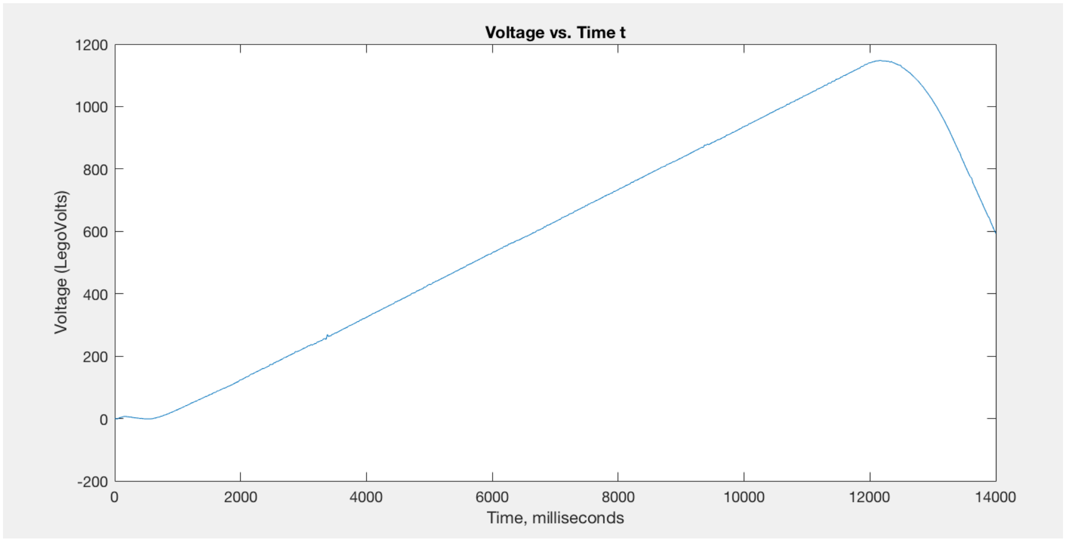
The PI controller stabilizes the system and maintains a steady-state gain from reference, r to the output, Ω. Anti-windup logic is not necessary as long as the reference is not causing control output to exceed the actuator's capacity limit.

5. reference trajectory that does result in bad integral windup.

(a) One graph of r and Ω versus t



(b) separate graph of V versus t. Windup of controller integrator will be evident in both of these graphs.



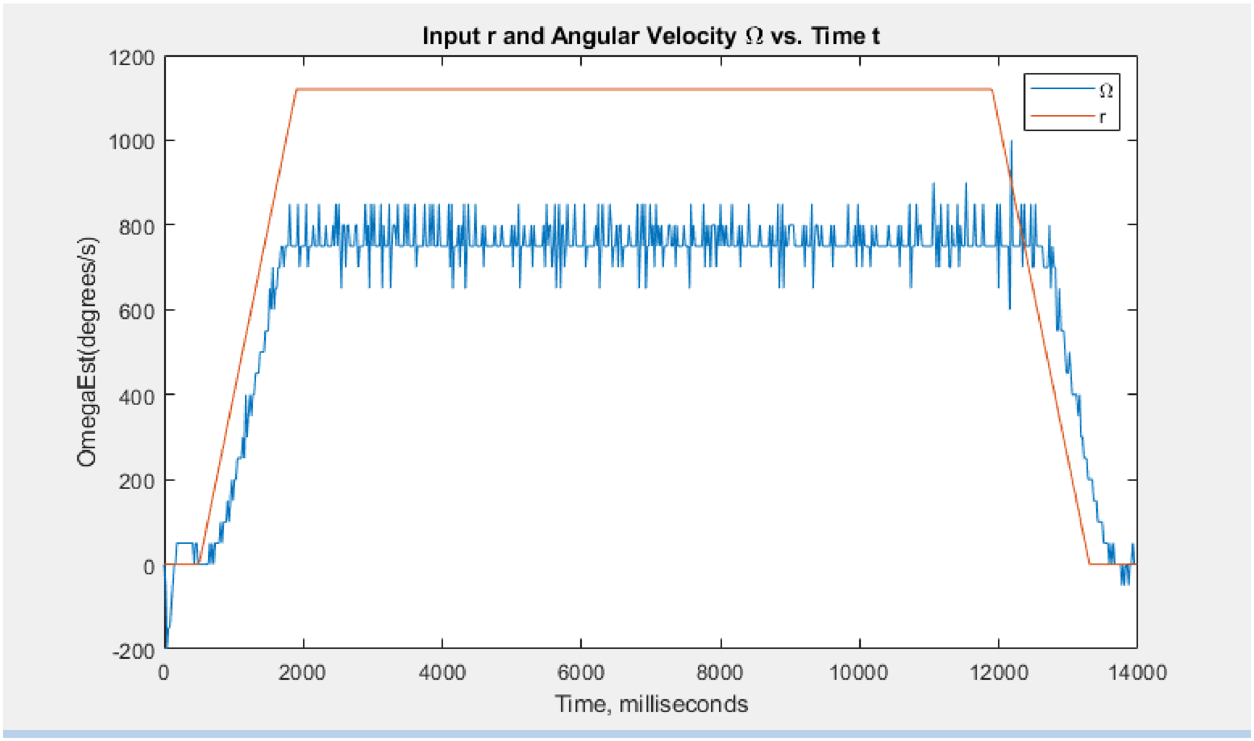
6. 1-2 sentence summary (no longer)

If the reference is causing control input to exceed the capacity limit of the actuator, the output cannot track the reference promptly when the anti-windup logic is not present. In this case, the controller is unable to switch off the integration when saturated.

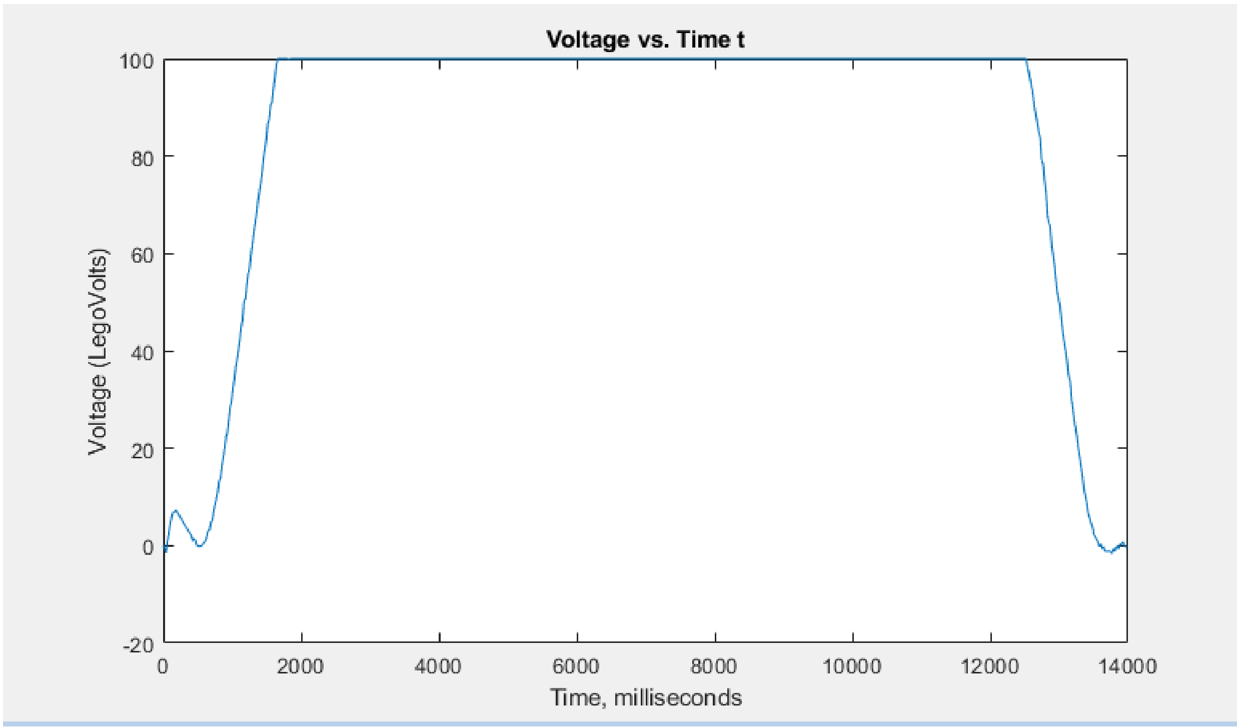
5 PI control of speed with Anti-Windup logic (...continuing Lab 3)

1. Closed-loop response to reference input that caused problem in previous section.

(a) One graph of r and Ω versus t



(b) separate graph of V versus t.



2. Cut/paste of c-code (cleaned up).

```
#pragma config(Motor, motorA, mA, tmotorEV3_Large, openLoop, encoder)
task main()
{
        float q;
        q=0;

        int T1Val;
        int thetaDeg, thetaDegPrevious = 0;
        float voltage, OmegaEst, OmegaDes;
        int ExperimentLength = 14000;
        int SamplePeriod = 20;
        float SampleTime = SamplePeriod*0.001;
        float gammaM = 8.7408;
        float a=1/gammaM;
        float tauM = 0.067;
        float Ks=2.4;

        float Zta = 0.85;
        float OmegaN = 8;
        float A = -14.92;
        float B2 = 130.459;
        float C = 1;

        float KP, KI, KF;
        KP = (2*Zta*OmegaN+A)/(B2*C);
        KI = pow(OmegaN,2)/(B2*C) ;
        KF = KP;
        OmegaDes = 550;
        setMotorBrakeMode(mA, motorCoast);
        clearTimer(T1);
        clearTimer(T2);
        resetMotorEncoder(mA);
        OmegaEst = 0;
        voltage = (a*OmegaEst)-(thetaDeg*Ks)+ OmegaDes*tauM/gammaM;
        motor[mA] = voltage;
        writeDebugStreamLine("Mdata = [0,0,%f", voltage);

        float uMax = 100;
        float uMin = -100;

        float errorOmega, qdot;

        writeDebugStreamLine("Mdata = [%i, %f, %f, %f, %f", time1(T1), OmegaEst, OmegaDes, voltage, q);
        while(time1[T1]<=ExperimentLength)
        {

                //code below goes in the WHILE loop
```

```
		T1Val = time1[T1];

		if (T1Val<=500){ OmegaDes = 0;}
		else if (T1Val<=1900){OmegaDes = (800*(T1Val-500)*0.001);}
		else if (T1Val<=11900){OmegaDes = 1120;}
		else if (T1Val<=13300){OmegaDes = (1120-800*(T1Val-11900)*0.001);}
		else if (T1Val<=15000){OmegaDes = 0;}

		if (time1[T2] >= SamplePeriod)
		{
			clearTimer(T2);
			thetaDeg = getMotorEncoder(mA);
			OmegaEst = (thetaDeg - thetaDegPrevious)/SampleTime;

			voltage = KI*q + KF*OmegaDes - KP*OmegaEst;

			if (voltage > uMax)
			{voltage = uMax;}
			else if (voltage < uMin)
			{voltage = uMin;}

		motor[mA] = voltage;

		errorOmega = OmegaDes - OmegaEst;

		if (voltage >= uMax && KI*errorOmega >= 0) {
			qdot = 0.0;
			} else if (voltage <= uMin && KI*errorOmega <= 0) {
			qdot = 0.0;
			} else {
			qdot = errorOmega;
		}

		thetaDegPrevious = thetaDeg; // save theta for next time through

		q = q + SampleTime*qdot;
		writeDebugStreamLine("%i, %f, %f, %f, %f", time1(T1), OmegaEst, OmegaDes, voltage, q);
	}
}
writeDebugStreamLine("];");

}
```

3. 1-2 sentence summary (no longer)

With the present of anti-windup logic, the controller is able to switch off the integration when saturated; therefore, the output cannot reach an over-demanded magnitude, but it can track the reference and make an adjustment so that the reference drops below the saturation line.

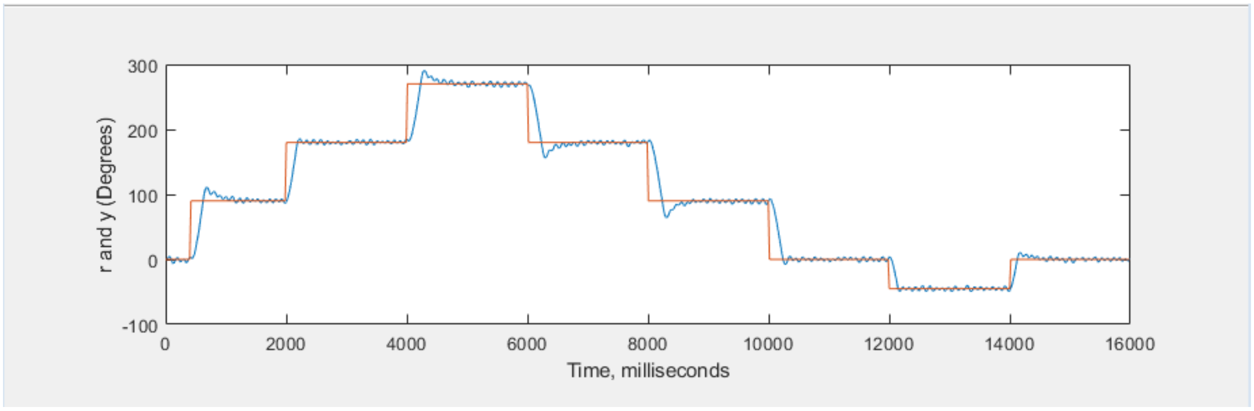
6 PI control of position with State Feedback and Anti-Windup logic (All of Lab 4)

1. Choice of (ξ,ω_n,α).

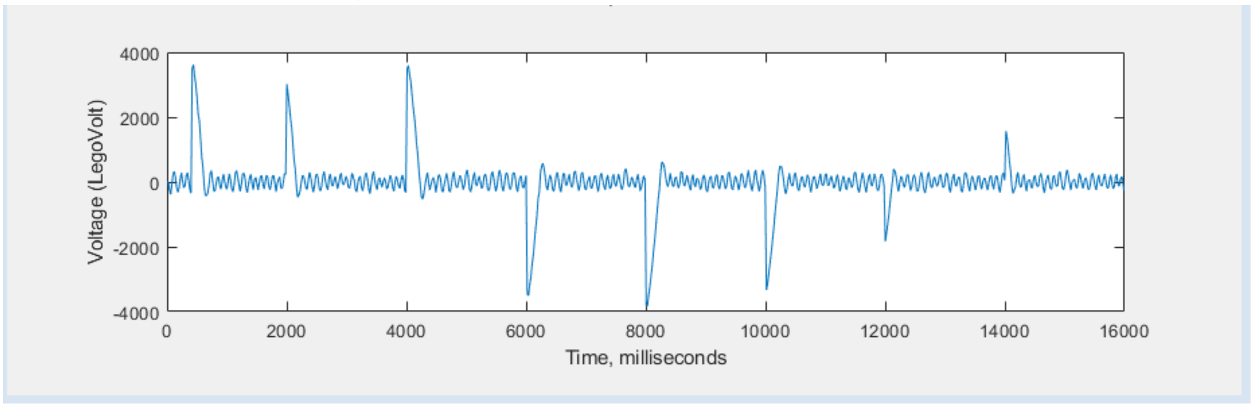
- $\xi = 1.3$
- $\omega_n = 30$
- $\alpha = 1.5$

2. One interesting reference trajectory

(a) graph of r and θ versus t



(b) separate graph of V versus t.



3. Cut/paste of c-code (cleaned up).

```
#pragma config(Motor, motorA, mA, tmotorEV3_Large, openLoop, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
task main()
{
        float gammaM = 8.7408;
        float tauM = 67;
        float zta = 1.3;
        float wn = 30;
        float alpha = 1.5;

        float K1 = (1-tauM*(2*alpha*zta+1)*wn*wn)/gammaM;
        float K2 = -(tauM*(2*zta+alpha)*wn)/gammaM;
        float KI = tauM*alpha*wn*wn*wn/gammaM;
        float KF = -K1;
float q, qdot, uVal, e = 0;
float thetaDes, thetaDeg, thetaDegPrevious, OmegaEst = 0;

float tWidth = 400;
int uMax = 100;
int uMin = -100;

int ExperimentLength = 40*tWidth;
float SamplePeriod = 20;
float SampleTime = SamplePeriod*0.001;
float T1Val;

setMotorBrakeMode(mA, motorCoast);
clearTimer(T1);
clearTimer(T2);
resetMotorEncoder(mA);

writeDebugStreamLine("Mdata = [0,0,0,0,0;");

while(time1[T1]<=ExperimentLength){

        T1Val = time1[T1];
        if (T1Val <= 1*tWidth){thetaDes = 0;}
        else if (T1Val <= 5*tWidth){thetaDes = 90;}
        else if (T1Val <= 10*tWidth){thetaDes = 180;}
        else if (T1Val <= 15*tWidth){thetaDes = 270;}
        else if (T1Val <= 20*tWidth){thetaDes = 180;}
        else if (T1Val <= 25*tWidth){thetaDes = 90;}
        else if (T1Val <= 30*tWidth){thetaDes = 0;}
        else if (T1Val <= 35*tWidth){thetaDes = -45;}
        else {thetaDes = 0;}

        if (time1[T2] >= SamplePeriod)
              {
                clearTimer(T2);
                thetaDeg = getMotorEncoder(mA);
                e = thetaDes - thetaDeg;
                OmegaEst = (thetaDeg-thetaDegPrevious)/SampleTime;

                if (uVal>=uMax && KI*e>=0)
                     {
                        qdot = 0.0;
                        }
                  else if (uVal<=uMin && KI*e<=0)
                     {
                        qdot = 0.0;
                        }
                  else
                     {
                        qdot = e;
```

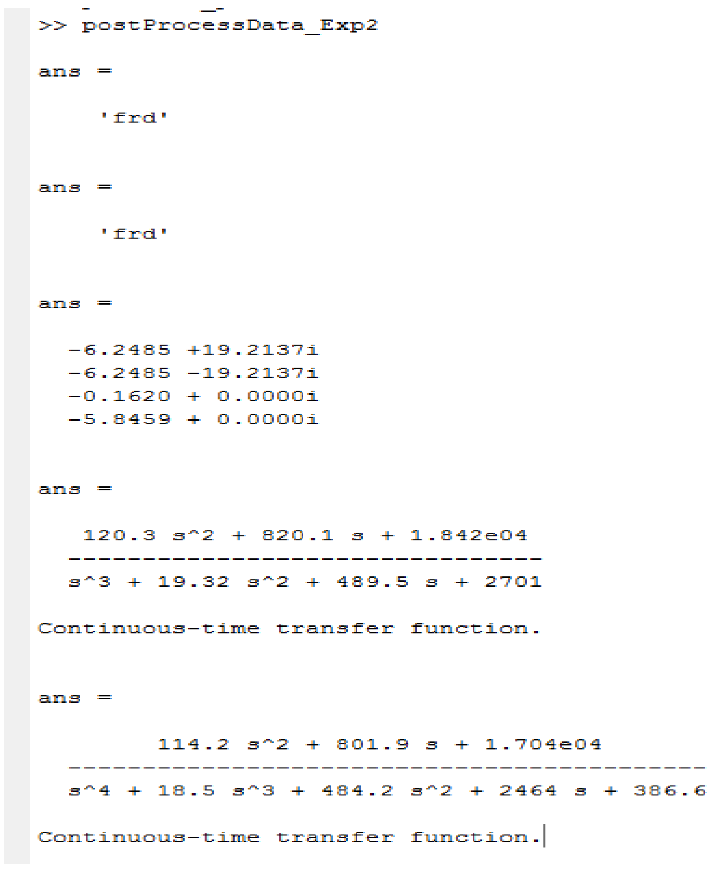
```
                    }
                    q = q + SampleTime*qdot;

                    uVal = 0.001*(KI*q + K1*thetaDeg + K2*OmegaEst + KF*thetaDes);
                    motor[mA] = uVal;

                    writeDebugStreamLine("%i, %f, %i, %f, %f;", time1(T1), thetaDeg ,          thetaDes, uVal,q);
                    thetaDegPrevious = thetaDeg;
            }
        }

        writeDebugStreamLine("];");
        writeDebugStreamLine("zta = %f;", zta);
        writeDebugStreamLine("wn = %f;", wn);
        writeDebugStreamLine("alpha = %f;", alpha);
}
```

4. 1-2 sentence summary (no longer)

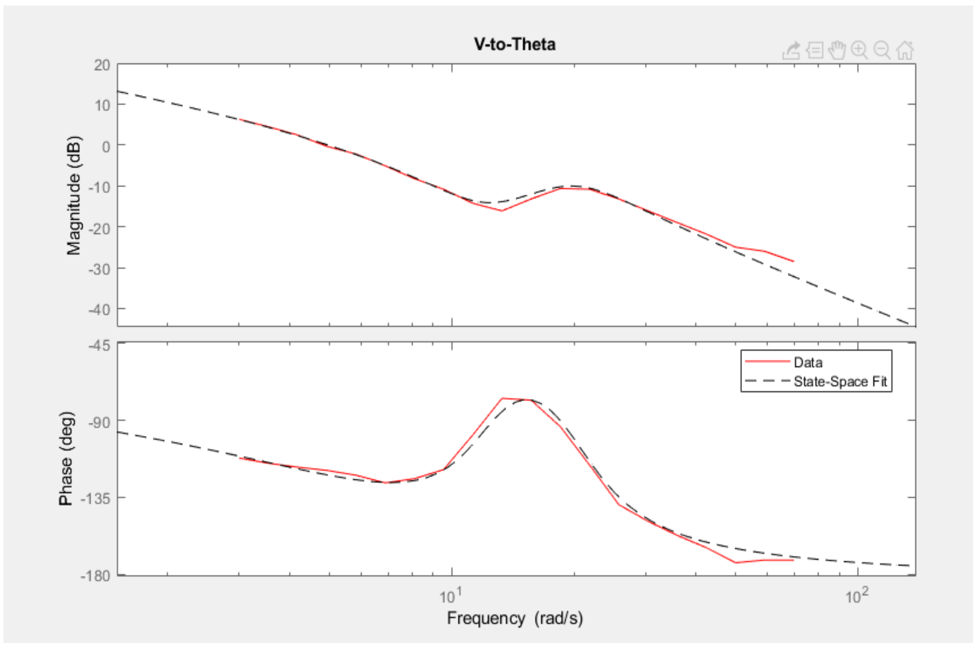
Our choices of parameters caused an underdamped output which converges in an exponential way compared to the reference and where we can only see one period of the sinusoid. Our overshoot is relatively small compared to the case with a larger value of natural frequency.

7 Frequency-response Identification of Motor (optional) (All of Lab 5)

1. **Estimated frequency-response plots, and fitted transfer functions from Matlab post-processing for both systems (file does all this for you...)**

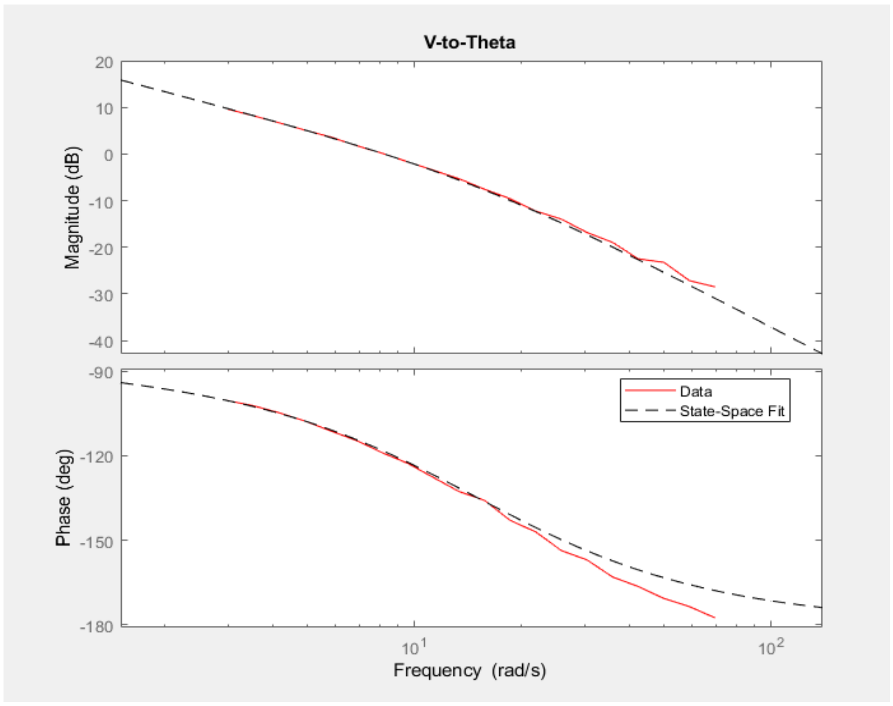


```
>> postProcessData_Exp2

ans =

    'frd'


ans =

    'frd'


ans =

  -6.2485 +19.2137i
  -6.2485 -19.2137i
  -0.1620 + 0.0000i
  -5.8459 + 0.0000i


ans =

   120.3 s^2 + 820.1 s + 1.842e04
  --------------------------------
  s^3 + 19.32 s^2 + 489.5 s + 2701

Continuous-time transfer function.


ans =

        114.2 s^2 + 801.9 s + 1.704e04
  -------------------------------------------
  s^4 + 18.5 s^3 + 484.2 s^2 + 2464 s + 386.6

Continuous-time transfer function.
```



2. **For isolated motor, comparison to the results from section 1 above.**

```
>> postProcessData

ans =

    'frd'


ans =

    'frd'


ans =

   -0.0422
  -14.9742


ans =

    144.1
  ---------
  s + 15.28

Continuous-time transfer function.


ans =

           140
  ----------------------
  s^2 + 15.02 s + 0.6325

Continuous-time transfer function.
```



Comment:

The magnitude and phase shift of angle and angular velocity response increases abruptly at approximately 20 rad/s when additional inertia is involved. This might result from the change of natural frequency due to the change of system configurations.

The frequency domain analysis of isolated motor case shows that if the input frequency increases, both the response magnitude for the angle and the angular velocity decrease, despite a few irregularities at the end of measurement.