Developing the Technology for a Combat Drone

New Mexico Supercomputing Challenge

Final Report

April 2, 2021

Team 24

Los Alamos High School

Team Members:

Daniel Kim

Andres Iturregui

The goal of this project was to determine and develop a reliable, long-term replacement to the job of being a combat pilot. Last year, we concluded that the idea of using a small drone with a small rocket was possible. This year we wanted to take the prior project and improve upon it by creating a higher tech drone and rocket, along with a reliable and accurate motor test stand to test the thrust of rocket engines.

This year, contrary to last, we designed the drone to be fast, powerful, and agile, with some consideration to cost. The rocket was also improved. It was eight times more powerful and considerably larger with more room for electronics. Because the FAA prohibits actually launching the rocket from the drone, we came up with multiple tests to ascertain if this concept was possible.

First of all, we built a rocket motor test stand using an Arduino and a load cell. This rig was able to measure the thrust of the rocket motor and give us a detailed, precise graph of the thrust. We compared this data with some tests run on the drone. We tested speed and acceleration, along with the ignition system and the blast deflection system.

After multiple tests, we were able to develop a combat UAV that could potentially launch an advanced missile to replace the hazardous job of being a military/combat pilot.

This project's long-term goal was to see if modern-day fighter jets could be replaced by small UAVs or unmanned aerial vehicles. After researching the death toll and causes of aviation accidents, many of these problems can be traced to the pilot. Furthermore, the cost of these aircraft is mostly put into pilot life support systems and safety. Because of this, many fighter jets are large, slow, and cost hundreds of million dollars per unit. One emerging solution is the UAV program within the Air Force. These vehicles have several advantages over fighter planes. The lack of life support systems reduced weight and cost, which allows the UAV to be cheaper, more agile and faster. Furthermore, they do not risk the life of the pilot. This project was designed to ascertain the possibility of a very small sized UAV.

Although small-sized UAVs are harder to engineer due to the size, they have several advantages. First of all, the size makes them difficult to detect on radar. Stealth is the new frontier. It allows planes to sneak into areas without being detected. Unfortunately, with bigger aircraft, the characteristics of the plane must be altered to deflect radar waves a certain way, and the trade-off is usually speed and cost. Small sized UAVs are small, which makes them naturally difficult to detect on radar. Second, the small size allows for easy deployment. Small size means that you do not need a large runway to deploy. We envision this UAV to be able to be deployed anywhere at any time. Finally, they are easy to replace. If there is a fleet of UAVs, if one gets shot down, there are many more to replace it, at the cost of one large UAV. Unfortunately, when you scale things down, the engineering becomes much more difficult, which is why we wanted to see if this idea was possible by using a series of tests.

To test this concept, without having to buy a missile, we used a drone and model rocket. The easiest way would be to strap the rocket to the drone and fire it off, but due to FAA restrictions, we were not able to do that. Instead, we devised a series of tests to see if this concept would work, without blowing someone up or breaking any laws.

The first tests would be to verify the rocket and the drone. Since the rocket is completely custom along with the drone, we needed to make sure they worked properly. The rocket is a two-stage model using the most powerful motor we could get, without needing certification. We equipped it with an electronics bay, which would hold the datalogger, and future flight computers for active stabilization. The drone is built using components used by hobbyists to build racing drones, so it is extremely fast and agile. It has an FPV, or first person view system that give the pilot a live video signal from the drone with only about 30 ms of latency. It also has a powerful flight controller with future possibilities of things such as gps and return to home functions, however we do not have those set up at the moment.

The next test was to find the effect of the rocket motor on the drone. When the rocket separates from the drone, there will be some type of force pushing against the drone. To find this, we developed a rocket motor test stand. This stand used an Arduino and special software to accurately measure the thrust of the rocket motor.

To see if the drone could handle the thrust of the rocket motor, we ran a series of tests on the drone. We used the flight controller to log acceleration data of the drone, and did a test where we would hover the drone so it is level, then do a full throttle punch out. Using Newton’s second law, we were able to find how much force the drone could handle for a short period of time based on the acceleration data. However, in the future we plan to use the rocket motor test stand mentioned previously to measure the drone’s thrust instead.

To validate the model rocket motor test stand, we were able to find some results of some similar tests performed on the exact same rocket motor. The data was almost exactly the same, with similar impulse, maximum thrust, and average thrust, with some differences due to differences within the motors caused by altitude variations, temperatures, different ages, etc.

In conclusion, the results from our model rocket motor test stand and the drone flight test, we were able to determine that it would be possible to launch this rocket off the drone we built. This is because we determined that the maximum force the drone is able to withstand for a short period of time is 49 N, whereas the rocket motor only exerted 30 N at maximum. Not to mention, the drone is not going to be hit with all 30 N of force the rocket motor exerts, meaning it would likely be even easier for the drone to handle this than the data shows.

The most significant achievement of this project was the development of the rocket motor test stand. This stand was able to use the software below to accurately measure the thrust of a rocket motor.

Special thanks to Robb Hermes for guiding us on the safety procedures for using rockets and rocket motors.

Copp, T. (2018, April 8). The death toll for rising aviation accidents: 133 troops killed in five years. Retrieved October 2, 2019, from https://www.militarytimes.com/news/your-military/2018/04/08/the-death-toll-for-rising-aviation-accidents-133-troops-killed-in-five-years/.

Opfer, C. (2018, March 8). Can drones replace fighter jets? Retrieved October 3, 2019, from https://science.howstuffworks.com/can-drones-replace-fighter-jets.htm.

“Learn More - Beyond the Basics of FPV Drone Racing.” The Drone Racing League, thedroneracingleague.com/learn-more/.

“Recreational Flyers & Modeler Community-Based Organizations.” FAA Seal, 13 Aug. 2019, www.faa.gov/uas/recreational_fliers/.

Omega. (n.d.). The UAV - The Future Of The Sky. Retrieved October 9, 2019, from https://www.theuav.com/.
Steven, H., & Joel. (2012, October 28). Will UAVs displace fighter jets soon? Retrieved October 10, 2019, from https://migflug.com/jetflights/uav-replace-fighter-jets/

“Drones and Weapons, A Dangerous Mix.” Federal Aviation Administration, 22 Aug. 2019, www.faa.gov/news/updates/?newsId=94424.

Software to measure rocket motor thrust.

```
#include "HX711.h"

#include <LiquidCrystal.h>




// HX711 circuit wiring


const int LOADCELL_DOUT_PIN = 2;

const int LOADCELL_SCK_PIN = 3;




HX711 scale;


double mtime;

double force;

double dt;

double dp;

double impulse;

double last;

double ig;
```

```
void setup() {

  Serial.begin(57600);

int done = 0;  // Declared as a global

Serial.println("Take off calibration weight!");
delay(500);
Serial.println("First column - time elasped.");
delay(1000);
Serial.println("Second column - thrust measured in N");
delay(1000);
Serial.println("Third Column - Total Impulse");
delay(500);
Serial.println("Press G and Enter to continue");
  while(done == 0)
  {
while (Serial.available() > 0)
{
if (Serial.read() == 'G')
{
done = 1;
}
}
  }
// now we clear the serial buffer.
while(Serial.available() > 0)
  {
byte dummyread = Serial.read();
  }



  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);


  double taret = 80;
  scale.tare(taret);
  scale.set_scale(215.42); // paste cal val here


  mtime = micros()/1000000.f;
  force = scale.get_units()*0.009806f;
  dt = 1/80.f;
  dp = force*dt;
  impulse = impulse + dp;

  last = mtime;

  ig = mtime + 10.f;


  pinMode(10, OUTPUT);

  digitalWrite(10, LOW);

}


void loop() {


  mtime = micros()/1000000.f;

  force = scale.get_units()*0.009806f;

  dt = mtime-last;

  dp = force*dt;

  impulse = impulse + dp;

  last = mtime;

}
```

Software to Calibrate the Motor Test Stand

```
//libraries

#include <HX711_ADC.h>

#include <EEPROM.h>

#include <LiquidCrystal.h>


//pins for LCD
```

```
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


//pins for test stand

const int HX711_dout = 6; //mcu > HX711 dout pin

const int HX711_sck = 10; //mcu > HX711 sck pin


//HX711 constructor:

HX711_ADC LoadCell(HX711_dout, HX711_sck);


const int calVal_eepromAdress = 0;

long t;


void setup() {

  Serial.begin(57600); delay(10);

  Serial.println();


  lcd.begin(16,2);

  lcd.print("Test Rig V.3.1");

  delay(2500);

  lcd.noDisplay();

  lcd.begin(16,2);

  lcd.print("Calibration21");
  LoadCell.begin();

  long stabilizingtime = 2000;

  boolean _tare = false;

  LoadCell.start(stabilizingtime, _tare);

  if (LoadCell.getTareTimeoutFlag() || LoadCell.getSignalTimeoutFlag()) {

    Serial.println("Error");

    while (1);

  }

  else {

    LoadCell.setCalFactor(1.0); // user set calibration value (float), initial value 1.0 may be used
for this sketch

    Serial.println("Startup is complete");



     lcd.noDisplay();

     lcd.begin(16,2);

     lcd.print("Startup Complete");

  }

  while (!LoadCell.update());

  calibrate(); //start calibration procedure

}


void loop() {

  static boolean newDataReady = 0;

  const int serialPrintInterval = 0; //increase value to slow down serial print activity
```

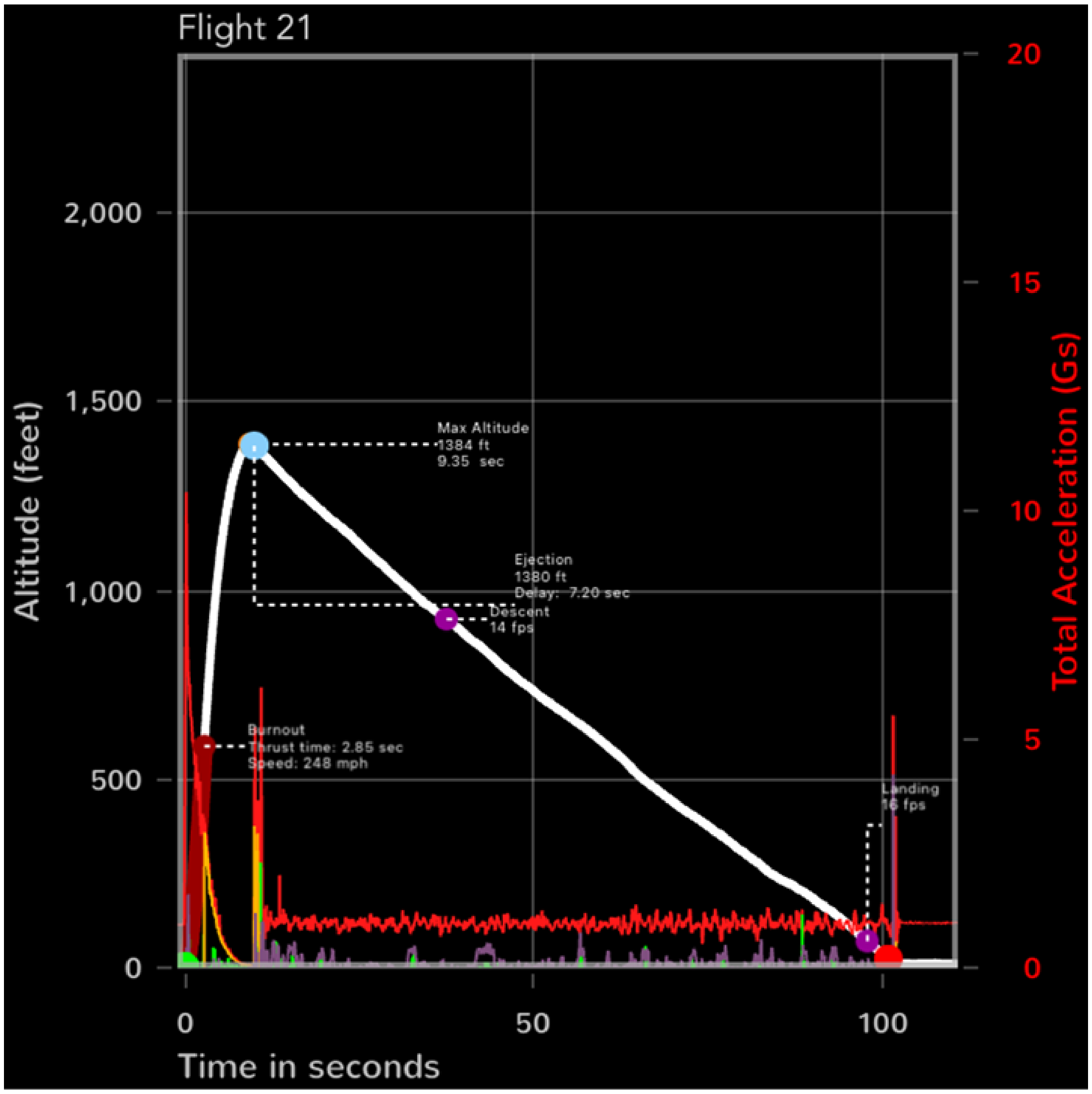
```
if (LoadCell.update()) newDataReady = true;


// get smoothed value from the dataset:

if (newDataReady) {

  if (millis() > t + serialPrintInterval) {

    float i = LoadCell.getData();

    Serial.print("Load_cell output val: ");

    Serial.println(i);

    newDataReady = 0;

    t = millis();

  }

}


// receive command from serial terminal

if (Serial.available() > 0) {

  float i;

  char inByte = Serial.read();

  if (inByte == 't') LoadCell.tareNoDelay(); //tare

  else if (inByte == 'r') calibrate(); //calibrate

  else if (inByte == 'c') changeSavedCalFactor(); //edit calibration value manually

}


// check if last tare operation is complete
```

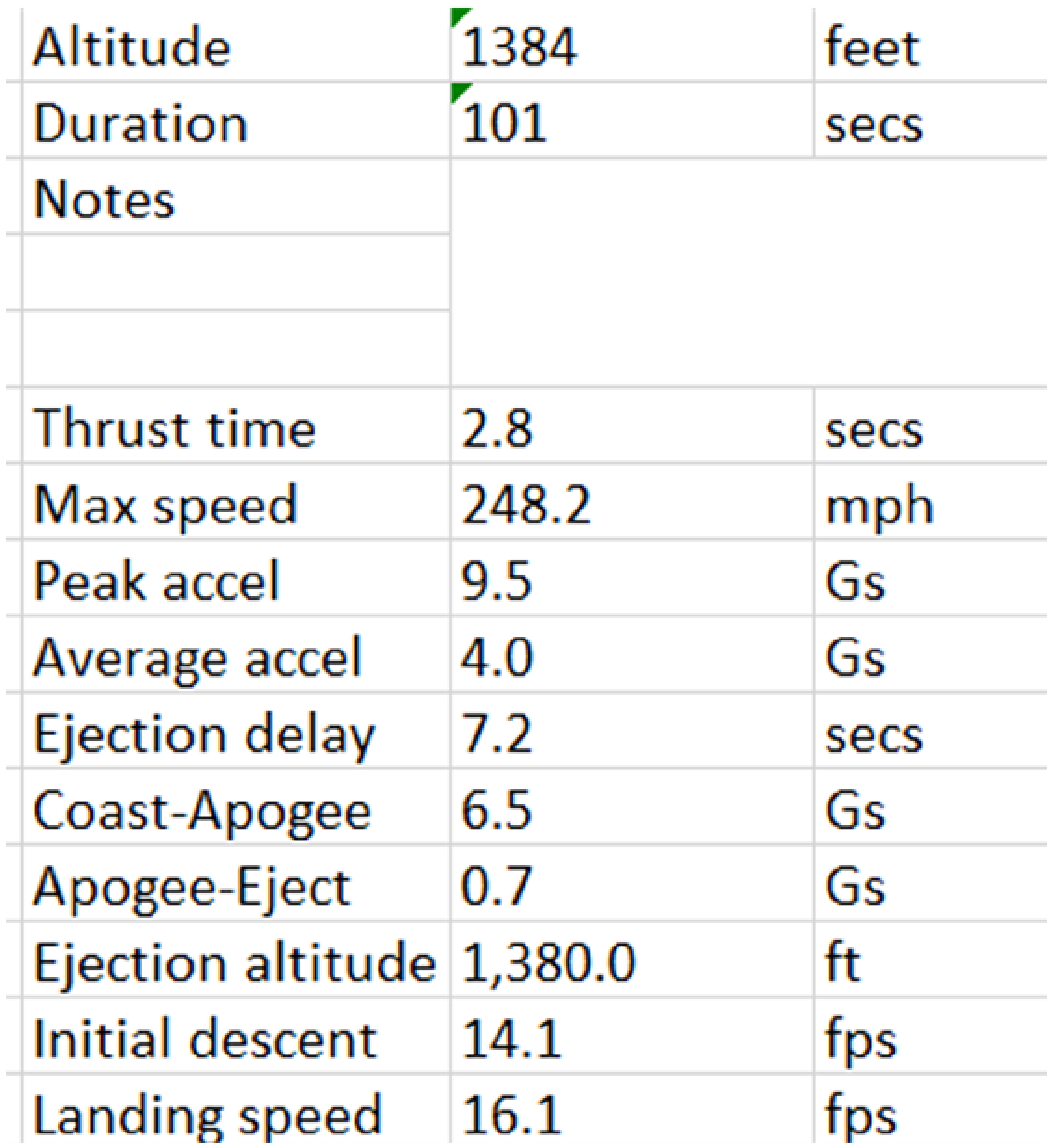
```
  if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
  }

}

void calibrate() {
  Serial.println("***");
  Serial.println("Send 't' from the serial monitor once tared.");

   lcd.noDisplay();
   lcd.begin(16,2);
   lcd.print("Tare: send t");

  boolean _resume = false;
  while (_resume == false) {
    LoadCell.update();
    if (Serial.available() > 0) {
      if (Serial.available() > 0) {
        float i;
        char inByte = Serial.read();
        if (inByte == 't') LoadCell.tareNoDelay();
      }
    }
  if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
    _resume = true;
     lcd.noDisplay();
     lcd.begin(16,2);
     lcd.print("Tare Complete");
  }
}

Serial.println("Place calibration weight");
Serial.println("Send weight in grams into serial monitor");

 lcd.noDisplay();
 lcd.begin(16,2);
 lcd.print("Place Weight on L.C");

float known_mass = 0;
_resume = false;
while (_resume == false) {
  LoadCell.update();
  if (Serial.available() > 0) {
    known_mass = Serial.parseFloat();
    if (known_mass != 0) {
      Serial.print("Known mass is: ");
```

```
        Serial.println(known_mass);

        delay(500);

        _resume = true;

         lcd.noDisplay();

         lcd.begin(16,2);

         lcd.print("Weight is...");

         delay(1000);

         lcd.noDisplay();

         lcd.begin(16,2);

         lcd.print(known_mass);

      }

    }

  }


  LoadCell.refreshDataSet(); //refresh the dataset to be sure that the known mass is measured
correct

  float newCalibrationValue = LoadCell.getNewCalibration(known_mass); //get the new
calibration value



  Serial.print("New calibration value has been set to: ");

  Serial.print(newCalibrationValue);

  lcd.noDisplay();

  lcd.begin(16,2);

  lcd.print("Cal. Val. is...");

  delay(1000);
  lcd.noDisplay();

  lcd.begin(16,2);

  lcd.print(newCalibrationValue);

  Serial.println(", use this as calibration value (calFactor) in your project sketch.");

  Serial.print("Save this value to EEPROM adress ");

  Serial.print(calVal_eepromAdress);

  Serial.println("? y/n");



  _resume = false;

  while (_resume == false) {

    if (Serial.available() > 0) {

      char inByte = Serial.read();

      if (inByte == 'y') {

#if defined(ESP8266)|| defined(ESP32)//This allows the calibration values to be stored on
onboard EEPROM or ESP32

        EEPROM.begin(512);

#endif

        EEPROM.put(calVal_eepromAdress, newCalibrationValue);

#if defined(ESP8266)|| defined(ESP32)

        EEPROM.commit();

#endif

        EEPROM.get(calVal_eepromAdress, newCalibrationValue);

        Serial.print("Value ");

        Serial.print(newCalibrationValue);

        Serial.print(" saved to EEPROM address: ");
        Serial.println(calVal_eepromAdress);
        _resume = true;


      }
      else if (inByte == 'n') {
        Serial.println("Value not saved to EEPROM");
        _resume = true;
      }
    }
  }


  Serial.println("End calibration");
  Serial.println("***");
  Serial.println("To re-calibrate, send 'r' from serial monitor.");
  Serial.println("For manual edit of the calibration value, send 'c' from serial monitor.");
  Serial.println("***");
}


void changeSavedCalFactor() {
  float oldCalibrationValue = LoadCell.getCalFactor();
  boolean _resume = false;
  Serial.println("***");
  Serial.print("Current value is: ");
  Serial.println(oldCalibrationValue);
  Serial.println("Now, send the new value from serial monitor, i.e. 696.0");
  float newCalibrationValue;
  while (_resume == false) {
    if (Serial.available() > 0) {
      newCalibrationValue = Serial.parseFloat();
      if (newCalibrationValue != 0) {
        Serial.print("New calibration value is: ");
        Serial.println(newCalibrationValue);
        LoadCell.setCalFactor(newCalibrationValue);
        _resume = true;
      }
    }
  }
  _resume = false;
  Serial.print("Save this value to EEPROM adress ");
  Serial.print(calVal_eepromAdress);
  Serial.println("? y/n");
  while (_resume == false) {
    if (Serial.available() > 0) {
      char inByte = Serial.read();
      if (inByte == 'y') {
#if defined(ESP8266)|| defined(ESP32)
        EEPROM.begin(512);
#endif
```

```
        EEPROM.put(calVal_eepromAdress, newCalibrationValue);
#if defined(ESP8266)|| defined(ESP32)
        EEPROM.commit();
#endif
        EEPROM.get(calVal_eepromAdress, newCalibrationValue);
        Serial.print("Value ");
        Serial.print(newCalibrationValue);
        Serial.print(" saved to EEPROM address: ");
        Serial.println(calVal_eepromAdress);
        _resume = true;
      }
      else if (inByte == 'n') {
        Serial.println("Value not saved to EEPROM");
        _resume = true;
      }
    }
  }
  Serial.println("End change calibration value");
  Serial.println("***");
}


  Serial.print(mtime,10);
  Serial.print("\t");
  Serial.print(force,10);
  Serial.print("\t");

  Serial.println(impulse,10);

}
```

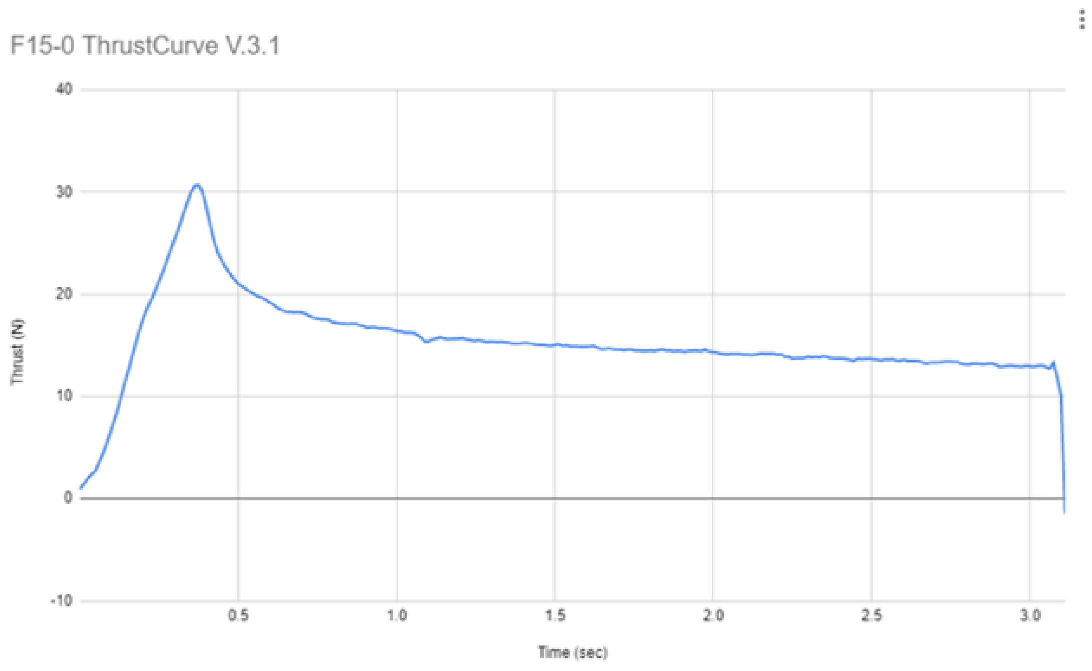
Graph 1.1 - The flight graph of the model rocket formed using acceleration and barometric data.

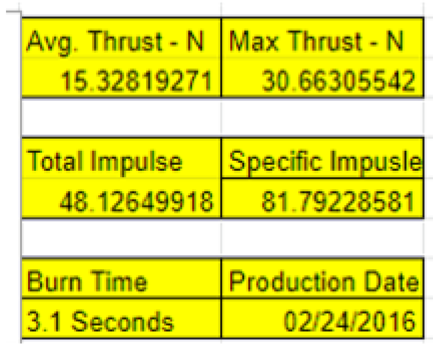




Altitude	1384	feet
Duration	101	secs
Notes		
Thrust time	2.8	secs
Max speed	248.2	mph
Peak accel	9.5	Gs
Average accel	4.0	Gs
Ejection delay	7.2	secs
Coast-Apogee	6.5	Gs
Apogee-Eject	0.7	Gs
Ejection altitude	1,380.0	ft
Initial descent	14.1	fps
Landing speed	16.1	fps

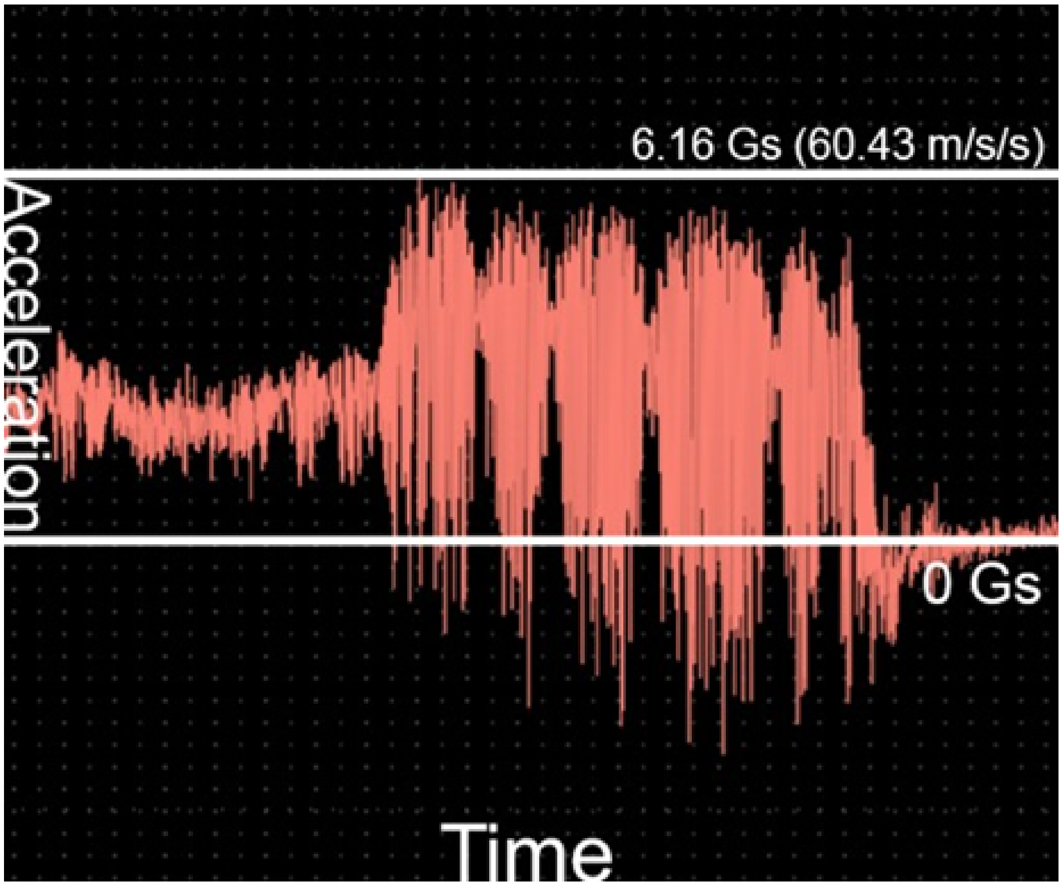
Key parts of the rocket flight gathered from the data logger.



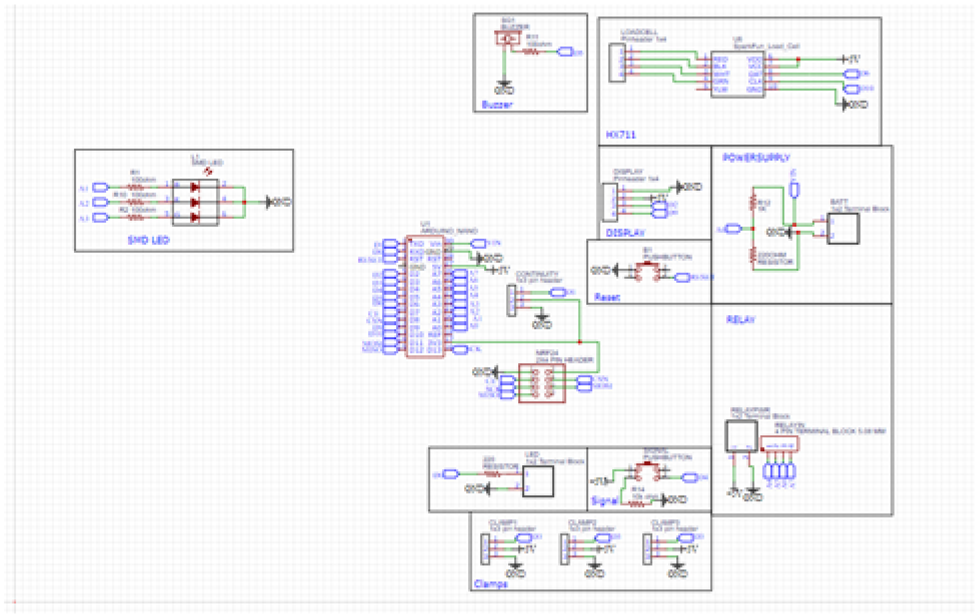


Avg. Thrust - N	Max Thrust - N
15.32819271	30.66305542
Total Impulse	Specific Impusle
48.12649918	81.79228581
Burn Time	Production Date
3.1 Seconds	02/24/2016

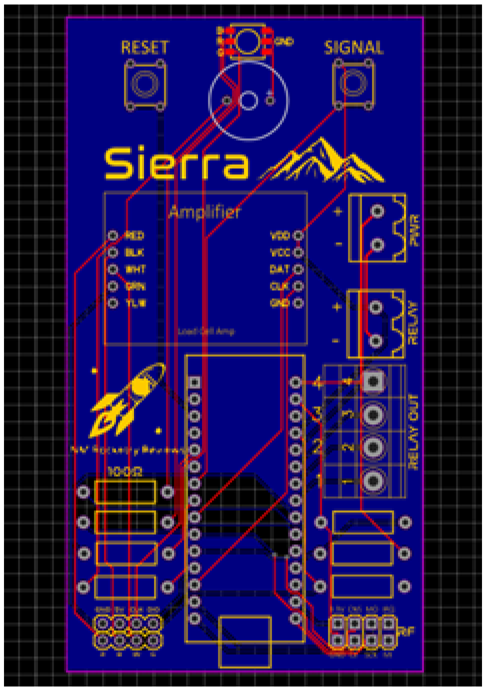
The graph formed by the rocket motor during the thrust tests along with key points of the test.

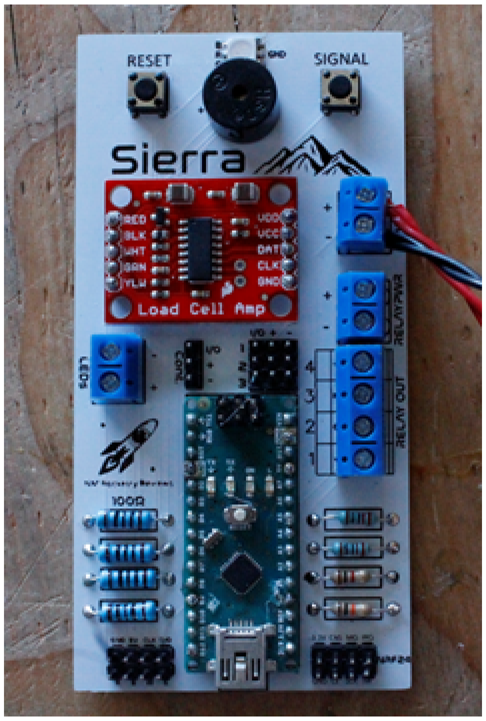


The data from the flight hardware of the drone. Using Newton's second law (F=ma) we were able to use this data to find the amount of force produced by the motors of the drone.

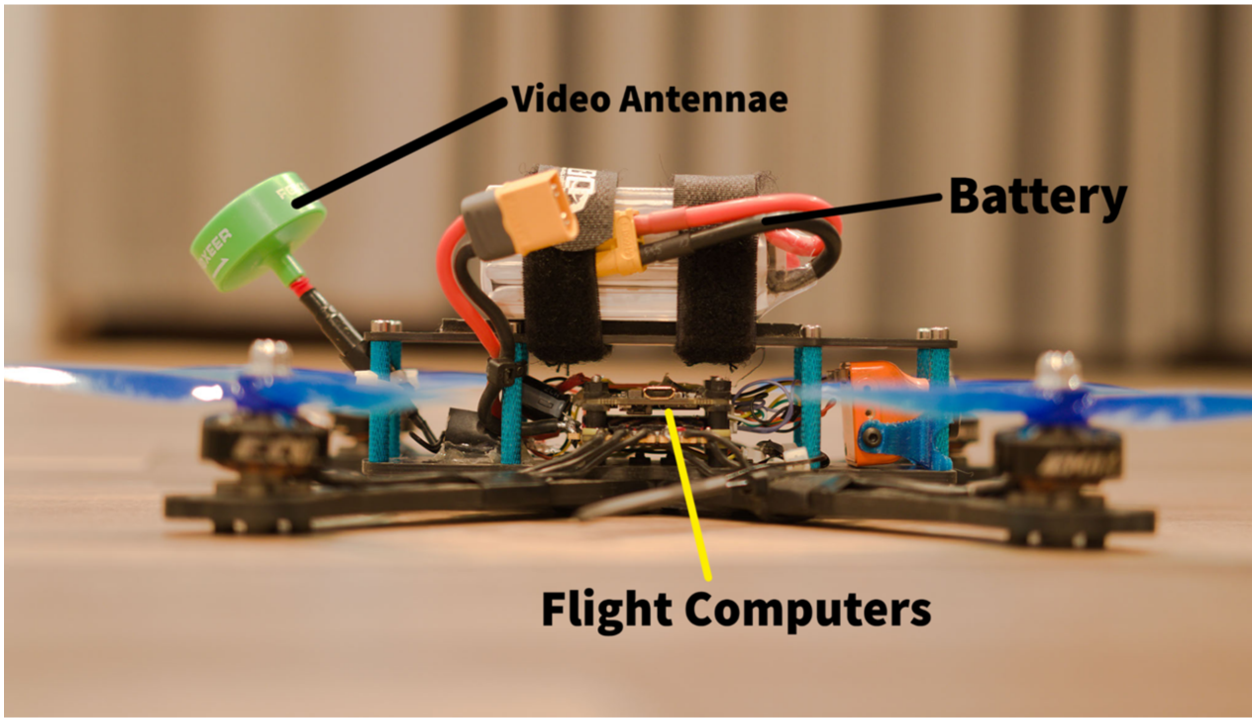


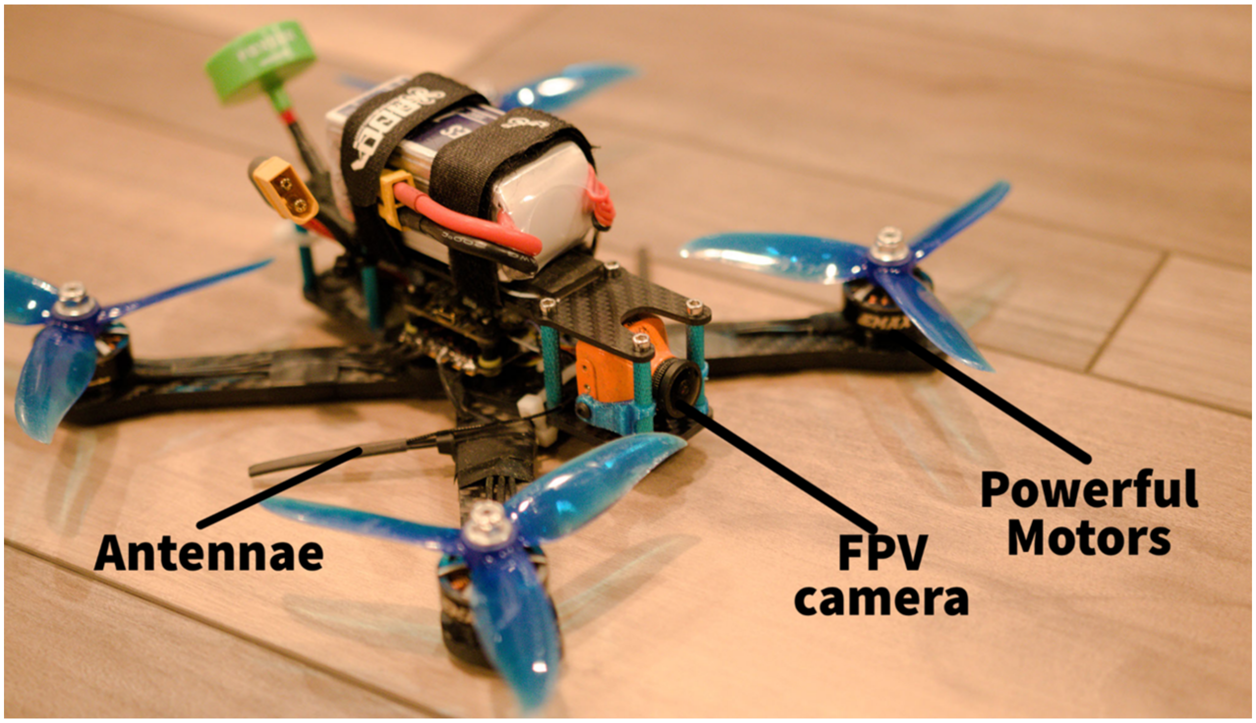
The schematic for the PCB used to measure the rocket thrust.





The designed board and the finished PCB board that we designed.





Labeled parts of the drone that was developed.