Spring 2021 MAE 106:
Self Balancing Robot
Final Report - Team 5

Jose Mari U. Aquino
Mechanical Engineering

Barry Pham
Aerospace Engineering

Steven Young
Mechanical Engineering

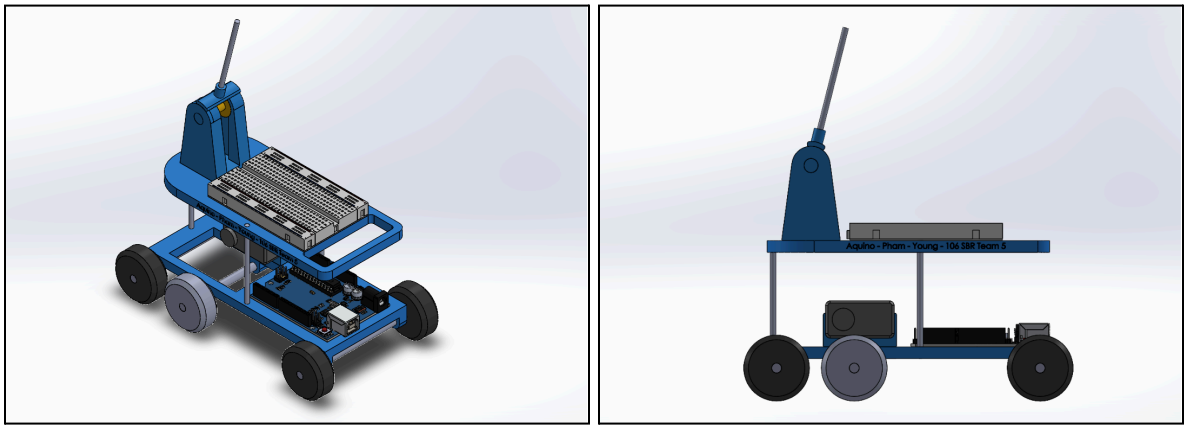


Figure 1. Isometric view of the robot (left). Side view of the robot (right).

System Description

Design Concept

Three Design Goals were determined during the manufacturing process of the self-balancing robot: the robot needed to be (1) Easy to Manufacture, have (2) Fast Corrective Motion, and (3) Account for Overcompensation.

Design Goals

1. **Ease of Manufacturing:** This goal was met by using primarily LEGO parts to construct the frame of the robot. Compared to other manufacturing methods such as 3D printing, LEGO parts were readily available and enabled rapid prototyping. LEGO parts could be easily switched out, varying gear ratios could be tested, and orienting electronics throughout the assembly was easier.
2. **Fast Corrective Motion:** This goal was addressed by using a gear train to increase the speed transmitted from the servo to the wheels of the cart. It was important to increase the speed in this way to achieve fast corrective motion of the cart, as the Servo on its own could not provide the necessary speed. A gear train provided a higher output RPM than the Servo alone.
3. **Accounting for Overcompensation:** This goal was met by increasing the length of the inverted pendulum. Adjusting the length of the inverted pendulum reduced the degree to which the proportionality controller would “overshoot” and oscillate. Reducing overshoot improved the robot’s ability to stabilize the pendulum.

In the final product, the use of LEGO parts eased the fabrication process and the increased length of the inverted pendulum improved stability. However, the gear train did not produce the desired increase in speed for the robot to sustain its balance.

Robot

CAD Model

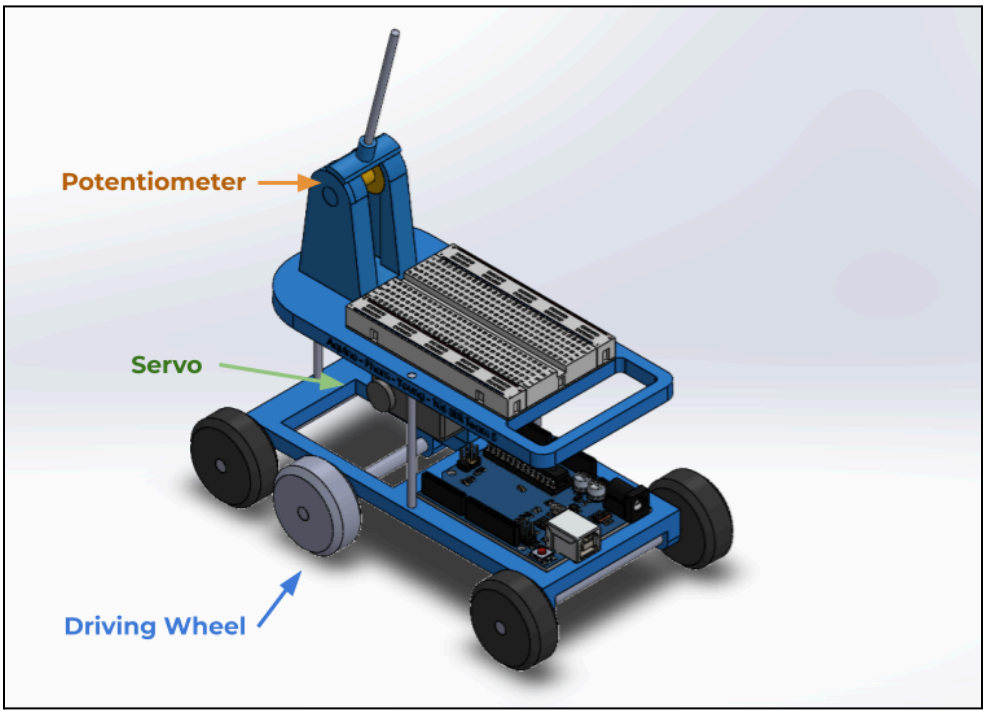


Figure 2. Isometric View of Self-Balancing Robot.
Servo powers the Driving Wheel to move the Robot back and forth.

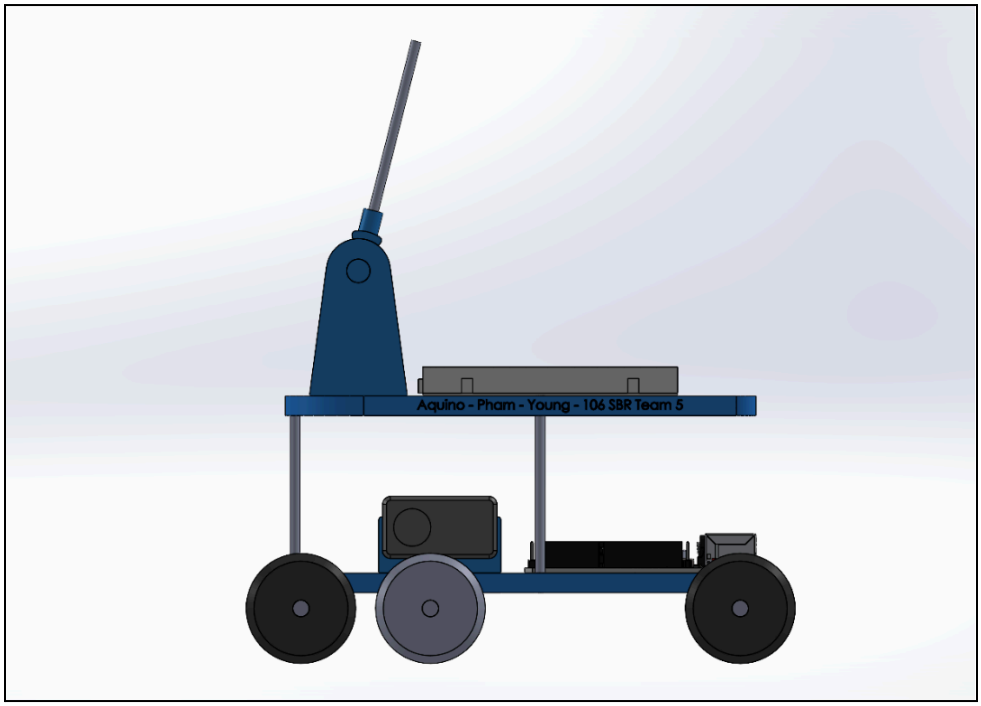


Figure 3. Side View of Self-Balancing Robot.

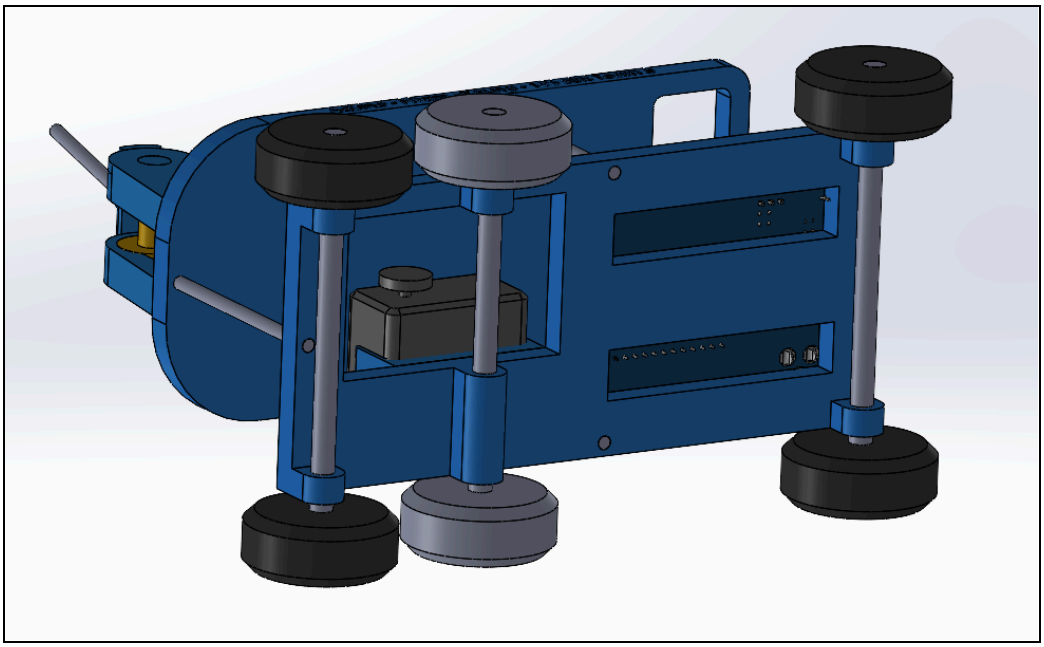


Figure 4. Underside of the carriage. The grey wheels are the wheels powered by the servo via a gear train..

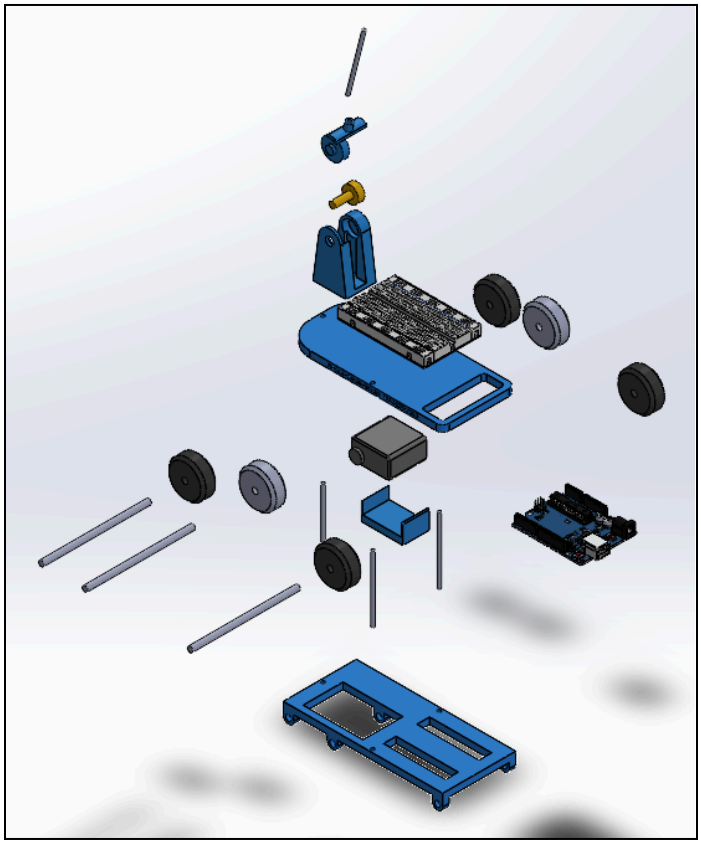


Figure 5. Exploded view of all the components of the robot.

Fabricated Model

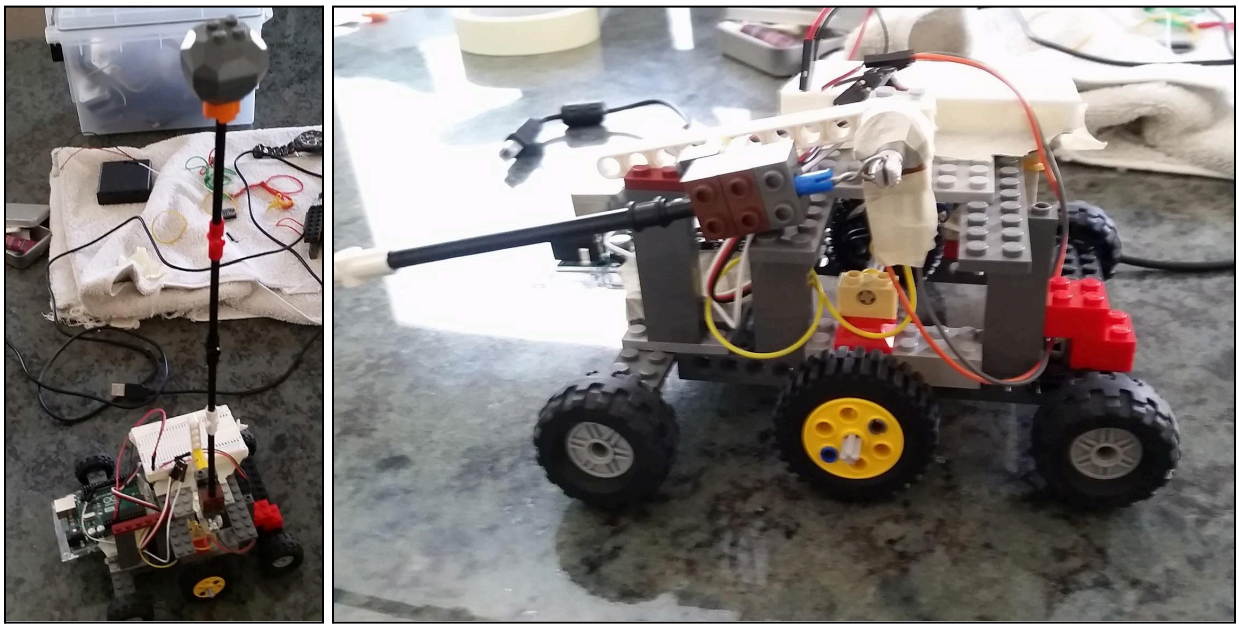


Figure 6. Full view with Pendulum (Left). Side view of cart with Potentiometer and Driving Wheel visible (Right).

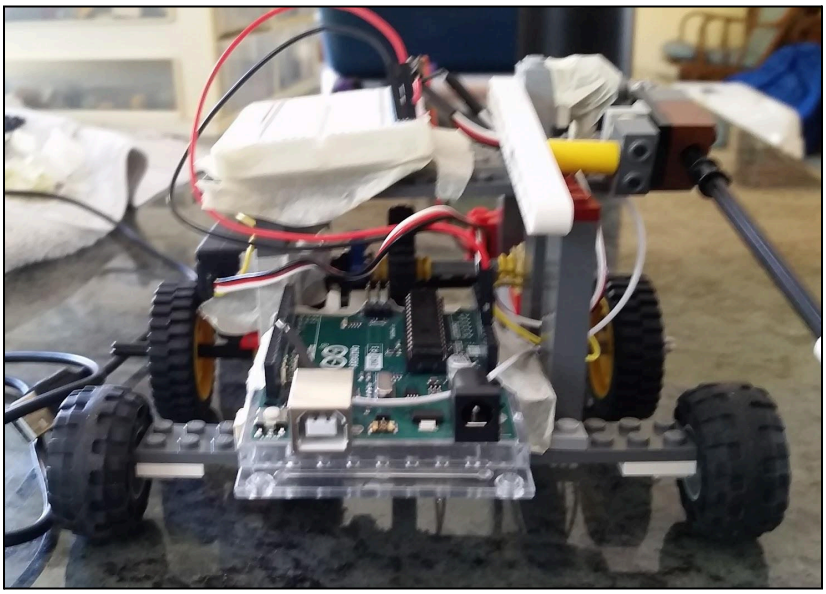


Figure 7. Front view of cart with Arduino, Breadboard, and Wiring visible.

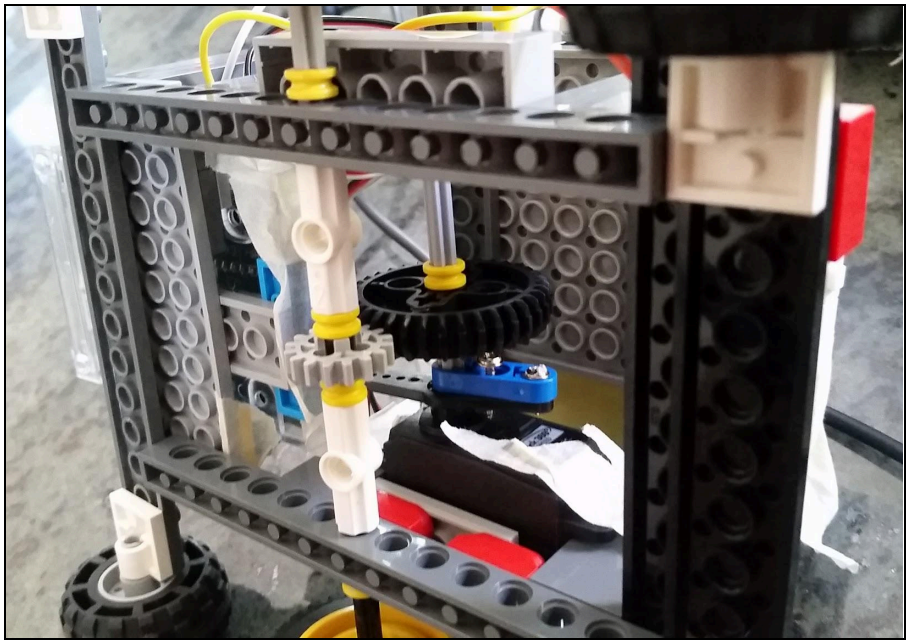


Figure 8. Underside of carriage with Servo and Gear Train visible.

Control Law

The system was controlled using a **Closed-Loop Control System**. The Arduino would read the position of the potentiometer attached to the inverted pendulum, compare the reading to an internal constant value for the desired position of the potentiometer, and send a signal proportional to the difference between these values to the servo to achieve the necessary correction.

Code

```
#include <Wire.h>
#include <Servo.h>
#include <LSM303.h>
#include <Filter.h> // Downloaded from the MegunoLink Library
Servo servo;

int servoPin = 9; //change to whichever pin the white wire goes to
int servo_pos = 90; // starting servo position which is at rest

float angleDes = 497; // the desired angle to which the compass heading
will be compared

unsigned long currentMillis;  // time in milliseconds
```

```
unsigned long oldMillis;  // previous measurement of time in milliseconds

float sampPeriod = 10; // Set to whichever value the Sample Period should
be
float angleResponse; // the response variable

float kp = 1.5; //proportionality constant

ExponentialFilter<float> FilteredReading(20, 0);

void setup() {
  Serial.begin(9600);
  pinMode(9,OUTPUT); //switching pin to send data
  Wire.begin();
  compass.init();
  compass.enableDefault(); //Set-up functions

  servo.attach(servoPin);  //connect the servo to the pin


  Serial.println("Done with setup!");
}

void loop() {
  currentMillis = (float)millis();
  if ((currentMillis - oldMillis) >= sampPeriod) //sampling rate
  {

    //Filtering the data
    float angle = analogRead(A0); //reading the voltage from the
    FilteredReading.Filter(angle);
    float reading = FilteredReading.Current(); //new filtered readings

    servo_pos = kp *(angleDes - reading)+90; //Closed loop angle servo
response
    servo.write(servo_pos);

    //debugging values
    Serial.print("90 ");
    Serial.print(angleDes);
    Serial.print(" ");
    Serial.print(reading);
    Serial.print(" ");
```

```
    Serial.print("  ");
    Serial.println(servo_pos);
    oldMillis = currentMillis;
  }
}
```

Testing and Development

Performance Criterion

A performance criterion that we experimentally optimized was the time it takes for pendulum to return to equilibrium given a disturbance.

Experimental Parameter

An experimental parameter that will be varied in this will be the length of the pendulum. Constant parameters will be the gain and the desired angle along with the degree of disturbance.

Experimental Results (continues on next page)

Experimental Results

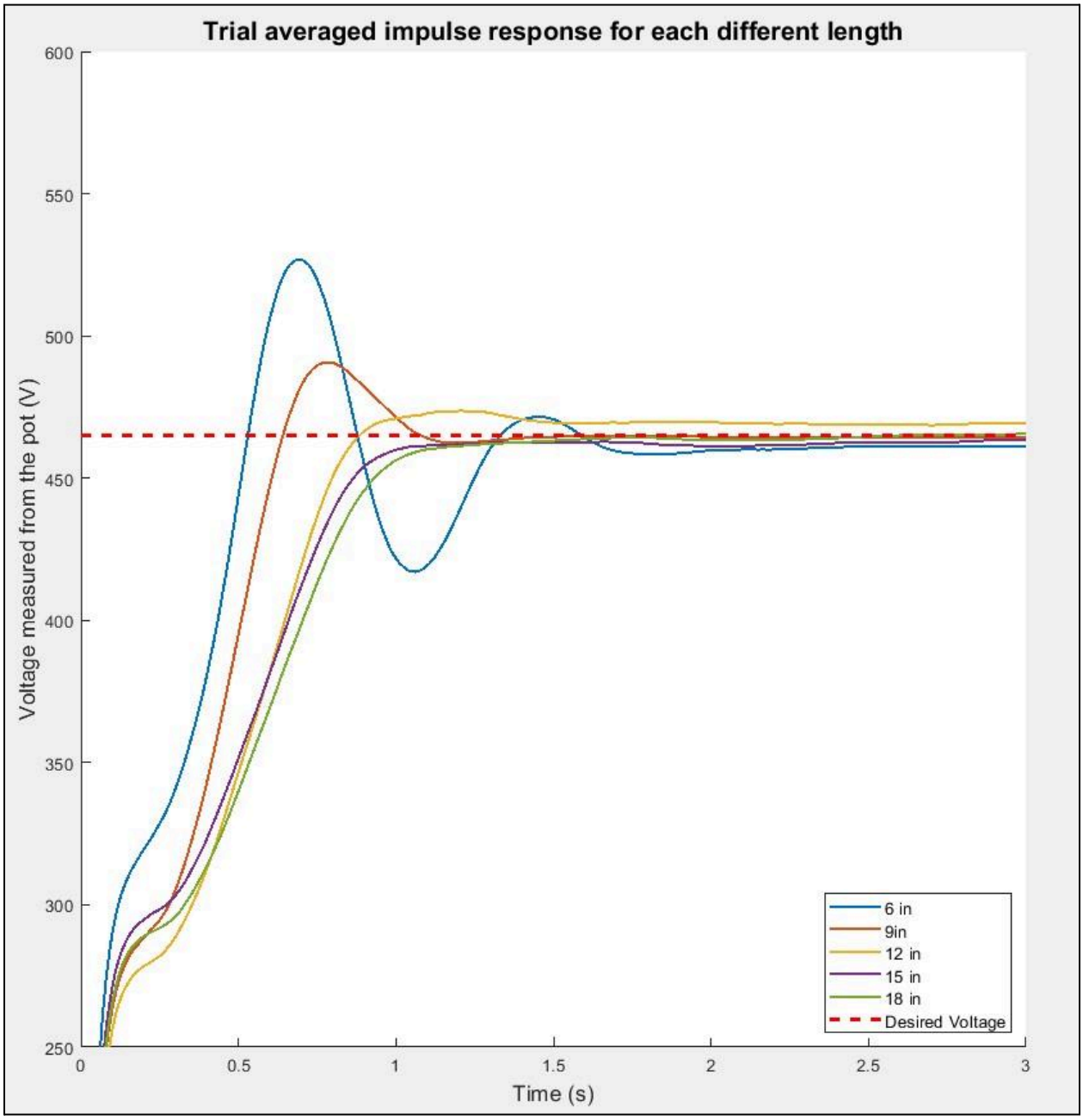


Figure 9. Trial Averaged response for each pendulum length

For each different pendulum length, we had 10 trials. With this, we took the average results of each trial data and plotted the results of each of the different pendulum lengths as can be seen in the results above in *Figure 8*. Using this we can get an overview of the effect of the pendulum length on the time it takes for the pendulum to reach equilibrium

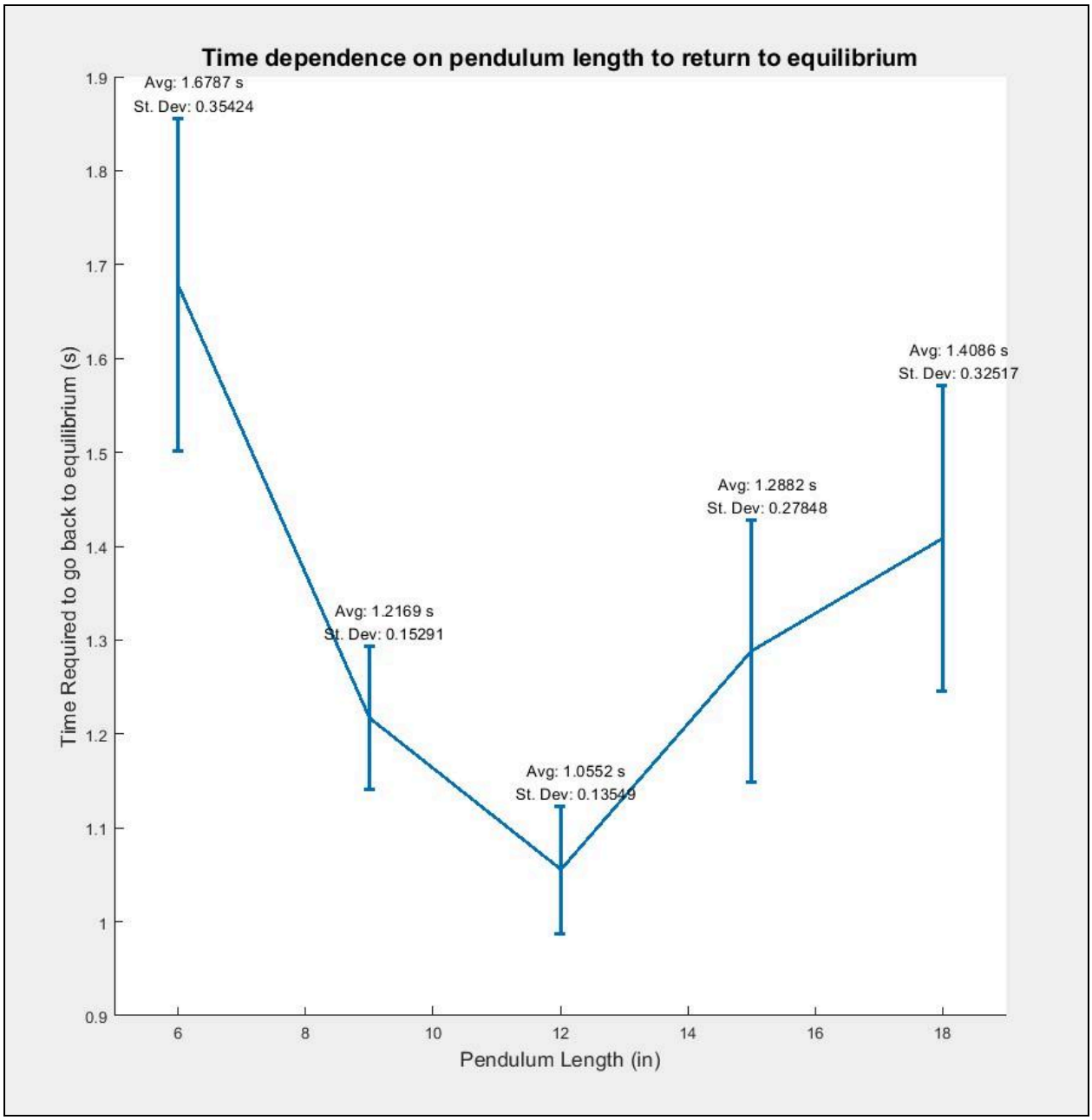


Figure 10. Time required to reach equilibrium as a function of pendulum length

From the data in the trials, we found the time that it took for the pendulum to reach equilibrium for each trial and then averaged each of them. The graphical results can be found in Figure 9 and as it can be seen there is a general parabolic shape to the graph. From this data, the most optimal length of the pendulum would be at the lowest point of the graph which is 12 inches as it provides the shortest average time and with relative consistency.