5-1-2021

# ACTIVE LOAD STABILIZATION ON STAIR CLIMBING VEHICLE VECTR SENIOR DESIGN TEAM 2021

Joseph Johnson

# ACTIVE LOAD STABILIZATION ON STAIR CLIMBING VEHICLE VECTR SENIOR DESIGN TEAM 2021

by

JOSEPH COLE JOHNSON

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2021

ACKNOWLEDGMENTS

I would like to thank Dr. Ashley Guy for his guidance and oversight of this project, including a great deal of technical and practical advice on project specifics. In addition, I would like to extend thanks to Dr. Raul Fernandez for high-level management of the Senior Design Honors process as a whole.

Our team thanks Dr. Huff for generous lease of several components for our prototype, without which our proof-of-concept device could not be built to full scale.

May 1, 2021

# ABSTRACT

## ACTIVE LOAD STABILIZATION ON STAIR CLIMBING VEHICLE VECTR SENIOR DESIGN TEAM 2021

Joseph Cole Johnson, B.S. Mechanical Engineering

The University of Texas at Arlington, 2021

Faculty Mentor: Ashley Guy

The VECTR Stair Climbing Robot team has designed a product to take loads up a stairway in a safe manner. This device carries the load in a basket. The basket's rotation is impeded by a set of parallel rotational dampers. This setup is sufficient to retard the oscillation of the load about its point of equilibrium, but is passive and slow to respond. As such, a system was designed to actively control the load orientation. This system will use a sensor to measure change in angle and adjust the load to keep it stable. This system provided a tangible decrease in response time in comparison to the current passive system, and consisted of three basic elements: a circuit mounted gyroscope; a processor with the ability to interpret data from said sensor and issue a command in response; and a brushed DC motor. A mathematical model of the problem was proven, and a control system

established as an approximation of the physical system. The final result was a physical model of the system demonstrating the viability of the concept.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

# CHAPTER 1

# BACKGROUND AND PREVIOUS WORK

## 1.1 Introduction and Justification

The VECTR Senior Design team vehicle uses a damped pendulum approximation to model the swinging of the load basket on the robot as it moves. While multiple options were discussed, this system was chosen due to its simplicity and low cost. As part of the normal duties of the design team, this model was simulated to validate its performance.

While the chosen system works adequately, given the theoretical commercial nature of this product, attention should be paid to the overall safety of the vehicle in operation. As the load swings, it gains momentum. If the magnitude of this momentum is too great, it could possibly destabilize the vehicle as it ascends the stairs, causing it to fall. If a person or pet were below, they could be injured by the heavy and bulky falling robot.

## 1.2 Previous Work

As previously mentioned, the uncontrolled simply damped system was modeled and simulated by the team. A free-body diagram was drawn up and an equation of motion for the system was obtained. The equations of motion were used in MATLAB with an ode45 numerical integrator to simulate the system [1]. A comparison was made between the behavior of the swinging load with and without a damper to justify its use in the prototype.

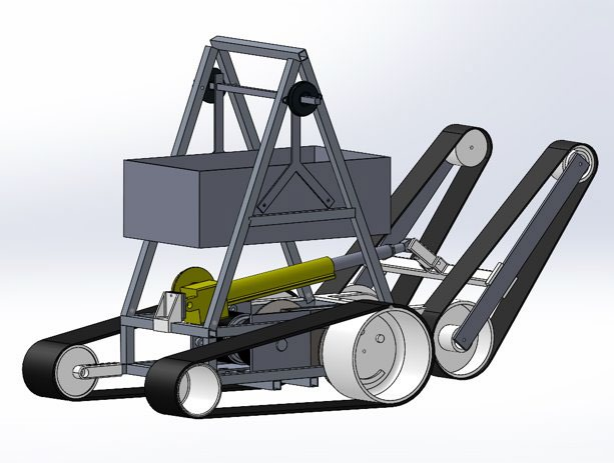


Figure 1.1: CAD Model

Figure 1.1 shows the full CAD model of the VECTR prototype. Note how the load swings on an axis of rotation while the rotation is damped by rotational dampers (shown as black discs). This system is approximated for the purposes of this report as a pendulum, shown below.

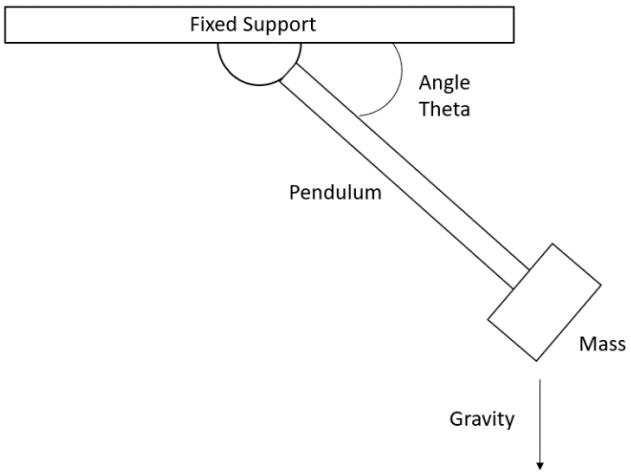



Figure 1.2: Free-Body Diagram of Load

The swinging of the load was approximated by the above diagram, where a mass on the end of a lever arm creates a moment about the hinged fixed support. In a controlled

system, there is a torque provided by an electric motor at this point to counter the moment of the pendulum.

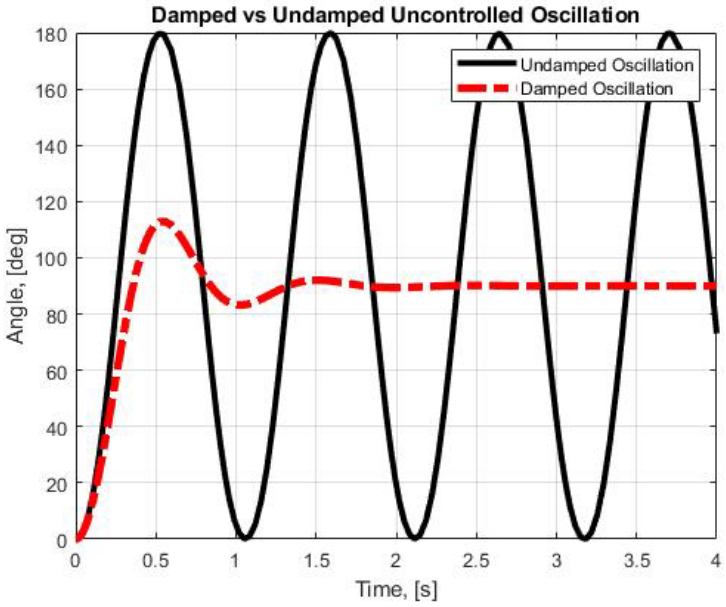



Figure 1.3: Damped versus Undamped Pendulum

Shown above is the simulated response of the pendulum acting with and without a damper. Note that the damper reduces the oscillation, but only after several swings and at roughly 1.5 seconds. The red dashed line is the damped oscillation, and the solid black line is the undamped oscillation, which swings without diminishing as would be expected.

## 1.3 Need for Improvement

The need for improvement in the load stabilization system boils down to a judgement between “outstanding” and “good enough”. While the current system has been proven to be good enough from a technical standpoint, there exists room for improvement in both the transient response of the controller to a standard impulse input and to any random input noise that may be encountered. In a product that is to be marketed to an elderly or disabled audience, it would seem that the safety of the product should not just meet a minimum standard.

From a technical standpoint, it is desired to reduce both the amplitude of the oscillations as well as their duration, to reduce the forces placed on the vehicle as it ascends the stairs, and to keep the motion of the load from interfering with any autonomous navigation that may be in action.

The introduction of active leveling in this load carrying vehicle is representative of an incremental improvement in technology common in many engineering applications, in which processes and methods evolve from passive, reactive assemblies to those able to respond with input from a sensing device. While this system could be analog, the most common incarnation of such a control system involves a microcontroller to receive data, compare it to some known standard, and initiate a response. This basic concept is responsible for a plethora of engineering improvements in many systems, from air conditioning climate control to active cruise control in automobiles.

As such, this modification represents a proof of concept for an upgrade to the existing vehicle design and is typical of technological progression in an engineering context. This project also incorporates a plethora of higher-level engineering courses such as Introduction to Controls Systems, Circuit Analysis, Kinematics and Dynamics of Machines, and Mechanical Design I, and requires this knowledge be applied in a creative combination of disciplines.

# CHAPTER 2

# METHODOLOGY

## 2.1 Analytical Work

To actively control the oscillation of the load, a PID controller was constructed to command the response of an input torque on the pendulum as provided by a DC motor. To construct this controller, the equations of motion for the pendulum with an input torque were derived. These equations of motion were converted to the "s" domain by a Laplace transform.

After taking a Laplace transform, a second order approximation of the system was established by defining performance parameters and using said parameters to find a target damping ratio and natural frequency. Once these were found, polynomial long division was conducted to solve equation for Kd, Kp, and Ki in terms of system parameters such as mass, length, and gravity. These equations could then be solved simultaneously to find the PID constants (Kp, Ki, Kd) after system constants were specified. The free-body diagram shown in Chapter 1 was used, reproduced below, with the addition of an input torque.

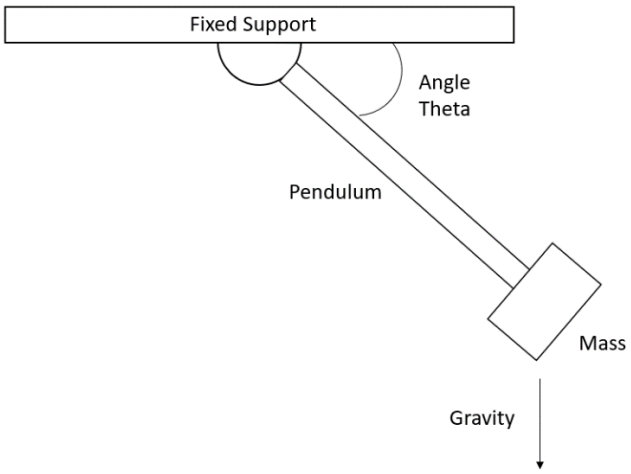



Figure 2.1: Free Body Diagram

$$\{1\} \qquad \ddot{\theta} = \frac{1}{mL^2}T - \frac{g}{L}sin(\theta)$$

$$\{2\} \qquad \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{mL^2} \end{bmatrix} T$$

Equation 1, the equation of motion for the system, was converted to a state space system, shown in Equation 2. This representation was used to simulate the system in MATLAB using an ode45 numerical integrator. A small angle approximation was used to linearize the system.

Using a small angle approximation, a Laplace domain model of the system was constructed. After the previously mentioned performance parameters were defined and long division carried out, the following constraint equations for the PID terms were found. This long division was necessary to use a second order approximation of a higher order system. The poles of the higher order system were set to ten times the lower order approximation to effectively negate their influence on the system.

Equations 3 and 4 quantify the performance constraints placed on the system.

$$\{3\} \qquad T_s = \frac{4}{\zeta \omega_n} = 1$$

$$\{4\} \qquad OS\% = e^{\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right)} = 10$$

Table 2.1 Constant Equations

| Constant | Equation |
|---|---|
| Kp | $\frac{1}{mL^2}$ |
| Ki | $\frac{687}{mgL}$ |
| Kd | $\left(166 - \frac{687(ml^2)}{mgL}\right)\frac{1}{mgL}$ |

Table 2.1 contains the derived equations for the values of the PID constants in terms of physical parameters. These parameters can be adjusted for varying values to generate correct constants for systems of differing sizes. A full derivation can be found in Appendix B.

## 2.2 Simulation

To plot the performance of the system in MATLAB, the initial free-body diagram was used to derive a state space representation of the system. Using the torque as an input, this state space system was written into MATLAB code using an ode45 numerical integrator function. The desired control input (such as input from a gyroscope) could then be fed into the integrator/controller function, which would take the error between the current and desired states and issue a response based on the PID constants and the physical parameters of the system. In this way, adjustments could be made to the input function to view the system response to a constant input, a sinusoid, or white noise. After tuning the

PID constants to provide a reasonably fast response with minimal overshoot or steady state error, the controlled and passive systems could then be compared to view the relative advantages. Since the ode45 integrator uses variable time steps to reduce overall error, interpolation functions were needed to scale the desire input vectors to the correct length for integration and plotting.

Basic signal flow can be seen in the following figure. While this figure was generated in Simulink, it serves a conceptual purpose only. A reference value of zero, which refers to degrees measured by the gyroscope with respect to gravity, is used to create an error signal based on the output angle of the system under the influence of inertial forces and the controller's provided torque.

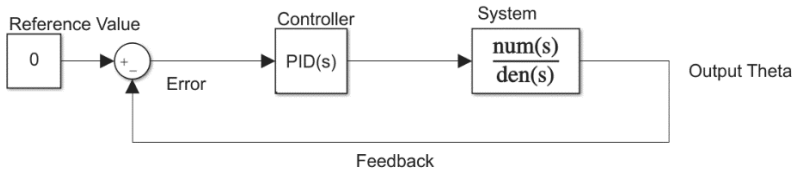



Figure 2.2: Control Diagram

## 2.3 Hardware and Testing

With the understanding that this analysis was to augment future iterations of the VECTR prototype, it was desirable to construct a physical model of the system to prove the ability of the controller in an applied setting. To do this, a PID controller was established in Arduino. This process was similar to those described above, however, the input was real angle data from a gyroscope mounted on the pendulum. This data was then compared to the desired set point (in this case zero) and the resulting error fed into a PID controller function to determine a value for the magnitude of the response. This value was then scaled and sent to the DC motor controller as a PWM signal. The motor controller

then determined the speed and rotation direction of the motor based on the PWM signal from the Arduino and a digital signal corresponding to the current quadrant location of the pendulum. This allowed the motor rotation to change directions as the pendulum moved about the desired angular location.

Early in testing, it was found that the error signal would grow at an unduly massive rate if the pendulum was at a large angle relative to the desired location. To remedy this, the controller was divided into sections based on the quadrant location of the pendulum. If the pendulum was +/- 45 degrees from its target, the controller was active. If the pendulum was outside this range, the controller was bypassed and the motor was fed a constant command at a set speed, whether clockwise or counterclockwise.

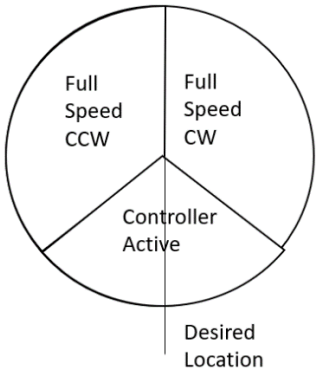



Figure 2.3: Arduino Controller Logic

Testing components included an Arduino Mega, a MPU6050 gyroscope/accelerometer module, a LN298H motor driver, and a 7.2V NiMh rechargeable battery.

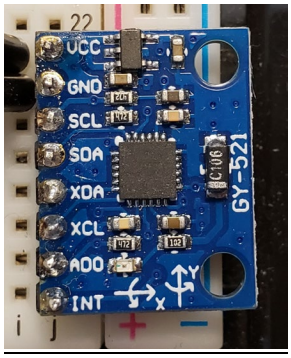



Figure 2.4: MPU6050 Gyroscope

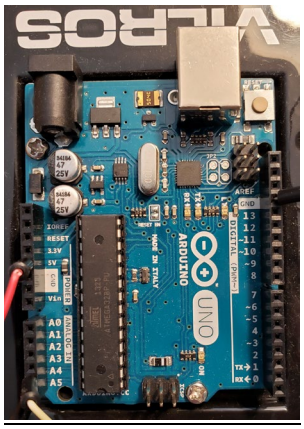



Figure 2.5: Arduino Mega

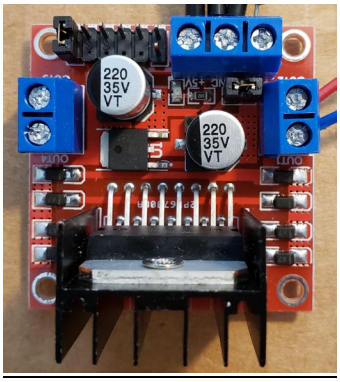



Figure 2.6: LN298H Motor Controller

A simple diagram of wiring and signal flow can be seen in Figure 2.2, and shows a representative circuit defining the relative connection and location of components.

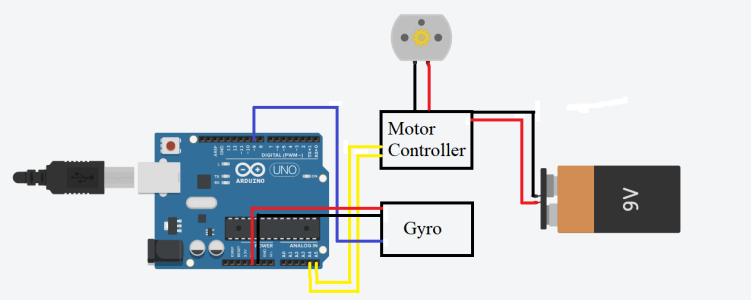



Figure 2.7: Basic Diagram

# CHAPTER 3

# RESULTS AND CONCLUSIONS

## 3.1 Simulation Results

The system was simulated in MATLAB using the state space representation described in the previous chapter. The resulting system behavior was plotted as a function of time in response to various inputs.

The most basic case represents system response to a constant input of zero degrees, which should be the difference between the gyroscope and the gravity normal vector in the plane of rotation. The following figure represents a system response with a 45-degree initial pendulum displacement. In this arrangement, the Ki term is negligible and can be set to zero if desired, as the natural equilibrium state is equal to the input.

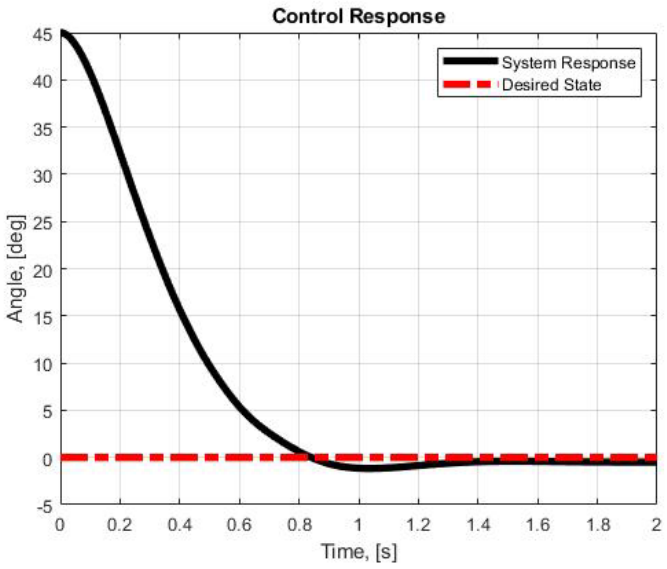



Figure 3.1: System Response to Nonzero Initial Conditions

The system was also simulated in response to stochastic white noise. This noise was generated by the MATLAB wgn function as a 1x n array, where n is the desired length. Due to computational requirements, this vector was created in advance and rounded to whole numbers. The magnitude of the noise was also scaled to represent a realistic range of degree values.

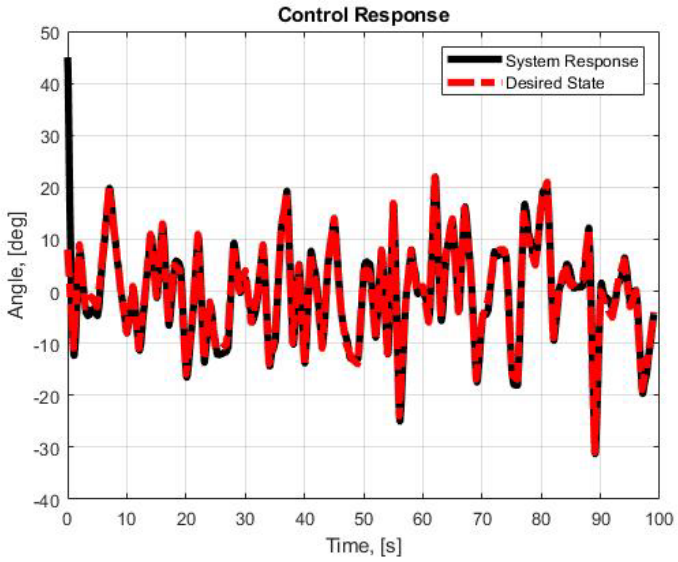



Figure 3.2: System Response to Random Noise

It is also beneficial to note the response of the system to a nonzero constant input, simulating the system response to a desired pendulum angle that is not straight up and down. In this configuration the value of Ki must be nonzero to remove steady state error.

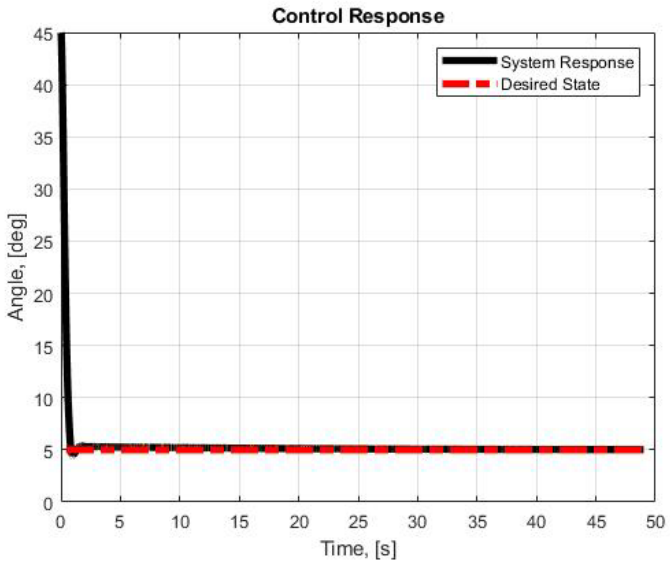



Figure 3.3: Nonzero Constant Input

## 3.2 Physical Model Testing

The hardware introduced in Chapter 2 was used to construct a working model of the basket-pendulum system. This allowed a visual representation of the system working and provided clarity to the layout of the system. Though it was small and not to actual prototype scale, the components could easily be scaled up for full size use. This would require simulating the system using full size system parameters and obtaining the torque curves from MATLAB, as has been discussed.

To obtain a more accurate system response, it would be necessary to include the gear ratio in the simulation such that the output angular speed and the torque step up would be apparent in the results.

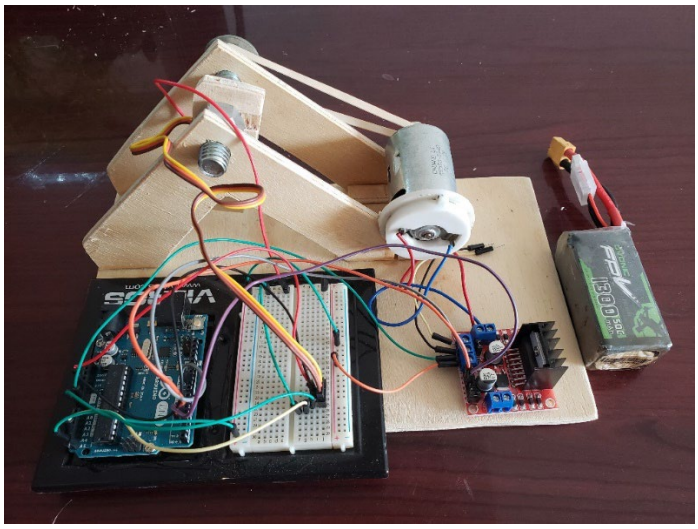


Figure 3.4: Physical Model

The Arduino can be seen in the lower left, the motor controller in the lower right, and the brushed DC motor in the middle left center of the image. The gyroscope is obscured behind the test stand, but the four wires that relay information and power can be seen leading to the breadboard.

## 3.3 Summary and Conclusion

An active load leveling system for the VECTR platform was designed and modeled in both state space and the Laplace domains, and the equations of motion for this active load leveling system were used to simulate the new load system using MATLAB. Hardware to model the active load stabilization system was sourced and wired to establish a model of the system, and Arduino code was written to control the motor in response to data from a gyroscope and the PID controller. This establishes the active load leveling system as practical, and demonstrates that this system could be easily scaled up to full size by specifying desired system loads; easily obtainable from simulation data and physical system constants.

In conclusion, the designed active load stabilization system is a dramatic improvement on the existing passive load stabilization in response time, overshoot, and response to random input, and is easily adaptable to different physical dimensions. In addition, the demonstration of a physical model establishes the feasibility of establishing this system on future VECTR prototypes.

# APPENDIX A

# ARDUINO AND MATLAB CODE

Arduino Code

```
#include<Wire.h>
#include<math.h>
//Variable Declarations
//////////////////////////////////////////////////////
const int MPU_addr=0x68;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
int minVal=265; int maxVal=402;
int motor1pin1 = 2;int motor1pin2 = 3;
int x; double y; double z; double var; double sinhh; double coshh;
//////////////////////////////////////////////////////
//PID constants
//////////////////////////////////////////////////
double kp = 1; double ki = 0; double kd = 1;
unsigned long currentTime, previousTime;
double elapsedTime;
double error;
double lastError;
double input, output, setPoint;
double cumError, rateError;
double NormOut; double PWMNormOut;
//////////////////////////////////////////////////

void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(9, OUTPUT);
  //
  setPoint = 270;   //set point at zero degrees
}

void loop(){
Wire.beginTransmission(MPU_addr);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,14,true);
AcX=Wire.read()<<8|Wire.read();
AcY=Wire.read()<<8|Wire.read();
AcZ=Wire.read()<<8|Wire.read();
int xAng = map(AcX,minVal,maxVal,-90,90);
```

```
int yAng = map(AcY,minVal,maxVal,-90,90);
int zAng = map(AcZ,minVal,maxVal,-90,90);
x= RAD_TO_DEG * (atan2(-yAng, -zAng)+PI);
y= RAD_TO_DEG * (atan2(-xAng, -zAng)+PI);
z= RAD_TO_DEG * (atan2(-yAng, -xAng)+PI);
Serial.print("AngleX= ");
Serial.println(x);
Serial.println("-----------------------------------------");

input = x;

  if ((x<225)&&(x>90)){
    Serial.print("Region B, out of bounds, max speed");
    Serial.println("-----------------------------------------");
    analogWrite(9, 150); //need to make this go to motor controller, this is max throttle
outside reasonable angle range
    digitalWrite(motor1pin1, HIGH);
    digitalWrite(motor1pin2, LOW);
  }
  if (((x<90)&&(x>0))||(x>315.00)){
    Serial.print("Region A, out of bounds, max speed");
    Serial.println("-----------------------------------------");
    analogWrite(9, 150); //need to make this go to motor controller, this is max throttle
outside reasonable angle range
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, HIGH);
  }

  if ((x>225)&&(x<315)){
    output = computePID(input);
    NormOut = output/45;
    PWMNormOut = abs(NormOut*255);
    var = tanh(x);
    Serial.print("Region C, controller active\n");
    Serial.print("output= \n");
    Serial.println(output);
    Serial.print("NormOut= \n");
    Serial.println(NormOut);
    Serial.print("PWMNormOut= \n");
    Serial.println(PWMNormOut);
    Serial.print("var= \n");
    Serial.println(var);
    Serial.println("-----------------------------------------");

    analogWrite(9, PWMNormOut); //need to make this go to motor controller
    if (x<265){
```

```
    digitalWrite(motor1pin1, HIGH);
    digitalWrite(motor1pin2, LOW);
   }
   if (x>275){
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, HIGH);
   }
   if ((x>265)&&(x<275)){
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, LOW);
   }
  }
//Controls motor direction, replace with hyperbolic tangent
//sinh = .5*(exp(x)-exp(-x));
//cosh = .5*(exp(x)+exp(-x));
//var = sinh/cosh;

delay(100);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
double computePID(double inp){
      currentTime = millis();                              //get current time
      elapsedTime = (double)(currentTime - previousTime);       //compute time elapsed
from previous computation

      error = setPoint - input;                           // determine error
      cumError += error * elapsedTime;                        // compute integral
      rateError = (error - lastError)/elapsedTime;              // compute derivative

      double out = kp*error + ki*cumError + kd*rateError;             //PID output

      lastError = error;                                  //remember current error
      previousTime = currentTime;                             //remember current time

      return out;                                         //have function return the PID output
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Matlab Code
Uncontrolled Damped Oscillation Simulation

```
clear
clc
close all

m =1500;
g = 9.81;

[t,x] = ode45(@eqns, [0 4],[ 0 0 ]);
[t2,x2] = ode45(@eqns2, [0 4],[ 0 0 ]);

theta = x(:,1); theta = theta*180/pi;
thetad = x(:,2);
plot(t,theta); hold on; title('Damped vs Undamped Uncontrolled
Oscillation')
xlabel('Time, [s]'); ylabel('Angle, [deg]');

theta2 = x2(:,1); theta2 = theta2*180/pi;
thetad2 = x2(:,2);
plot(t2,theta2);

function dx = eqns(t,x)
dx = zeros(2,1);

m = 15;
g = 9.81;
L = .2;

dx(1) = x(2);
dx(2) = g/L*cos(x(1));
end

function dx = eqns2(t2,x2)
dx = zeros(2,1);

m = 15;
g = 9.81;
L = .2;
b = 5;
dx(1) = x2(2);
dx(2) = g/L*cos(x2(1))-b*x2(2);
end
```

Matlab Code
Controlled Oscillation Simulation

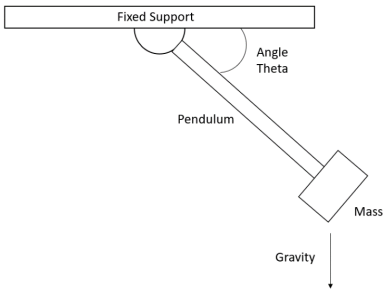
```
clear
clc
close all

global error input dt time i torque;
error = 0;
dt = 1;
%input = wgn(100,1,1);
%input = 10*sin(0:360);
input = zeros(1,3);
%input = [8 -11 9   -3  -1  -3  8   20  10  0   -8  1   -11 -1  11  -1
13  -6  5   4   -16 -6  11  -13 -2  -11 -11 -10 9   0   4   -6  -1  9
-14 -9  11  19  -10 5   -14 7   1   -11 6   14  -2  -10 -13 -14 4   4
-9  8   -12 17  -24 -3  8   0   1   -6  22  -5  8   14  -4  16  5   -17
-5  -2  7   8   8   -16 -17 15  9   5   17  21  -9  0   5   1   2   2
12  -31 0   -3  -5  2   6   -3  0   -19 -13 -4];
%input = 5*[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];
time = 0:dt:dt*(length(input)-1);

torque = zeros(1,145);
i = 0;

[t,x] = ode45(@eqns, [time(1) time(end)],[ 45 0 ]);

thetatarget = interp1(time, input, t);
e = x(:,1)-thetatarget;
%torque2 = interp1(time, torque,t);

figure()
thetagraph = x(:,1); %thetagraph = thetagraph*180/pi;
plot(t,thetagraph); hold on; title('Control Response')
xlabel('Time, [s]'); ylabel('Angle, [deg]');
plot(t,thetatarget); legend('Theta', 'Input');
% figure()
% plot(t, torque); hold on; xlabel('Time, [s]'); ylabel('Torque,
[N*m]');

function dx = eqns(t,x)
dx = zeros(2,1);

persistent PrevInput PrevTime
if isempty(PrevInput)
    PrevInput = 0;
    PrevTime = 0;
end
global error input dt time i torque;

i = i+1;
```

```
m = 20;
g = 9.81;
L = .2;

Kp = 20;
Kd = 7;
Ki = 2;

Current = interp1(time,input,t);
if t==PrevTime
    inputdot = 0;
else
    inputdot  = ( Current - PrevInput ) / (t-PrevTime);
end
PrevInput = Current;

T = Kp*(Current - x(1)) + Kd*( inputdot - x(2)) + Ki*error;
torque(i) = T;
error = error + (Current - x(1))*(t-PrevTime);

dx(1) = x(2);
dx(2) = (1/(m*L^2))*T - g/L*sin(x(1));

PrevTime = t;
end
```

# APPENDIX B

# ANALYTICAL WORK

Derivation of PID Constants




$$mL^2\ddot{\theta} + mgL\sin\theta = -T$$

If we assume sin(theta)=theta (small angle approximation)

$$mL^2\ddot{\theta} + mgL\theta = -T$$

Taking Laplace Transform:

$$\theta(s)[mL^2s^2 + mgLs] = -T(s)$$

$$\frac{\theta(s)}{T(s)} = \left[\frac{1}{mL^2s^2 + mgLs}\right]$$

If we set $mL^2 = A$ and $mgL = B$ for convenience:

$$\frac{\theta(s)}{T(s)} = \left[\frac{1}{As^2 + Bs}\right]$$

Taking a second-order approximation of the characteristic polynomial in series with a PID controller, using performance targets discussed in Chapter 2:

$$s^2 + 2\zeta W_n s + W_n = (As^2 + Bs)\left(Kp + \frac{Ki}{s} + Kds\right)$$

Substituting values in the left side to achieve target dominant poles:

$$Akds^3 + s^2(AKp + BKd) + s(AKi + BKp) + BKi = (s + 15)(s^2 + 8s + 45.83)$$

$$Akds^3 + s^2(AKp + BKd) + s(AKi + BKp) + BKi = (s^3 + 23s^2 + 166s + 687)$$

Matching powers of s:

$$AKd = 1$$

$$(AKp + BKd) = 23$$

$$166 = (AKi + BKp)$$

$$BKi = 687.45$$

From this, we extract:

$$Kd = \frac{1}{A}$$

$$Ki = \frac{687}{B}$$

$$Kp = \left(166 - \frac{687A}{B}\right)\left(\frac{1}{B}\right)$$

# APPENDIX C

## LIST OF HARDWARE

List of Hardware

LN298H Motor Controller

Arduino Mega

MPU 6050 Gyroscope

Brushed DC Motor

REFERENCES

[1] VECTR Senior Design Team Proposal, Spring 2021 UTA Student Design team, drafted 1/21; members A. Burge, H. Pavlik, G. Hadley, J. Crowson, K. Matthee, J. Johnson

[2] Control Systems Engineering, 7th Ed. Nise, Norman S., Wiley Publishers, Copyright 2015

## BIOGRAPHICAL INFORMATION

Joseph Cole Johnson is a student at the University of Texas at Arlington. He is a member of the Honors College, having been admitted as an incoming freshman, and was awarded the Terry Foundation Traditional scholarship for exemplary leadership, academic performance, and service to community.

During his time at UTA, Cole completed numerous Honors projects in completion of his Honors degree, most notably in Dynamics and Kinematics and Dynamics. Active on campus, Cole served a term as Engineering Senator and performed various leadership roles in the UTA Cycling Club, Pro Life Mavericks, and the Baptist Student Ministry, and achieved the College of Engineering Dean's list.

Cole will graduate with a Bachelor of Science in Mechanical Engineering in May 2021, with a degree from the UTA Honors College.