California State University, Long Beach

Department of Electrical Engineering

EE 471 Sec 01 7731 Design of Control Systems

Final Lab Project: Design of Control Systems to Swing Up and Stabilize an Inverted Pendulum With a Reaction Wheel End Effector

Team Members

Katie Choi	016446948
Hien Nguyen	016271110
Grayson Galisky	016620693

Faculty Advisor

Professor Hossein Jula

December 12, 2020

Table of contents

Preface

Antun, creator of the simpleFOC project, was very helpful in the creation of this project. The CAD files and code in our project are modified versions of the ones found on the simpleFOC page*. Our modified CAD files and code can be downloaded from this link (https://ggalisky.weebly.com/control-systems-projects.html) or you can request a copy from graysongalisky@gmail.com if the aforementioned link is no longer working. These modifications are to the mounting holes to allow the system to fit our specific brushless motor and other parts. Our swing up controller is also heavily inspired/ derived from the simpleFOC swing up controller*. Our lab team greatly appreciates the availability of Antun to answer questions via email about his project and different control methods he has explored. A video presentation of the information in this project report can be found at this link: . If the video link is no longer working please email the previously mentioned email address for a copy of the video presentation

Introduction

In this project we will be designing and building a control system to Swing Up and Stabilize an Inverted Pendulum With a Reaction Wheel End Effector (IPWRW). We will first create a theoretical physics based model of an IPWRW. Then we will use our model to design a IPWRW hardware platform and control systems to drive swing up and stabilization. We will be approaching the problem of stabilization by using both an LQR and PID controller. The swing up controller will remain the same for both LQR and PID controllers.

Methods

List of equipment used

- Arduino IDE
- MATLAB R2019B
- SIMULINK
- Pycharm IDE
- Agilent 6654A 0-60VDC 0-9A Variable Lab Power Supply
- Rigol DS1054 Osilicsope
- Protek 608 multimeter
- Hakko Soldering iron
- SC-2kg pocket scale
- Various test leads
- Ultimaker 2 3D printer
- Ender 3 Pro 3D printer
- ABS 3D printing filament
- PLA 3D printing filament
- 3D printed parts*
- simpleFOC motor driver shield
- 2 x 8mm x 16mm x 5mm radial bearings
- 1 x Gimbal Brush motor Part number:: BGM4108-150HS
- 1 xArduino Uno
- 2 x CUI AMT103-V quadrature encoders
- 1 x 8mm OD 6mm x 30mm stainless steel tubing
- Various M3, M4, and M5 screws

Total project cost (not including test equipment and software): $170 USD

Physics modeling

In order to develop our control system we first need to understand the key physics parameters of our system. Figure x gives a visual that highlights key parameters of our system. Table 1 outlines all of the parameters we will be using to build our state space equations.

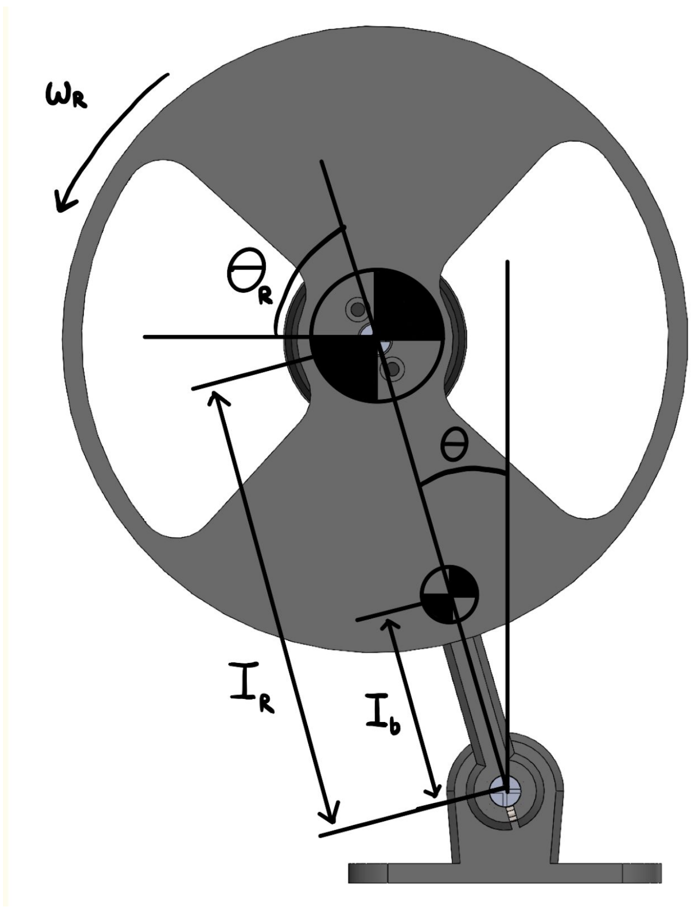


Figure 1 - Labeled 2D view of a IPWRW

Pendulum Parameters

- Mp - full pendulum mass
- Mr - mass of reaction wheel and motor
- Lr - distance from the base axis to the center of mass of the pendulum
- g - gravitational acceleration
- Iw - wheel moment of inertia
- Ip - full pendulum moment of inertia

Motor parameters

- Tm - motor torque constant
- Temf - motor back EMF constant
- Rp - motor phase resistance

Experimental and theoretical measurements of key system parameters

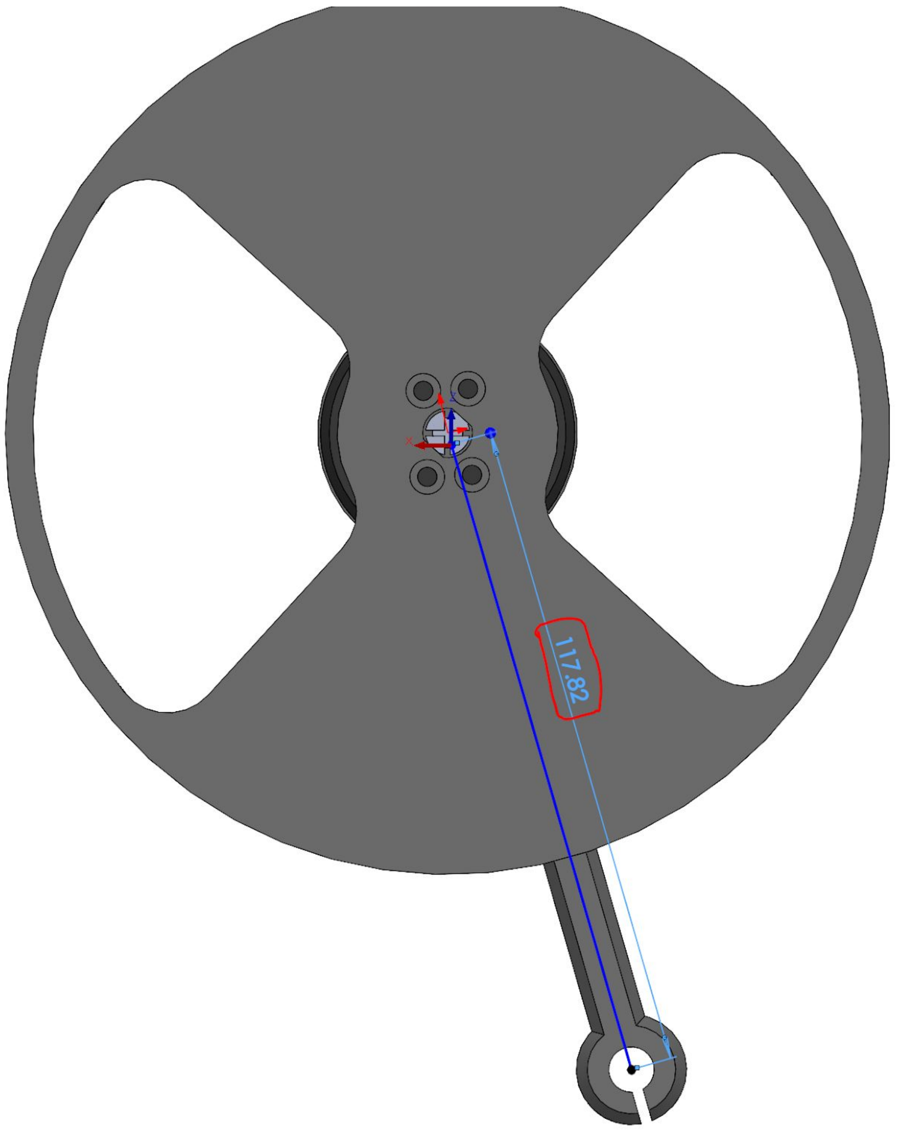
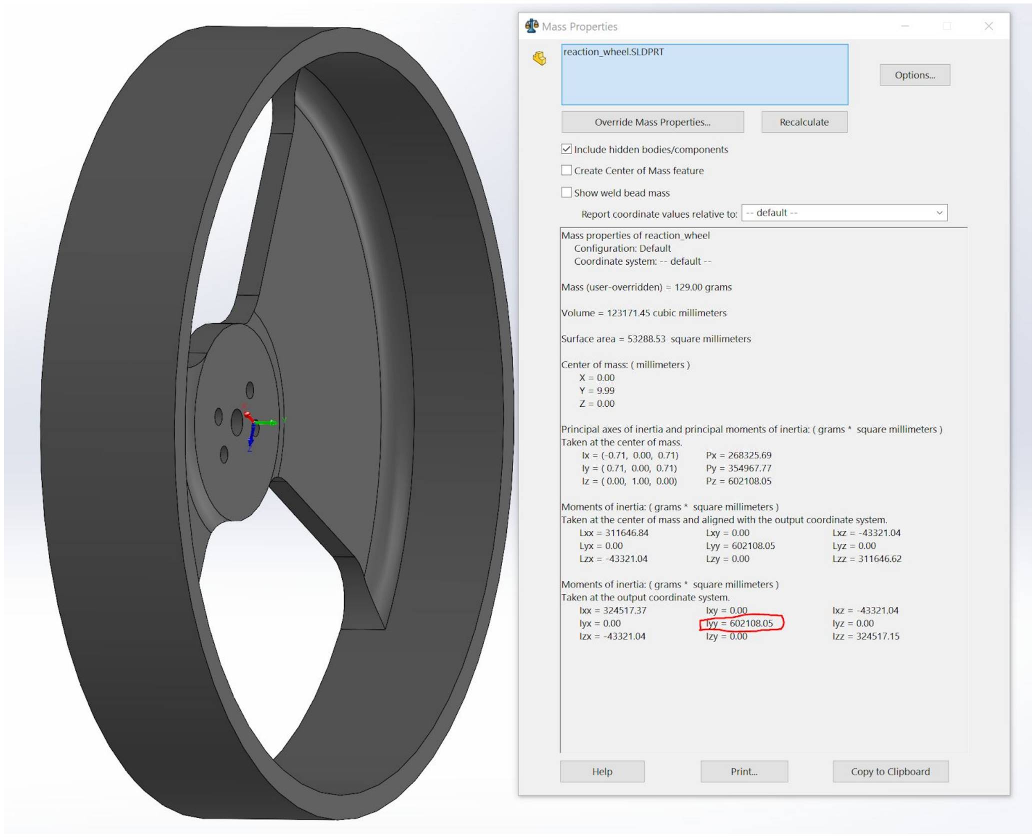
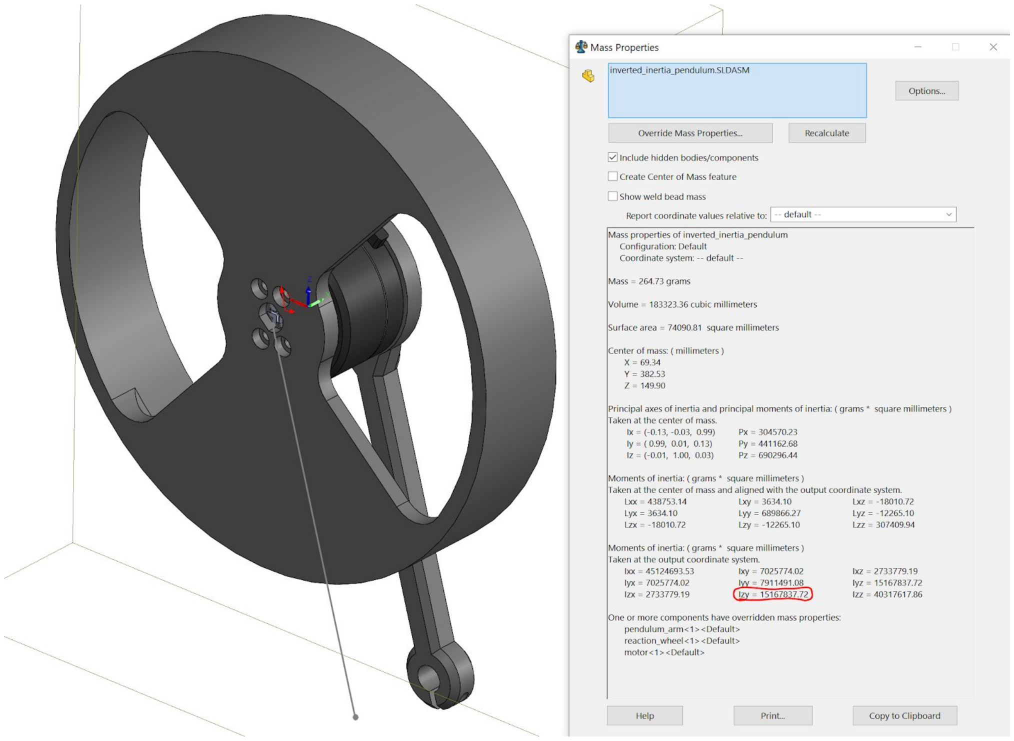


Figure 2 - Distance from base to center of mass in mm



```
[✓] Include hidden bodies/components
[ ] Create Center of Mass feature
[ ] Show weld bead mass
    Report coordinate values relative to: -- default --

Mass properties of reaction_wheel
    Configuration: Default
    Coordinate system: -- default --

Mass (user-overridden) = 129.00 grams

Volume = 123171.45 cubic millimeters

Surface area = 53288.53  square millimeters

Center of mass: ( millimeters )
     X = 0.00
     Y = 9.99
     Z = 0.00

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
      Ix = (-0.71,  0.00,  0.71)       Px = 268325.69
      Iy = ( 0.71,  0.00,  0.71)       Py = 354967.77
      Iz = ( 0.00,  1.00,  0.00)       Pz = 602108.05

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 311646.84            Lxy = 0.00                Lxz = -43321.04
     Lyx = 0.00                 Lyy = 602108.05           Lyz = 0.00
     Lzx = -43321.04            Lzy = 0.00                Lzz = 311646.62

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
     Ixx = 324517.37            Ixy = 0.00                Ixz = -43321.04
     Iyx = 0.00                 Iyy = 602108.05           Iyz = 0.00
     Izx = -43321.04            Izy = 0.00                Izz = 324517.15
```

Figure 3 - Moment of inertia of reaction wheel



```
[✓] Include hidden bodies/components
[ ] Create Center of Mass feature
[ ] Show weld bead mass
    Report coordinate values relative to: -- default --

Mass properties of inverted_inertia_pendulum
    Configuration: Default
    Coordinate system: -- default --

Mass = 264.73 grams

Volume = 183323.36 cubic millimeters

Surface area = 74090.81  square millimeters

Center of mass: ( millimeters )
     X = 69.34
     Y = 382.53
     Z = 149.90

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
      Ix = (-0.13, -0.03,  0.99)       Px = 304570.23
      Iy = ( 0.99,  0.01,  0.13)       Py = 441162.68
      Iz = (-0.01,  1.00,  0.03)       Pz = 690296.44

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 438753.14            Lxy = 3634.10             Lxz = -18010.72
     Lyx = 3634.10              Lyy = 689866.27           Lyz = -12265.10
     Lzx = -18010.72            Lzy = -12265.10           Lzz = 307409.94

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
     Ixx = 45124693.53          Ixy = 7025774.02          Ixz = 2733779.19
     Iyx = 7025774.02           Iyy = 7911491.08          Iyz = 15167837.72
     Izx = 2733779.19           Izy = 15167837.72         Izz = 40317617.86

One or more components have overridden mass properties:
     pendulum_arm<1><Default>
     reaction_wheel<1><Default>
     motor<1><Default>
```

Figure 4 - Moment of Inertia of Full Pendulum

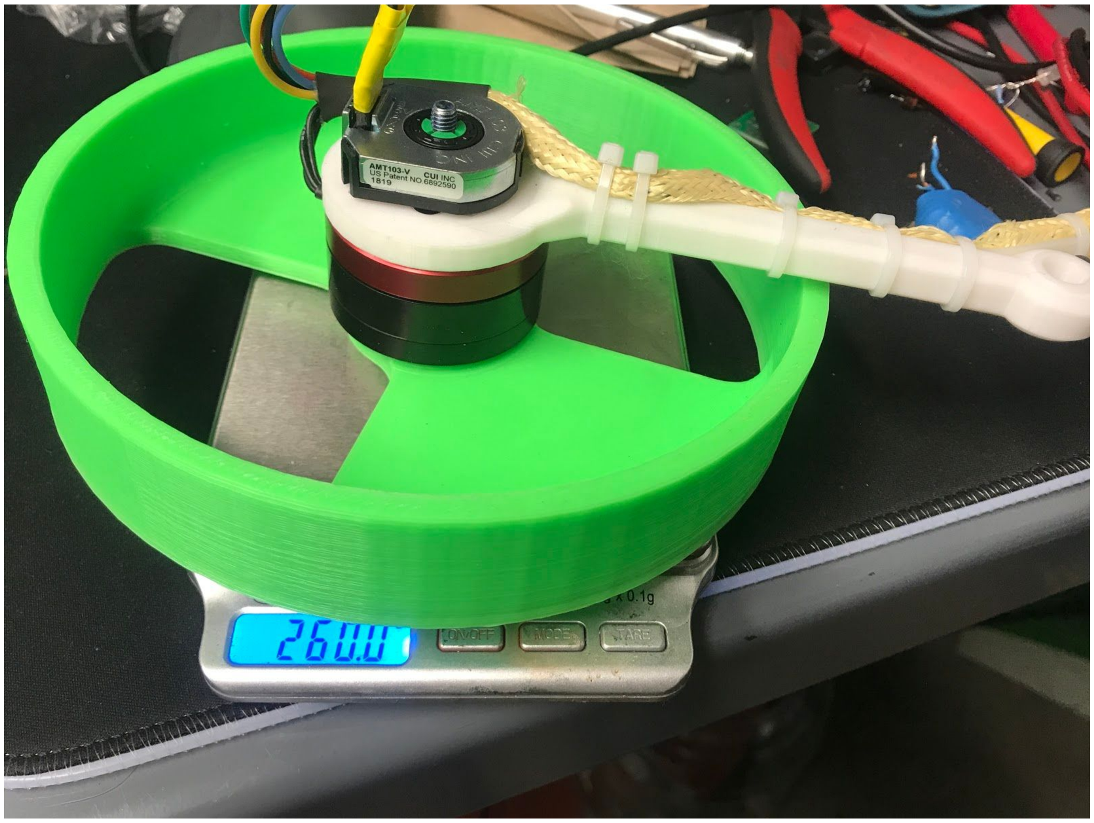


Figure 5 - Mass Measurement of Pendulum, Reaction Wheel, Encoder, and Hardware

Parameters	**Measurement Method**	**Value**	**Units**
Mp - full pendulum mass	Weighing scale	0.273	kg
Mr - mass of reaction wheel and motor	Weighing scale	0.260	kg
Lr - distance from the base axis to the center of mass of the pendulum	Solidworks measurement tool	0.11782	meters
Iw - wheel moment of inertia	Solidworks measurement tool	6.02108×10^{-4}	grams*mm^2

Ip - full pendulum moment of inertia	Solidworks mass measurement tool	0.0053	grams*mm^2
Tm - motor torque constant	Data sheet of motor	0.25	N*m/A
EMF - motor back EMF constant	Data sheet of motor	8/325	V
Rp - motor phase resistance	Ohm meter	9.8	Ohms
g - gravitational coefficient	Standard value	9.81	m/s^2

Table 1 - Measured System Parameters

State Space Modeling

Recall that:

$$\frac{d}{dt}\theta = \dot{\theta} = \omega$$

Figure 6 - Angular velocity

$$\dot{\omega} = \frac{g}{L_r}\theta + \frac{T_{emf}T_m}{I_p R_p}v - \frac{T_m}{I_p R_p}V$$

Figure 7 - Angular acceleration of the pendulum

$$\dot{v} = -\frac{g}{L_r}\theta - \frac{T_{emf}T_m}{I_w R_p}v + \frac{T_m}{I_w R_p}V$$

Figure 8 - Motor acceleration

$$\dot{x} = Ax + Bu$$
$$y = Cx + D$$

Figure 9 - General SS equation

$$u(t) = V$$

Figure 10 - Input vector

$$x(t) = \begin{bmatrix} \theta \\ \omega \\ v \end{bmatrix}$$

$$\dot{x}(t) = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{v} \end{bmatrix}$$

Figure 11 - State vector

$$y(t) = \theta$$

Figure 12 - Output equation

$$A = \begin{bmatrix} 0 & 1 & 0 \\ \frac{g}{L_r} & 0 & \frac{T_{emf}T_m}{I_pR_p} \\ -\frac{g}{L_r} & 0 & -\frac{T_{emf}T_m}{I_wR_p} \end{bmatrix}$$

Figure 13 - State matrix

$$B = \begin{bmatrix} 0 \\ -\frac{T_{emf}}{J_pRp} \\ \frac{T_{emf}}{J_wRp} \end{bmatrix}$$

Figure 14 - Input Matrix

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Figure 15 - Output Matrix

$$D = \begin{bmatrix} 0 \end{bmatrix}$$

Figure 16 - Feedforward Matrix

LQR Modeling In Matlab

Pendulum, Motor, Sampling Time, and Power Supply Parameter Configuration

```
Mp = 0.273;                         % full pendulum mass
Mr = 0.260;                         % motor+wheel
Lr  = 0.11782;                      % center of mass distance
Iw = 602108.05e-9;                  % Wheel moment of inertia
g = 9.81;                           % gravity acceleration
Ip = Mp*Lr^2 + 1516783.77e-9;       % full pendulum moment of inertia
Tm = .25;                           % motor torque constant
Temf = 8/325;                       % motor back EMF constant
Rp = 10.2;                          % motor armature resistance
Ts = 25e-3;                         % controller sampling time
max_v = 18;                         % Maximum power supply voltage
```

State Space Modeling and LQR Controller

```
A = [0,1,0;g/Lr,0,Temf*Tm/(Ip*Rp);-g/Lr,0,Tm*Temf/(Iw*Rp)];%State Matrix
B = [0;-Tm/(Ip*Rp);Tm/(Iw*Rp)];                            %Input Matrix
C = [1,0,0];                                               %Output Matrix
D = [0];                                                   %Feedforward Matrix
G = ss(A,B,C,D);     %generate state space equation
T = ss2tf(A,B,C,D);  %Transfer Function from state space equation
Gd = c2d(G,Ts);
Qd = diag([1,1,1]);
Rd = 500;
[K, P] = dlqr(Gd.a,Gd.b,Qd,Rd);
K =-K;
disp(strcat('Linearized LQR Controller Generated With dlqr '))
disp(strcat('Target Voltage = ',num2str(K(1)),'*pendulum angle +
',num2str(K(2)),'*pendulum velocity + ',num2str(K(3)),'*motor velocity'))
```

OUTPUT:

```
Linearized LQR Controller Generated With dlqr
Target Voltage =48.9411*pendulum angle +5.4781*pendulum velocity +0.16154*motor velocity
```

Simulink Model

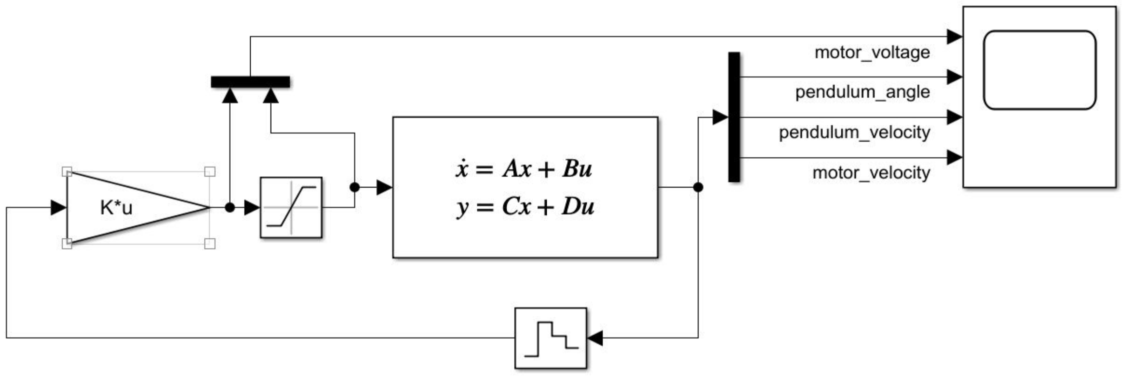


Figure 17 - Simulink Block Diagram

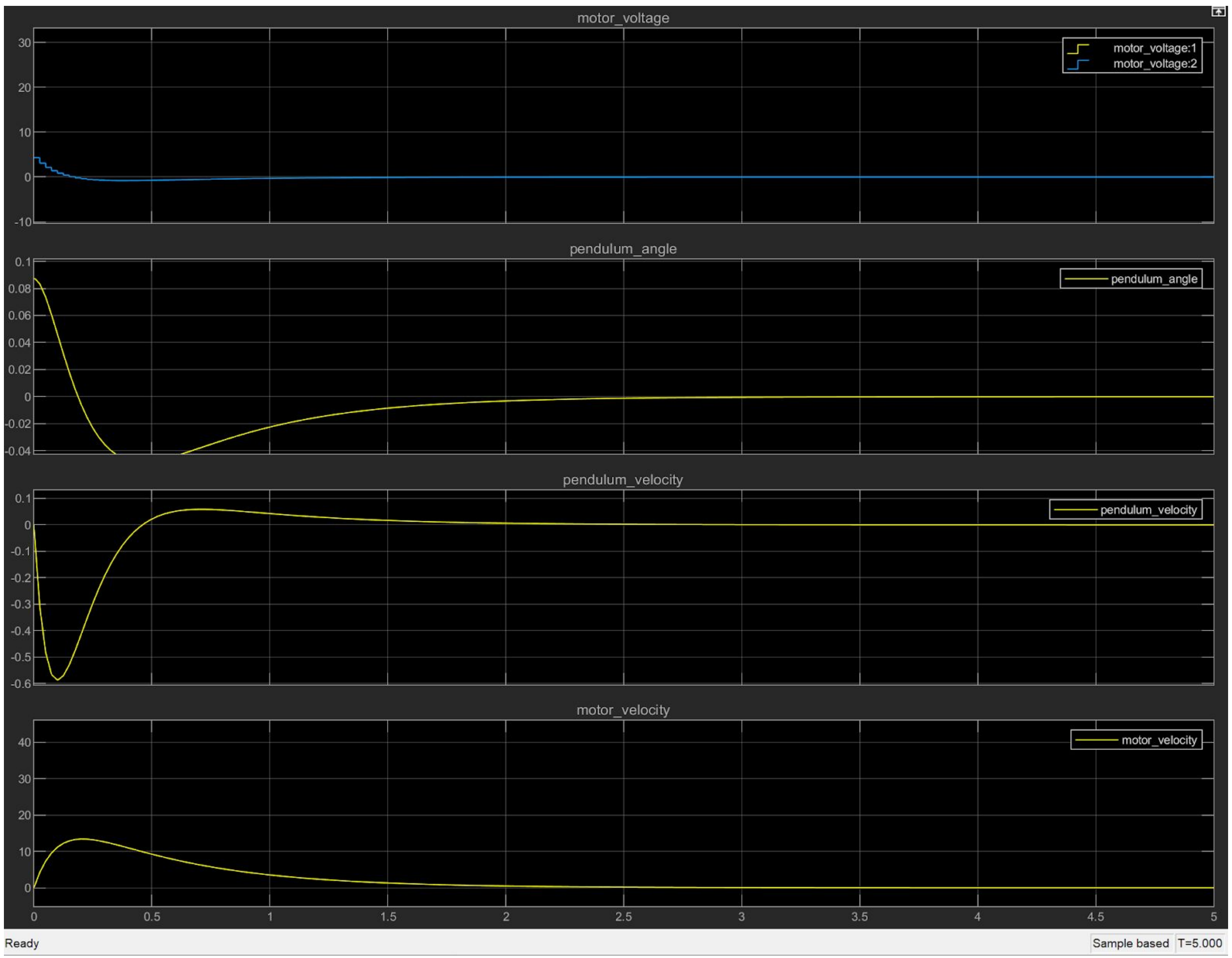
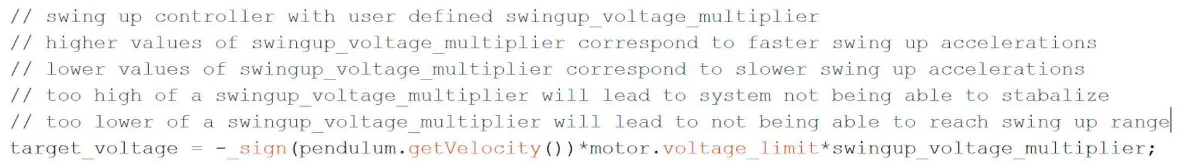


Figure 18 - Simulink Model Output

Swing up controller

As noted in the preface, our swing up controller is nearly identical to the swing up controller featured in the simpleFOC project*. The swing up controller checks the velocity value of the pendulum and then sends voltage to the motor to accelerate the motor in the opposite direction of the pendulum velocity. This results in the pendulum swinging back and forth higher and higher until it reaches a stabilization range. Once it reaches the stabilization range the swing up controller switches to either our LQR or PID controller depending on the experiment we are running. The variable in our code "swingup_voltage_multiplier" specifies how much voltage the swing up controller will apply when swinging up. If this value is too high, the pendulum will swing past the stabilization range too fast for the stabilization controller to compensate. If the value is too low, the pendulum will not be able to swing up. The code for our swing up controller can be seen in figure x below and in figure XX which shows the swing up controller in the full context of the rest of the firmware



```
// swing up controller with user defined swingup_voltage_multiplier
// higher values of swingup_voltage_multiplier correspond to faster swing up accelerations
// lower values of swingup_voltage_multiplier correspond to slower swing up accelerations
// too high of a swingup_voltage_multiplier will lead to system not being able to stabalize
// too lower of a swingup_voltage_multiplier will lead to not being able to reach swing up range
target_voltage = -_sign(pendulum.getVelocity())*motor.voltage_limit*swingup_voltage_multiplier;
```

Figure 19 - Swing up controller code

Arduino Firmware for LQR Controller

NOTE: The firmware shown here is derived from the simpleFOC firmware found on github. We have made modifications to the example code given to get our system working. These modifications include reassigning pins, adding the signum function, modifying the swing up controller, and modifying the LQR controller to better suit our specific hardware setup. A copy of this code file can be found at the link provided in the preface.

```
// simpleFOC library, check out github for more information on its functions
#include <SimpleFOC.h>
```

```
// software interrupt library
#include <PciManager.h>
#include <PciListenerImp.h>
const double swingup_voltage_multiplier = .35;
// init of BLDC motor NOTE: make sure to find your motor pol pair value prior to running this code
// your pole pair number will go in place of the "11" below
BLDCMotor motor = BLDCMotor(11);
// driver instance. Parameters passed through are pin numbers of the arduino uno that are PWM capable
BLDCDriver3PWM driver = BLDCDriver3PWM(9, 10, 6, 8);
// init for motor encoder. First two parameters are arduino pins for channel A and B. Last parameter is encoder resolution.
// we are using the CUI AMT 103V which has a user specifiable quadrature resolution. Ours is set to the default value of 2048
Encoder encoder = Encoder(2, 3, 2048);
// interrupt routine
void doA(){encoder.handleA();}
void doB(){encoder.handleB();}
// init for pendulum encoder
Encoder pendulum = Encoder(A1, A2, 2048);
// interrupt routine
void doPA(){pendulum.handleA();}
void doPB(){pendulum.handleB();}
// PCI manager interrupt
PciListenerImp listenerPA(pendulum.pinA, doPA);
PciListenerImp listenerPB(pendulum.pinB, doPB);
// defining signnum function used later
#define sign(x) ((x) < 0 ? -1 : ((x) > 0 ? 1 : 0))

void setup() {
  // setting up serial comm for passing back information for future analysis
  Serial.begin(115200);
  // initialise motor encoder hardware
  encoder.init();
  encoder.enableInterrupts(doA,doB);
  // init the pendulum encoder
  pendulum.init();
  PciManager.registerListener(&listenerPA);
  PciManager.registerListener(&listenerPB);
  // set control loop type to be used
  motor.controller = ControlType::voltage;
  // link the motor to the encoder
  motor.linkSensor(&encoder);
  // the value below should not exceed 24V.
  driver.voltage_power_supply = 18;
  driver.init();
  // link the driver and the motor
  motor.linkDriver(&driver);

  // initialize motor
  motor.init();
  // align encoder and start FOC
  motor.initFOC();
}

// loop downsampling counter
long loop_count = 0;

void loop() {
  // ~1ms
  motor.loopFOC();

  // control loop each ~25ms - sampling time
  if(loop_count++ > 25){

    // calculate the pendulum angle
    float pendulum_angle = constrainAngle(pendulum.getAngle() + M_PI);

    float target_voltage;
    if( abs(pendulum_angle) < 0.5 ) // if angle small enough stabilize
      target_voltage = controllerLQR(pendulum_angle, pendulum.getVelocity(), motor.shaftVelocity());
    else // else do swing-up
      // swing up controller with user defined swingup_voltage_multiplier
      // higher values of swingup_voltage_multiplier correspond to faster swing up accelerations
      // lower values of swingup_voltage_multiplier correspond to slower swing up accelerations
      // too high of a swingup_voltage_multiplier will lead to system not being able to stabilize
      // too lower of a swingup_voltage_multiplier will lead to not being able to reach swing up range
      target_voltage = -_sign(pendulum.getVelocity())*motor.voltage_limit*swingup_voltage_multiplier;

    // set the target voltage to the motor
    motor.move(target_voltage);

    // restart the counter
    loop_count=0;
  }
```

```
}

// function constraining the angle in between -pi and pi, in degrees -180 and 180
float constrainAngle(float x){
   x = fmod(x + M_PI, _2PI);
   if (x < 0)
       x += _2PI;
   return x - M_PI;
}

// LQR stabilization controller functions
// calculating the voltage that needs to be set to the motor in order to stabilize the pendulum
float controllerLQR(float p_angle, float p_vel, float m_vel){
  // if angle controllable
  // calculate the control law
  // LQR controller u = k*x
  //  - k = [39.5, 5.6, 0.16]
  //  - x = [pendulum angle, pendulum velocity, motor velocity]'
  float u =  39.5*p_angle + 5.6*p_vel + 0.16*m_vel;

  // limit the voltage set to the motor
  if(abs(u) > motor.voltage_limit*0.7) u = sign(u)*motor.voltage_limit*0.7;

  return u;
}
```

Figure 20 - Full Arduino LQR Control System Firmware

Documentation of Hardware Setup

Figure 21 below shows a photo of our test setup. It consists of the parts listed in the "Parts List" on page 4. The hardware for this project is straight forward. It consists of an Arduino nano with the simpleFOC shield plugged into it. Connected to the simpleFOC shield are the motor and encoders. The wiring harness that goes to the motor and motor encoder is covered in kevlar tube mesh. We would not recommend this to others as it tends to get caught on small edges. The entire system is clamped to a metal table via a very large N52 neodymium magnet. We would also not recommend this as magnets this size can easily break fingers and cause injury. A final note on the system our hardware setup had pinch points and lab members received minor cuts from getting fingers pinched in the hardware. It is highly recommended for others attempting this project to either modify the design to have less pinch points, or use more caution during experimentation. The sharp metal aluminum base did not help with safety and it is recommended that others trying to replicate this experiment build their test stand differently.

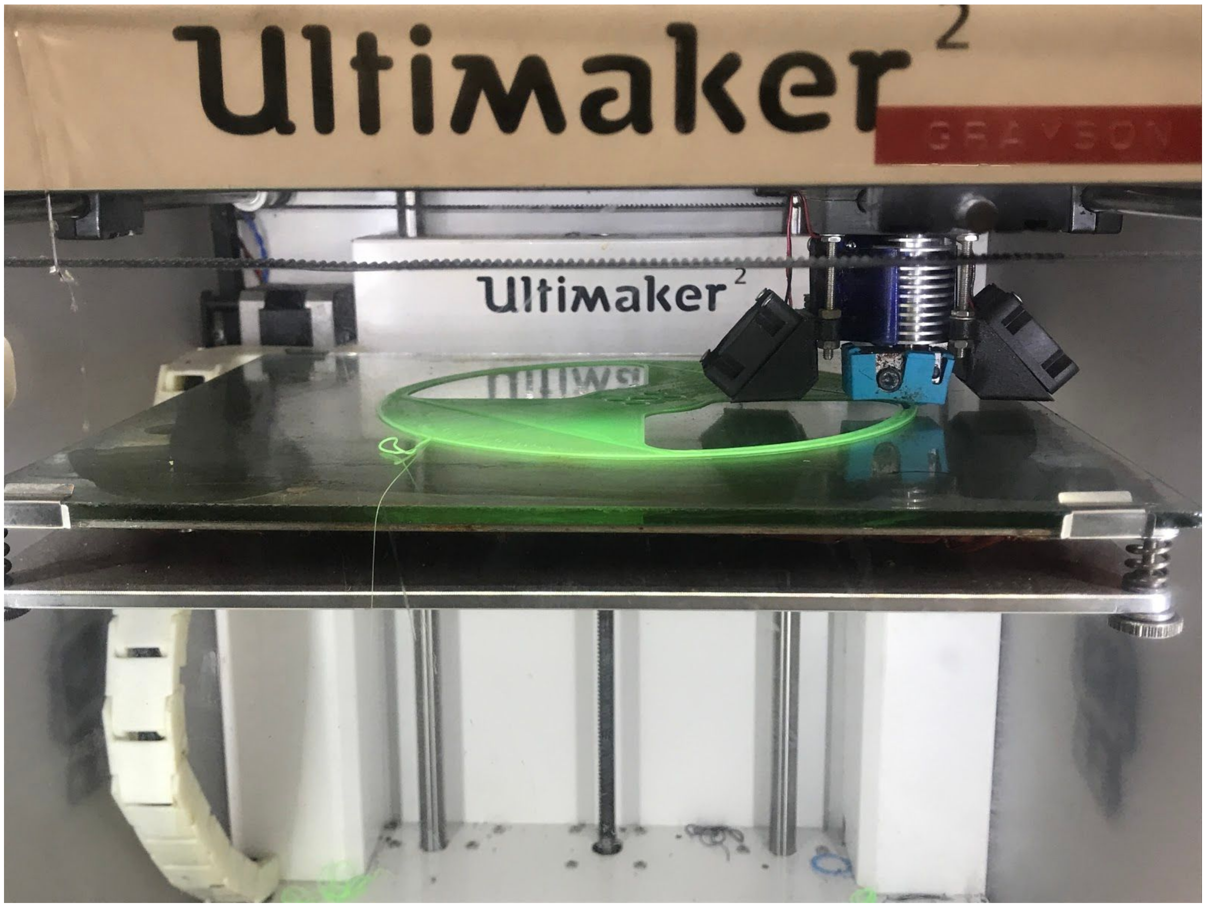


Figure 21 - Photograph of 3D Printing of The Reaction Wheel

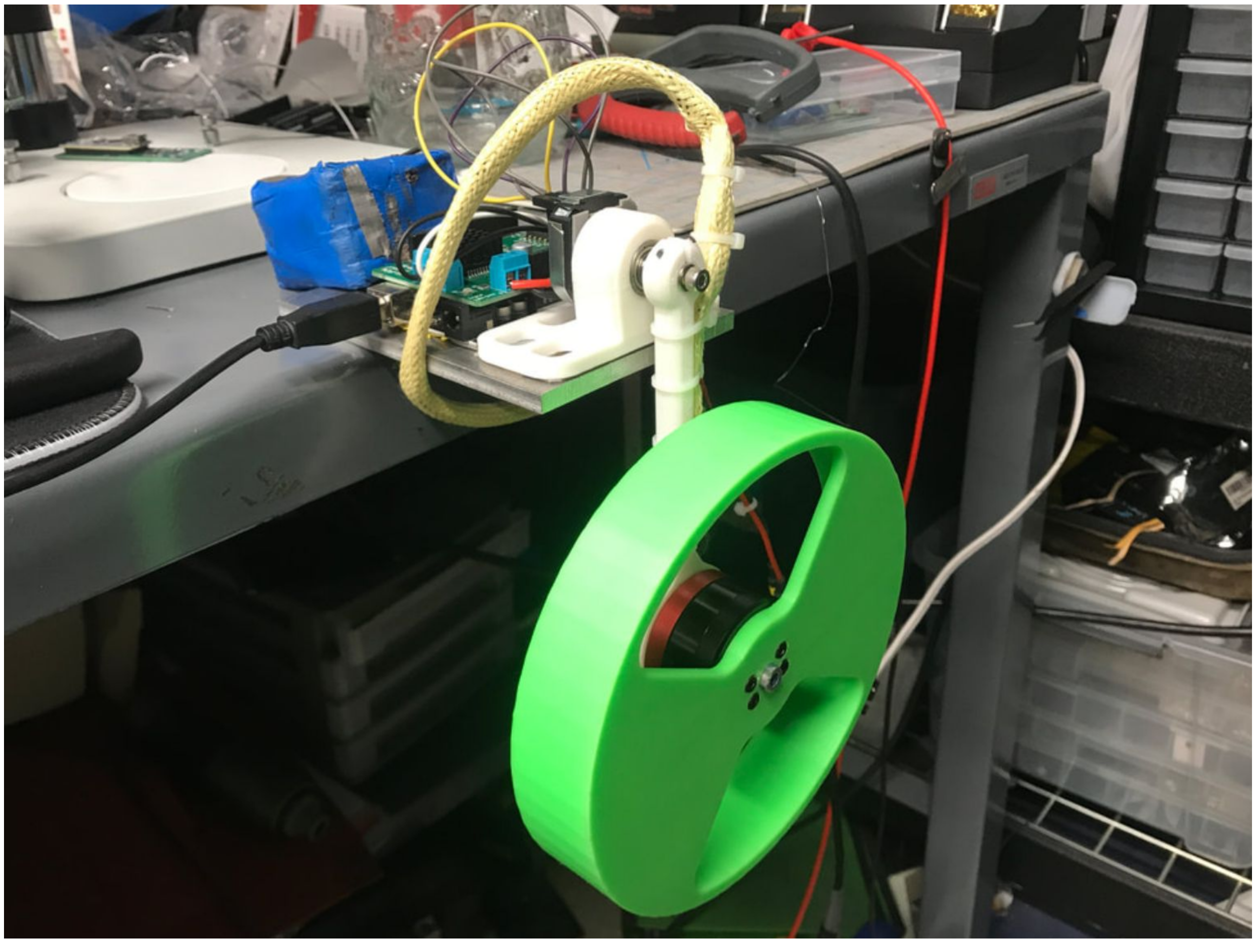


Figure 22 - Photograph of IPWRW Hardware Test Bench

Experimental Results for LQR controller

PID intro

PID stands for Proportional, Integral, Derivative and help eliminate error caused by disturbances combining the P, I and D. When setting these values, we must understand that if we set proportional gain too high, the controller can constantly overshoot (can lead to oscillation). There needs to be a good medium when setting this value as we need to reach small steady state error and a stable system. The integral factor eliminates steady-state offset and we choose a value that allows our system to curve to a line at amplitude of one. Finally, the derivative factor looks at the rate of change of the error, it works to fix the overshoot that is created by proportional and integral gain. For our inverted pendulum, we first simulate the system in MATLAB then use the pidtool() command to see the PID tuner window. Using the tuning tools in the top of the window, we adjust the plot until our desired output is shown. This shows our tuned response.

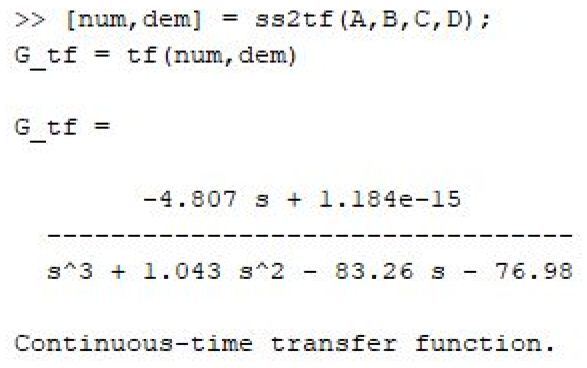
Then we click on show parameters and it tells us the values of Kp, Ki, and Kd to achieve the graph that we created (shown in section *Matlab for PID*).

System transfer function

$$TF = c^T(sI - A)^{-1}B$$

Figure 23 - SS to transfer function.

Using Matlab, we get the following transfer function for theta:



```
>> [num,dem] = ss2tf(A,B,C,D);
G_tf = tf(num,dem)

G_tf =

        -4.807 s + 1.184e-15
  --------------------------------
  s^3 + 1.043 s^2 - 83.26 s - 76.98

Continuous-time transfer function.
```

Figure 24 - Open-loop transfer function of the system for theta.

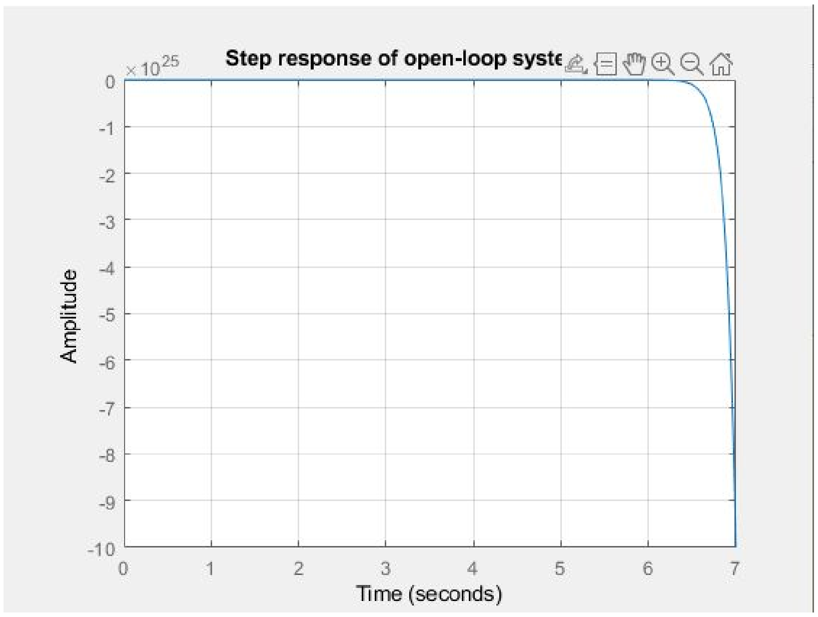


Figure 24 - Step response of open-loop system for theta.

A quick look at the step response of the open loop system shows that it is unstable in figure X.

Matlab for PID

With the acquired transfer function of the open-loop system, we then design the appropriate PID controller with Matlab PID tuner for the closed-loop system.

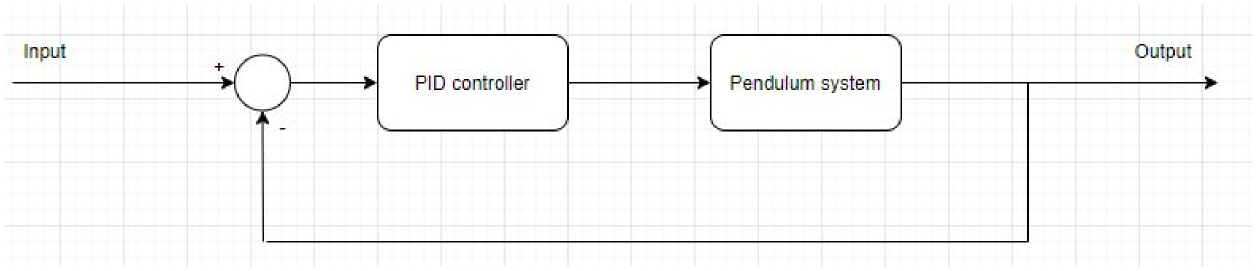


Figure 25 - Block diagram of the closed- loop system.

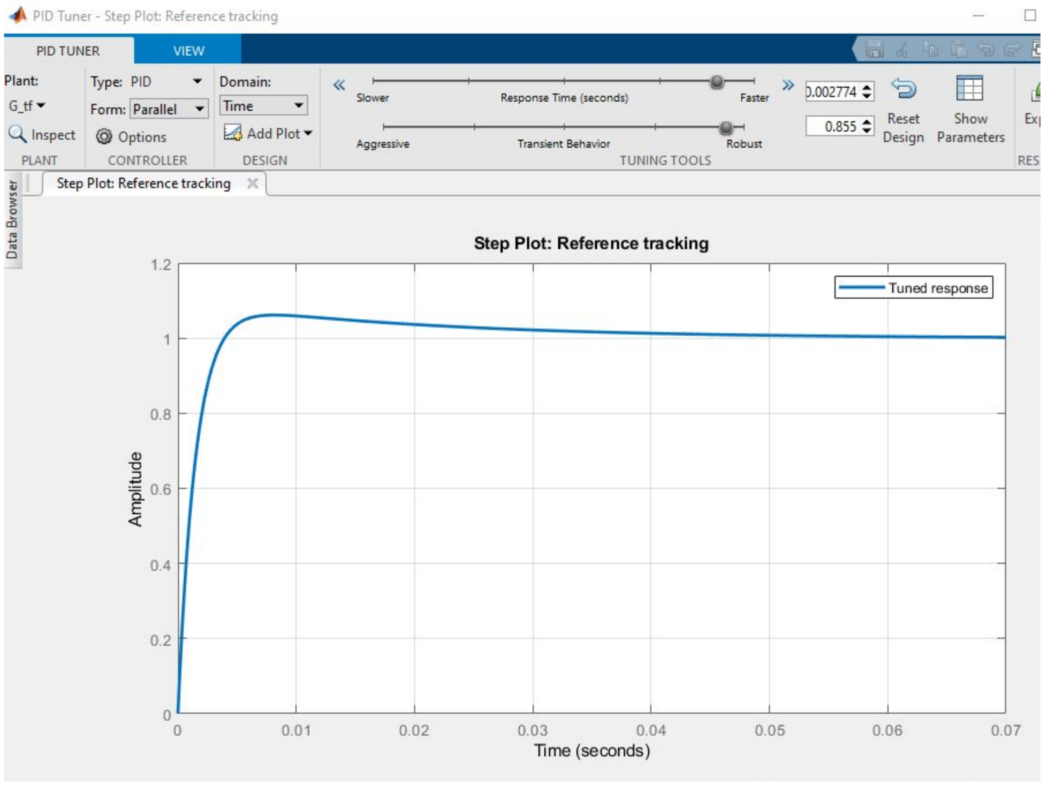


Figure 26 - PID Tuned Response

When parameters are identified, we see the controller parameters of the tuned response as:

```
Kp = -1040;
Ki = -4127;
 Kd = -65;
```

Although the system is said to be stable in these parameters, you can clearly see that the values of the PID controller are not "normal" due to being such a high number, for instance, Ki is around negative 85k. We are unsure on why this is, and expect to test this value in the experimental implementation of PID.

Arduino Firmware for PID Controller

```
#include <AutoPID.h>
#define sign(x) ((x) < 0 ? -1 : ((x) > 0 ? 1 : 0))
#include <SimpleFOC.h>
// software interrupt library
#include <PciManager.h>
#include <PciListenerImp.h>

// BLDC motor init
BLDCMotor motor = BLDCMotor(11);
// driver instance
BLDCDriver3PWM driver = BLDCDriver3PWM(9, 10, 6, 8);
//Motor encoder init
Encoder encoder = Encoder(2, 3, 2048);
// interrupt routine
void doA(){encoder.handleA();}
void doB(){encoder.handleB();}

// pendulum encoder init
Encoder pendulum = Encoder(A1, A2, 2048);
// interrupt routine
void doPA(){pendulum.handleA();}
void doPB(){pendulum.handleB();}
// PCI manager interrupt
PciListenerImp listenerPA(pendulum.pinA, doPA);
PciListenerImp listenerPB(pendulum.pinB, doPB);

//pid settings and gains
#define OUTPUT_MIN -255
#define OUTPUT_MAX 255
#define KP -1040
#define KI -4127
#define KD -65

double input, setPoint, output;
AutoPID myPID(&input, &setPoint, &output, OUTPUT_MIN, OUTPUT_MAX, KP, KI, KD);

unsigned long currentTime, previousTime;
float elapsedTime;
float error;
float lastError;

float cumError, rateError;

void setup() {
  Serial.begin(115200);
  int setPoint = 0;
   myPID.setBangBang(4);
  //set PID update interval to 4000ms
  myPID.setTimeStep(4000);
  // initialise motor encoder hardware
  encoder.init();
  encoder.enableInterrupts(doA,doB);
  // init the pendulum encoder
  pendulum.init();
  PciManager.registerListener(&listenerPA);
  PciManager.registerListener(&listenerPB);

  // set control loop type to be used
  motor.controller = ControlType::voltage;

  // link the motor to the encoder
  motor.linkSensor(&encoder);

  // driver
  driver.voltage_power_supply = 18;
  driver.init();
  // link the driver and the motor
  motor.linkDriver(&driver);

  // initialize motor
  motor.init();
  // align encoder and start FOC
  motor.initFOC();
}

// loop downsampling counter
long loop_count = 0;

void loop() {
  // ~1ms
  motor.loopFOC();

  // control loop each ~25ms
  if(loop_count++ > 25){

    // calculate the pendulum angle
    float pendulum_angle = constrainAngle(pendulum.getAngle() + M_PI);
    input = pendulum_angle;
    float target_voltage;
    if( abs(pendulum_angle) < 0.5 ) // if angle small enough stabilize
    {
//PID output here
myPID.run();
target_voltage = output;
if(abs(target_voltage) > motor.voltage_limit*0.7) target_voltage = sign(target_voltage)*motor.voltage_limit*0.7;
    }

    else // else do swing-up
      // sets 40% of the maximal voltage to the motor in order to swing up
      target_voltage = -_sign(pendulum.getVelocity())*motor.voltage_limit*0.4;

    // set the target voltage to the motor
    motor.move(target_voltage);

    // restart the counter
    loop_count=0;
    Serial.print(pendulum_angle);
    Serial.print("    ");
    Serial.println(target_voltage);
  }

}

// function constraining the angle in between -pi and pi, in degrees -180 and 180
float constrainAngle(float x){
    x = fmod(x + M_PI, _2PI);
    if (x < 0)
        x += _2PI;
    return x - M_PI;
}
```
Figure 27 - Arduino firmware for PID controller.

Experimental implementation of PID

For the implementation of the PID controller, we noticed that the system is unstable even though the PID controller is fully operational. Multiple attempts to tune the PID controller for different Kp, Ki and Kp values yield the same results. Further investigation shows that the velocity of the motor was the limiting factor for the implementation of the PID. We realized that there is a limit to how much our motor can spin. As such, the motor can only produce a small amount of acceleration, thus force, if only theta is taken into account. Using Matlab, we plotted the step response of the system for theta and motor velocity for comparison.

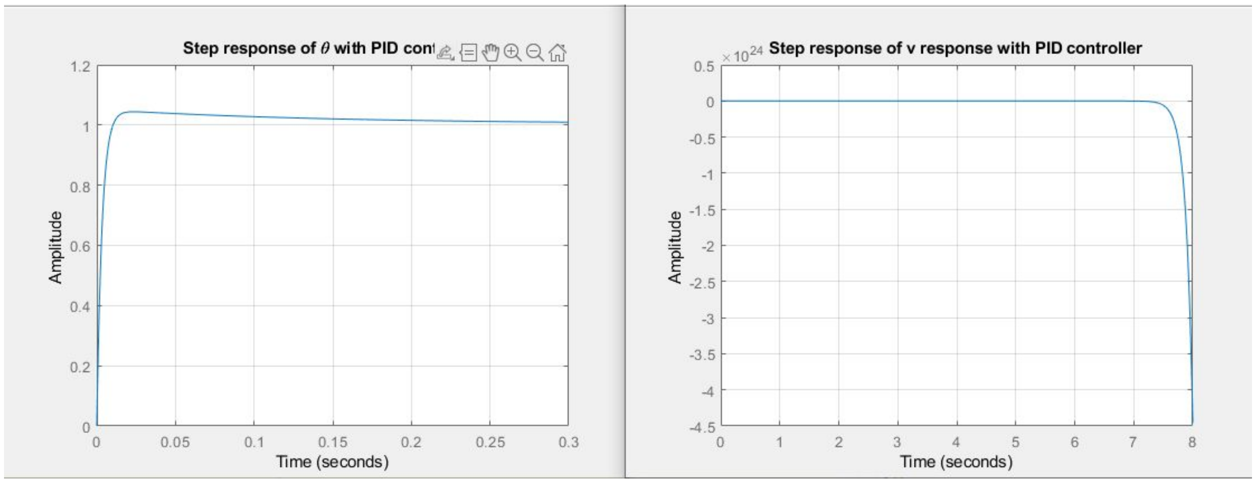


Figure 28 - Comparison of step response for theta and v systems.

From figure 28. we can clearly see that for selected parameters of Kp, Ki and Kd, the step response of the theta system is stable while the step response of v is unstable. As such, the speed of the motor quickly saturates at its maximum rpm and the motor no longer produces acceleration/force for the system.

PID Modeling In Matlab

```
%% linear system equations
A = [0          1       0;...
    g/l_cm      0       c_e*c_m/(J_p*R_a);...
    -g/l_cm     0       -c_m*c_e/(J_w*R_a)];

B = [0;...
    -c_m/(J_p*R_a);...
    c_m/(J_w*R_a)];

C = [1 0 0];
C2 = [0 0 1];
D = [0];

%% Transfer function in s domain
[num,dem] = ss2tf(A,B,C,D); %theta system
G_tf = tf(num,dem);

[num2,dem2] = ss2tf(A,B,C2,D); %v system
G_tf2 = tf(num2,dem2);

% PID tuning
pidTuner(G_tf,pid)

Kp = -1040;
Ki = -4127;
Kd = -65;
C = pid(Kp,Ki,Kd);
T = feedback(G_tf*C,1);
figure
step(T)
grid on
title("Step response of \theta with PID controller")

T2 = feedback(G_tf2*C,1);
figure
step(T2)
grid on
title("Step response of v response with PID controller")
```

<u>Discussion/Results</u>

Overall, we decided that the LQR controller is the best design for this system. The resulting system can be seen in our video demonstration with adequately good responses to disturbances. The PID controller, though, works in theory for the theta system, it fails to take into account other states of the system, in this case, motor velocity. As such, the PID controller is not suitable to control the pendulum.

<u>Conclusion</u>

In conclusion, we were able to create the project with a cost of $170 USD along with 3D printing parts for the wheel. We learned that 3D printing is fairly useful when creating a project like this. LQR control is a powerful tool to control real world systems and PID control is a bit difficult to use as we get odd gains when using the PID tuner and it fails to take other states of the system into account. The implementation of LQR is successful with good response to disturbances.

References page

Skuric, Antun. “Askuric/inverted_inertia_pendulum.” *GitHub*, 29 June 2019, github.com/askuric/inverted_inertia_pendulum.

Simplefoc. “Simplefoc/Arduino-FOC-Reaction-Wheel-Inverted-Pendulum.” *GitHub*, 24 May 2020, github.com/simplefoc/Arduino-FOC-reaction-wheel-inverted-pendulum.

Belascuen, Gonzalo, and Nahuel Aguilar. ARGENCON, 2018, pp. 1–9, *Design, Modeling and Control of a Reaction Wheel Balanced Inverted Pendulum*.