Proportional Control of a Brushless Motor System

ME 670

Simon Popecki

James Skinner

Jesse Feng

Table of Contents

[INTRODUCTION 3](#_Toc482724463)

[METHODS 4](#_Toc482724464)

[EXPERIMENTAL SETUP 7](#_Toc482724465)

[DESIGN CONCEPT 8](#_Toc482724466)

[SYSTEMS MODELLING 9](#_Toc482724467)

[CODE 9](#_Toc482724468)

# INTRODUCTION

The goal of this project was to control a motor precisely and repeatably by sending signals to an ESC from a personal computer (workstation) through a microcontroller. The motor controlled the speed of different Magnus effect airfoils for the purpose of windtunnel testing in the UNH student windtunnel. This project required the development of a specialized PWM system since the ESC used required a specific range of pulse widths rather than an analog voltage. In addition to computer control of the motor speed, a proportional control system was implemented to govern the rate of change of motor rotational speed. The proportional control system helped to prevent the motor from stalling while accelerating from low to high speeds.

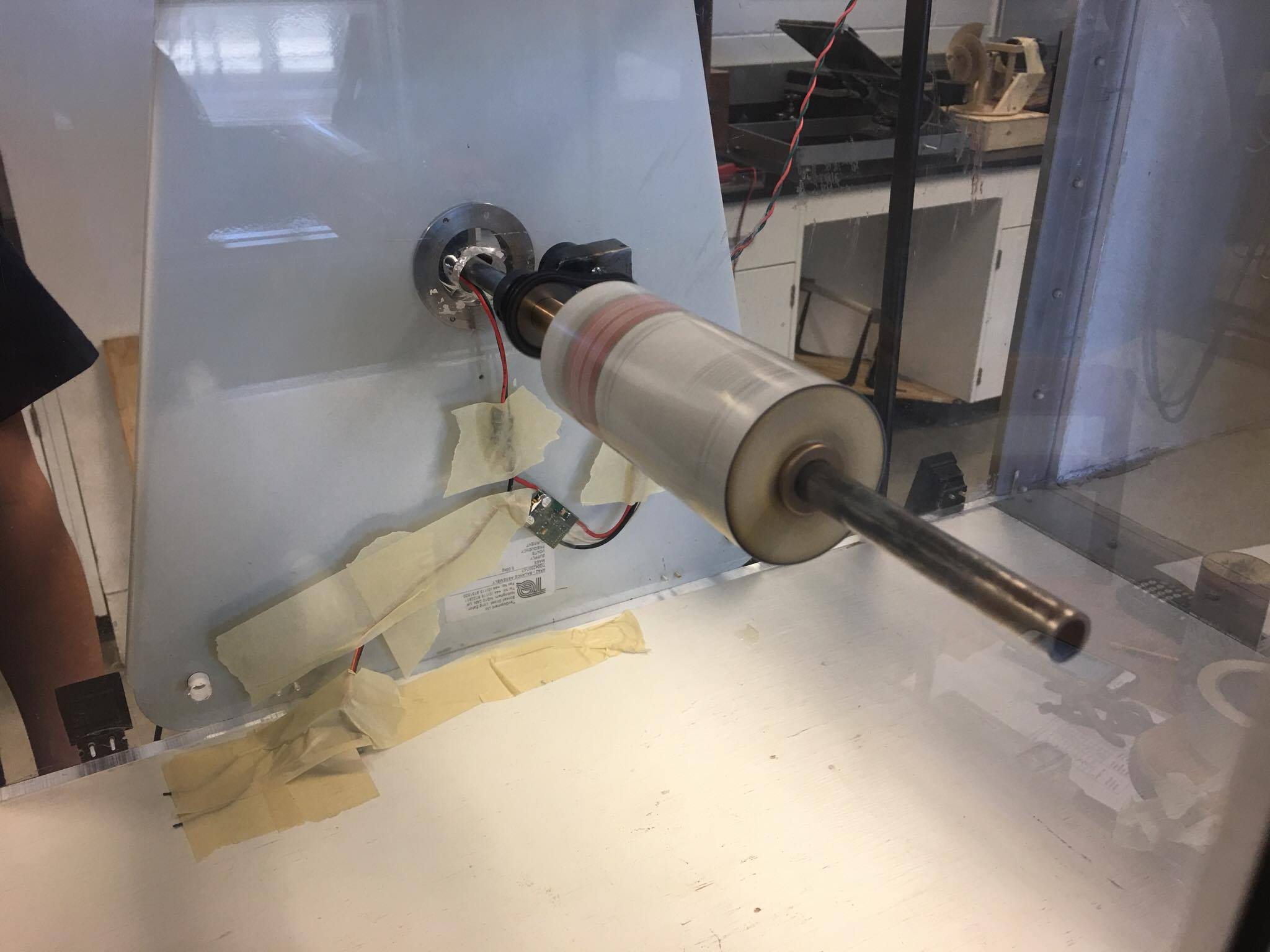


Figure 1 – The smallest airfoil at speed in the UNH student windtunnel. This airfoil is rotating at approximately 6,000 RPM in 30 m/s airflow.

# METHODS

Code was written in C++ and compiled using the Arduino IDE built in compiler. The program is stable on the Elegoo UNO (Arduino UNO clone) with ATmega16U2 processor architecture.

A human operator is able to communicate with the Arduino from a personal computer (workstation) via serial over USB.

The Thunderbird 18 ESC will not accept conventional 490 Hz PWM from the Arduino’s built in PWM pins - instead a 50 Hz pulse must be sent, with a duty cycle of 1-2 milliseconds on, and total period of 20 milliseconds. These values were found by copying the output of a Spektrum dx6i transmitter and BR6000 receiver by watching pulse width values on the Tektronix 515 oscilloscope. A 1 millisecond on pulse corresponds to zero throttle, and a 2 millisecond on pulse corresponds to full throttle. Note that in the code, pulse width values are handled in microseconds rather than milliseconds.

Any arbitrary normalized input (in this case a desired power level in percentage) can be sent to the Arduino which will subsequently determine the difference between the current outputted pulse width, and the target pulse width to be sent to the electronic speed controller. The program will then approach the new target pulse width at a speed determined by two proportional control tuning parameters in the software – the *denominator* of the fraction of pulsewidth difference per step, and the time per step. Increasing the fractional parameter will increase the response speed. A fractional value of 1 would cause the motor to reach its steady state value in one time step, and a fractional value of zero would result in the motor never reaching it’s steady state value (this would require the denominator of the fractional tuning parameter to be infinite, which is not possible due to floating point precision limitations). In practice the maximum denominator value is 255, as the constant that stores this value is an 8 bit unsigned integer. Any inputs larger than 255 can result in buffer overflow and subsequent unexpected results if the memory size was not changed as well. The default (denominator) parameter is 4, which results in a fractional parameter of ¼, meaning that the proportional control algorithm adds ¼ of the pulse width difference per time step to the outputted pulse width. The default time step parameter is 200 milliseconds. The constant names of the respective factors in the code are: RESPONSETUNING and TIMETUNING, and can be seen in the code section of the report.

**Mechanical Construction**

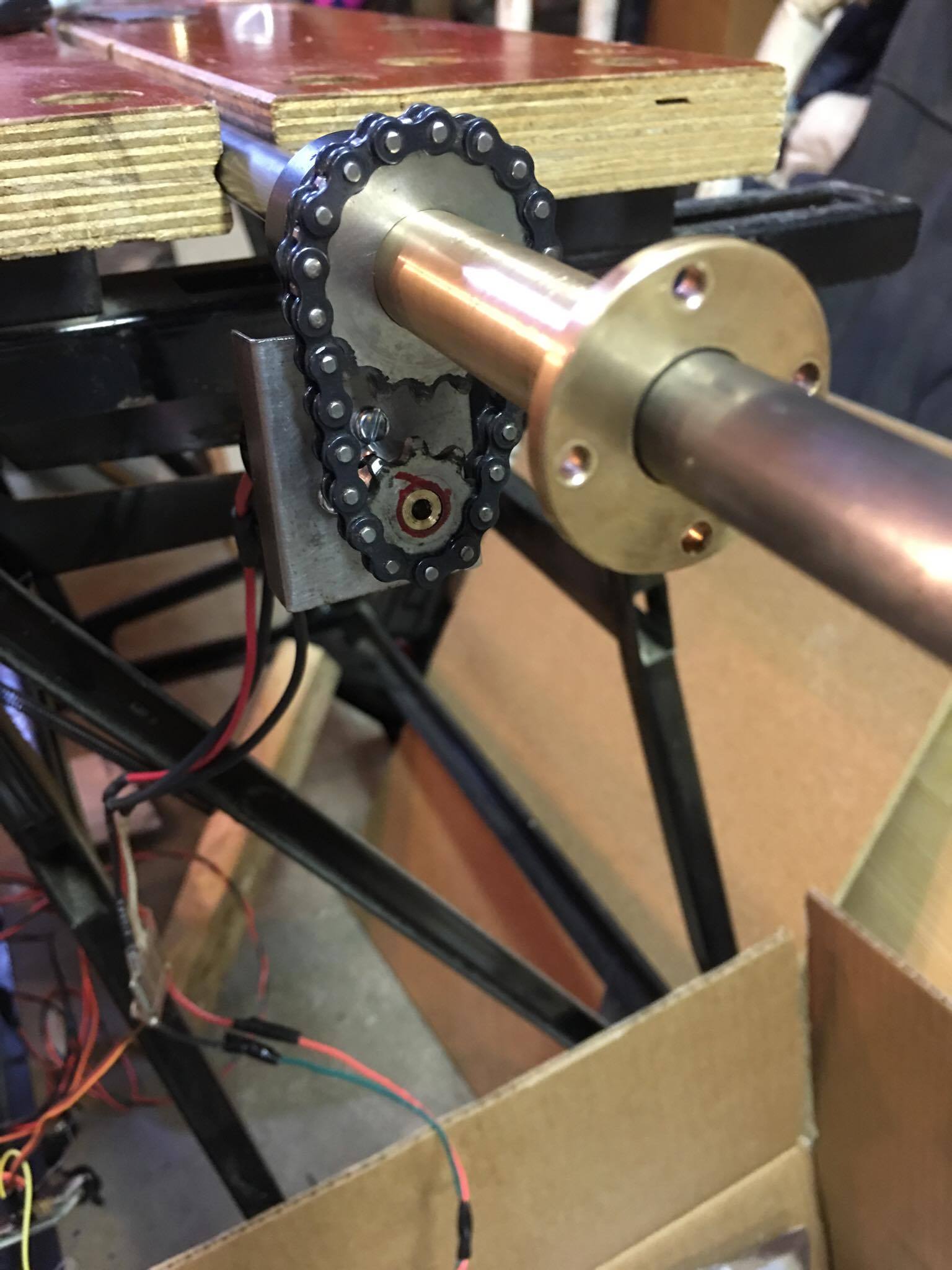


Figure 2 – Testing the fit of the motor mount bracket before welding. The bronze part is the shaft that connects to the airfoil.

The assembly used in testing was designed to fit into the load balance of the student wind tunnel in Kingsbury hall. A motor mounting bracket was fabricated from sheet metal and welded onto a shaft collar, allowing for axial and radial positioning of the motor mount along the shaft. Torque was transferred from the motor to the airfoil by sprockets and roller chain via a plain bearing connected to the end of the airfoil. This allowed the airfoil to rotate on a stationary shaft, which is required because the shaft must be anchored in the force balance. The end caps of the airfoil were lazer cut from plywood, balanced for rotation up to 6,000 RPM, and press fit into the ends of each airfoil (beer can). The pressfit allowed the end caps to allign themselves with the axis of the shaft, which was required to prevent binding allowing for smooth operation. Any time the airfoil was removed from the shaft carelessly or dropped, the allignment process was repeated. The process consisted of running the motor at low speed for roughly 20 seconds, or in the case of the largest airfoil – manually turning the airfoil many times, until satisfactory allignment was achieved.



Figure 3 – A shaft insert is being cut so that the sprocket (.25” ID) could fit on the motor shaft (.125” OD).

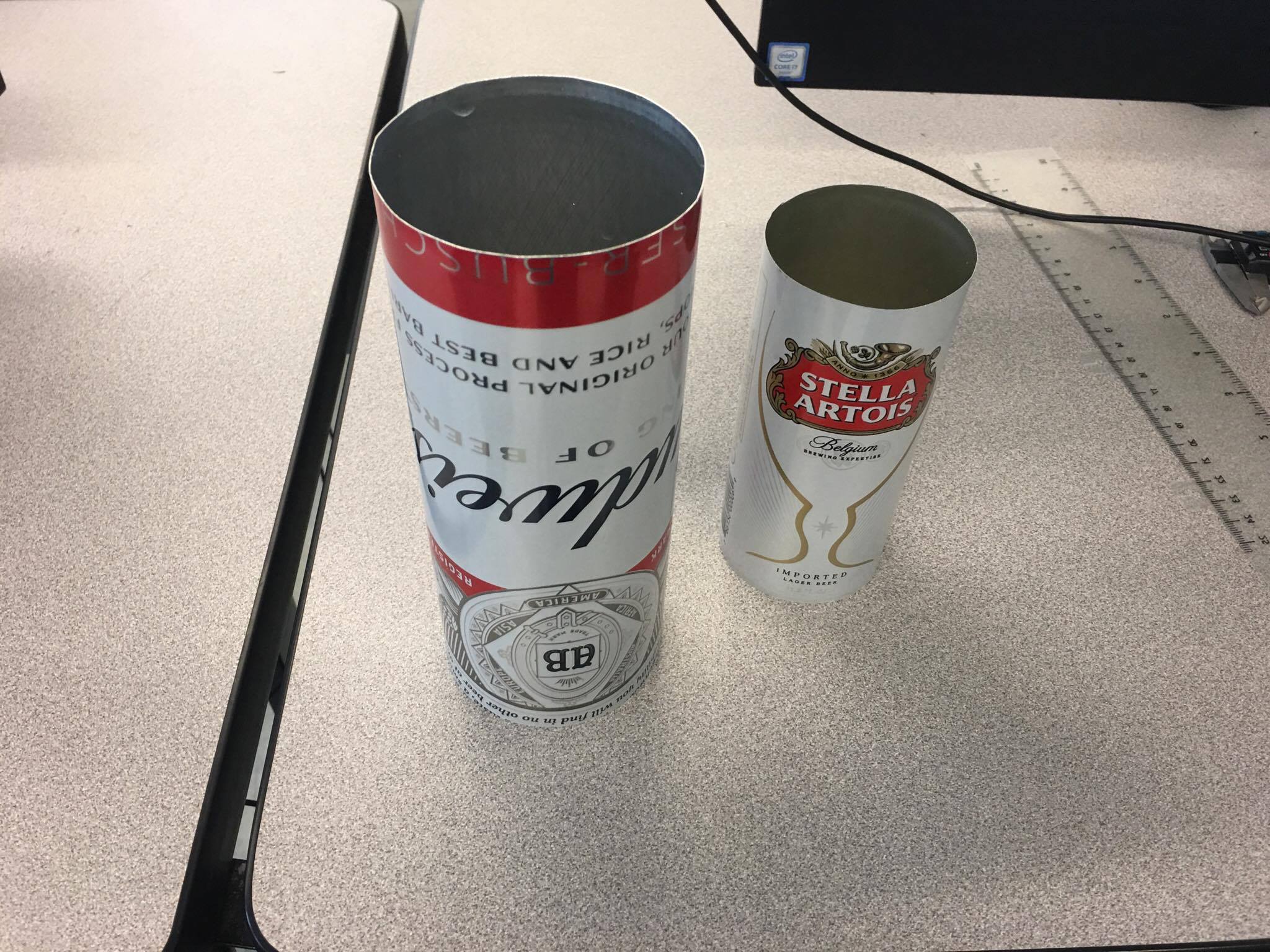


Figure 4 – Beer cans were used as airfoils due to their thin metal construction and availability on campus.

# EXPERIMENTAL SETUP

Ink Drawings
Personal Computer
Ink Drawings
Arduino UNO
Ink Drawings
Ink Drawings
Ink Drawings
Tektronix 515
Oscilloscope
Ink Drawings
Thunderbird 18
ESC
Ink Drawings
DC Power Supply
(12 V)
Ink Drawings
Ink Drawings
MEGA ACn 16/15/8
Brushless Motor
Ink Drawings
Ink Drawings

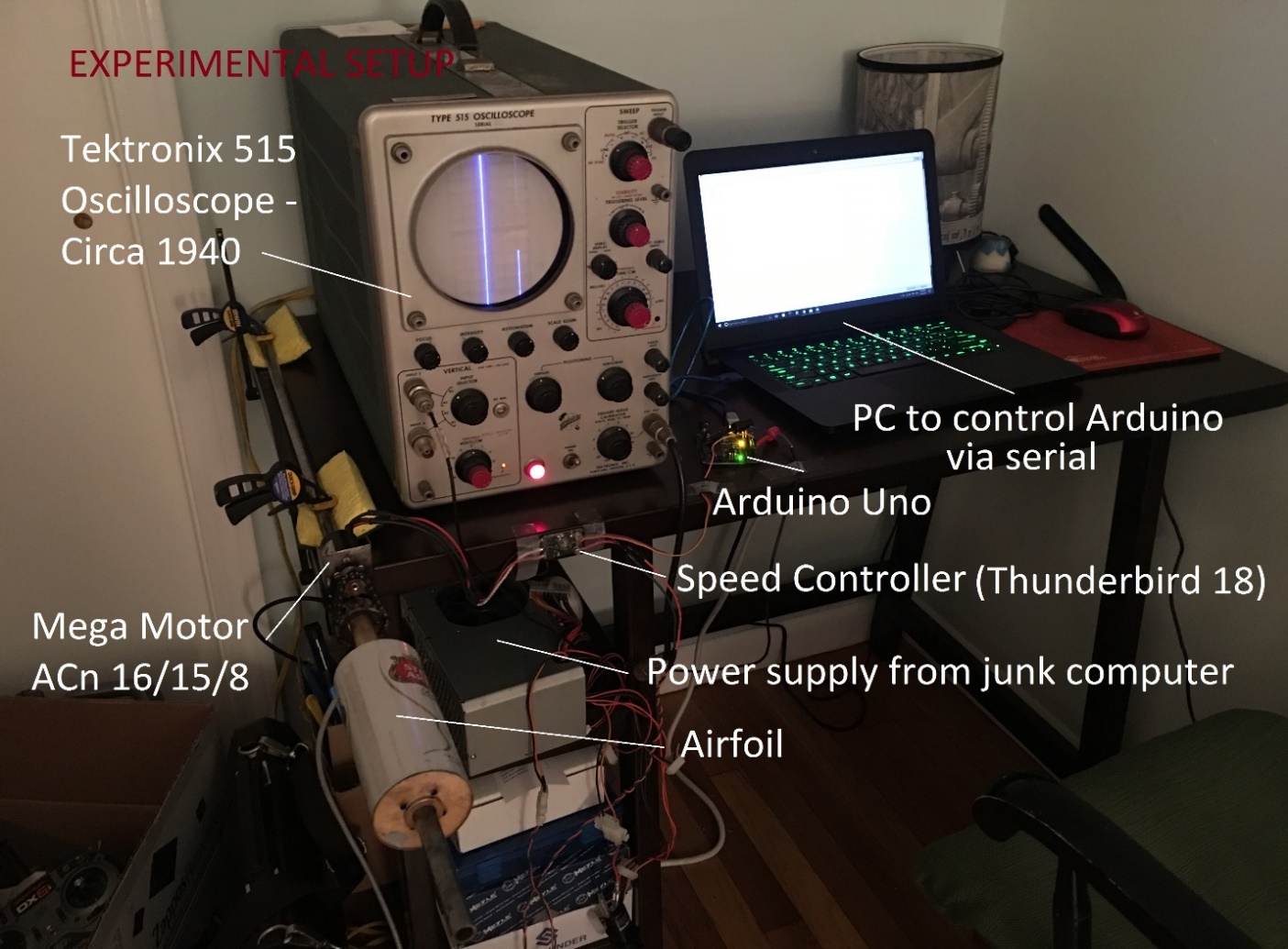
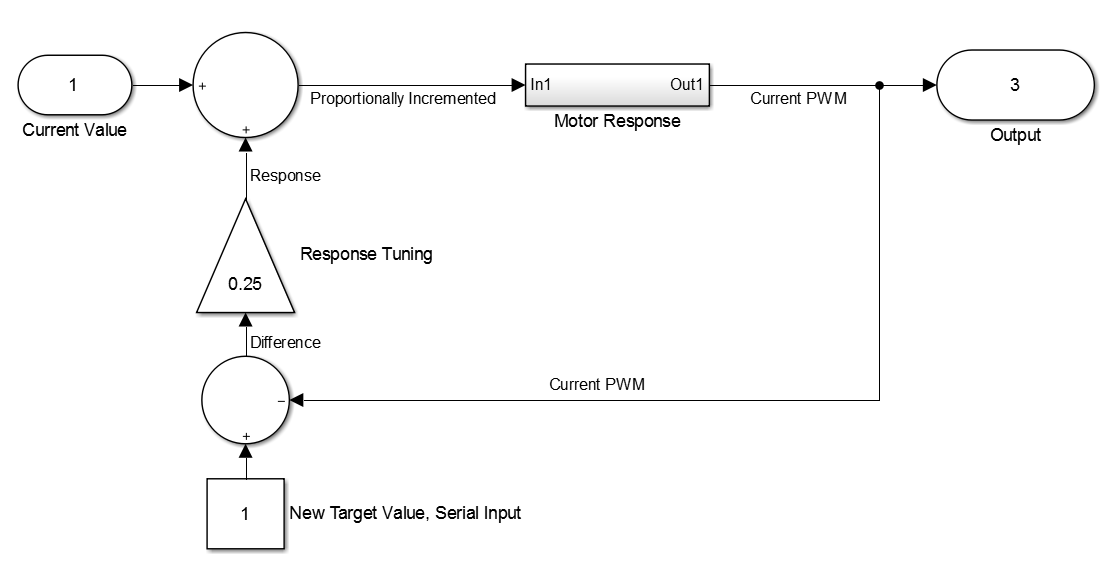



Figure 5 – Testing/manufacturing was carried out at Simon’s house due to machine tool availability. All other tools used for development are labeled in this figure.

# DESIGN CONCEPT



# SYSTEMS MODELING

# CODE

//MOTOR CONTROL PROGRAM FOR ME 670 FINAL PROJECT

//This program allows a user to input a motor power level with the serial monitor, and have the motor spin to that power

//This version allows for proportional control

//Simon Popecki

//James Skinner

//Jesse Feng

//Developed on Arduino UNO

//For: Thunderbird 18 ESC; MEGA Motor ACn 16/15/8

//This program is for temperamental electronic speed controllers that require the 50 Hz PWM signal given by hobby receivers (1-2 ms on, 18-19 ms off)

//A pulse width of 2 ms on represents full throttle, 1 ms

#include <**Servo**.h>

#include <stdint.h>

const uint8\_t SIGNAL = 3; //this is the pin on the Arduino producing the signal for the ESC (does not need to be a PWM output pin)

const uint32\_t BAUD = 9600; //rate of communication with the arduino

const uint8\_t RESPONSETUNING = 4; //this is a proportional control parameter, it adjusts how fast the motor will reach the desired speed. Higher values decrease the response speed

const uint16\_t TIMETUNING = 200; //this is another proportional control parameter, it adjusts how long the motor spins at a certain value during speed transient

uint16\_t current = 1000; //the current pulse value - zero throttle to arm the ESC - this value must be 1000 microseconds during start up, but is changed later as the program runs when the variable is used for proportional control

Servo THUNDERBIRD; //Creating a servo object for the thunderbird 18 ESC (it is controlled the same way a servo would be)

void setup()

{

THUNDERBIRD.attach(SIGNAL); //this binds the servo object to a specific pin, the servo object is used instead of a pin number from here on out

**Serial**.begin(BAUD); //start a serial communication - make sure that the serial monitor is set to a BAUD of 9600, and no line endings!

**Serial**.println("Enter a motor speed (percentage of full throttle):");

THUNDERBIRD.writeMicroseconds(current); //initializing the ESC - it won't take any commands until it sees zero throttle (1000 microseconds). This is done in setup so the user doesn't have to do it in the terminal.

}

void loop()

{

while (**Serial**.available())

{

uint16\_t target = **Serial**.parseInt(); //whatever number the user just typed in is the new motor pulse speed IN PERCENT!

**Serial**.print("New Speed: ");

**Serial**.print(target); //displaying the percentage value to the user before it is converted to a pulse width for the ESC to understand

**Serial**.print("%");

**Serial**.print('\n'); //new line for the next output

target = (target\*10)+1000; //converting percentage to a pulse width for use with theThunderbird 18 ESC

int16\_t difference = target-current; //finding the difference between the target speed (pulse length) and the current speed (pulse length) - the units are pulse width difference in microseconds

while (difference != 0)

{

int16\_t response = difference/RESPONSETUNING; //response can be positive or negative, and uses a tuning factor to generate a value to be added or subtracted from the current pulse width to get closer to the target value

current = current+response; //updating the new speed of the motor by factoring in the response modifier

THUNDERBIRD.writeMicroseconds(current);

difference = target-current;

//Serial.println(difference); //the difference between the current speed setting and target speed setting can be displayed if desired

delay(TIMETUNING);

if ((difference <= 5) && (difference >= -5)) //if the difference is small enough then just ignore it and go straight to the target value

{

THUNDERBIRD.writeMicroseconds(target); //go straight to the target pulse width before terminating the while loop

current = target;

difference = 0; //terminates the while loop and resets the difference variable for further use

}

}

}

}