4-1-2019

AIAA Electronics/Telemetry Challenge

Walter A. Deitzler

AIAA Electronics/Telemetry Challenge

by

Walter A. Deitzler

An Honors Capstone
Submitted in partial fulfillment of the requirements
For the Honors Diploma
To

The Honors College

of

The University of Alabama in Huntsville

April 1, 2019

Honors Capstone Director: Dr. Paul Collopy

_____

Student Date

_____

Director Date

_____

Department Chair Date

_____

Honors College Dean Date

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis. In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

_____

Student Name

_____

Student Signature

_____

Date

Contents

Introduction

When driving a rover around an extraterrestrial body, the data gathered can be imperative to understanding vehicle health and increasing understanding of the planet. Telemetry is data that is collected and transmitted for the purpose of monitoring different variables, such as temperature, pressure, and humidity. A package was designed to transmit data similar to what an actual human exploration rover would want to transmit. Due to a limited budget, not everything that was wanted ended up being included, but this package is a good look into what data might be transmitted from a future manned mission to Mars. As well as sensor data, live video could be provided. On a mission to Mars, being able to see what a rover is seeing is crucial for mission control. Thus, both telemetry and video were provided.

Requirements

The requirements for this report were chosen *NASA Human Exploration Rover Guidebook, AIAA Telemetry/Electronics Award.* A screenshot of the exact page in question is shown in Appendix C. An option was given between transmitting video or providing sensor data. For whichever choice made, the decision as to why had to be backed up. There were no requirements for what sensors had to be used, or how it had to be transmitted. Along with all the data, a written report had to be presented, and an oral presentation had to be given. The project could be built by an outside company, so long as permission to use it was provided to the judges. All power had to be labeled.

Circuit Design

A. Process

To start designing the circuit, I picked a microcontroller to run my processes. The two options that were seriously considered where the Arduino Mega2560 and the Raspberry Pi. The Pi was going to be used in the case of having to use onboard video processing, but that was decided against later. Since there was not an abundance of processing power needed, and the Arduino provided familiarity, it was the processor of choice.

Next, the sensors to be used were picked. Two boards that each provided multiple sensing opportunities were chosen. First off, the dht11, a board that provided both temperature and humidity sensing. It was on hand and had many supporting documents for wiring, software and calibration. On top of that, knowing humidity and temperature when out an exploring on a foreign planet is essential. Mars has temperatures in the ranges of -125ºC to 20ºC, knowing when the temperature is outside of safe range is essential when using a human powered rover. Dangerous humidity levels could corrode vital electronics if not carefully monitored. I also used the gy-521 board, which provides accelerometer, gyroscope, and temperature data. The acceleration and rotation data that this provides is essential for a vehicle navigating an extraterrestrial environment. You need to know how fast you are going and which direction you are going. Since these calculate in 3 axes, you also get data on whether you are going up or down. The temperature sensor also provides redundancy with the dht11, and one can be used in case the other goes down. The only other sensors that I wish I had was a pressure sensor to

measure atmospheric pressure and some strain gauges to observe vehicle health, but due to some roadblocks, they were left out.

After the sensors were decided, the data had to be sent to the ground station. To do this, an ESP8266 controller, which can act as a WiFi connecter, was wired in. To get the ESP8266 connected to WiFi and transmitting data, a unique solution was thought up. Using a phone mounted in the telemetry package, I will connect the esp8266 to a mobile hotspot on the phone and use that to send it to the internet of things, IoT. Radio was considered but broadcasting video over a radio transmission is not an easy thing to do, due to broadband restrictions on video transmission. Since data will already be uploaded and download to IoT, a WiFi transmission of video makes the most economic sense.

The advantage of having the phone already mounted is that it can be used to transmit video over a streaming service such as twitch.tv or Facebook live. Being able to be dual used as both a camera and a mobile hotspot is cost effective and efficient. In this case, a OnePlus 6 is the phone in question being used, the personal phone of one of those on the telemetry team. Back at the central ground station, data will be received by a computer hooked up to a mobile hotspot. ThingSpeak is being used to receive the data, and Facebook live is being used to broadcast video.

B. Cost of Components

Item	Cost
Arduino Mega 2560	$18.99
Dht11	$5.00
GY-521 Breakout Board	$5.79
ESP8266	$6.99
Total	$36.77

Note that most of these components were already on hand before this project started. The only component that had to be purchased to complete this project was the ESP8266. Everything else was on hand.

C. Circuit Diagram

The circuit diagram was drawn using Fritzing, a program that has a large library of Arduino parts and can be used to make diagrams.

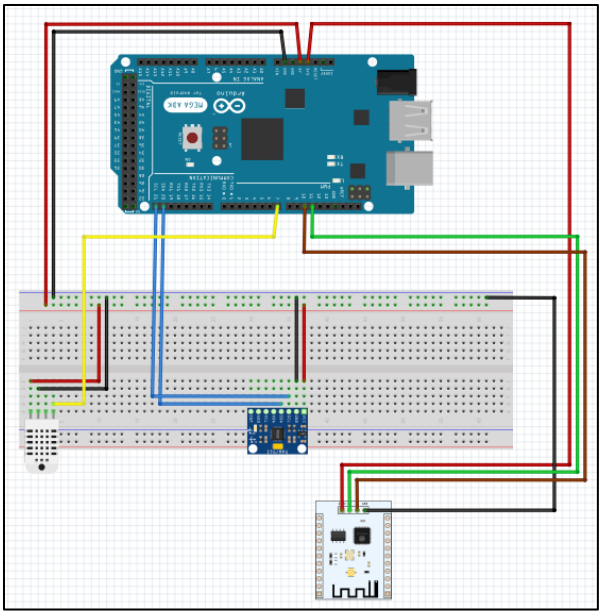


Figure 1.1 – Circuit Diagram

Software Design

A. Process

The software was written in the Arduino IDE. The libraries were based in C, as that is what I had to most familiarity with. To start, each sensor had to be working. The dht11 and the gy-521 each had their own libraries that had to be included, but they also had large amount of documentation on how to use them. The dht11 was programmed first, as it was the first sensor chosen and had a lot less data being output than the gy-521. An Arduino sketch was made to make sure that data could be obtained before I tried integrating the sensor with everything else. Once I was able to get temperature and humidity to read, I moved the code over to the main file.

After that, the gy-521 had to be programmed to have data read by the Arduino. A quick look at the datasheet for the MPU 6050 gave us the registers for the acceleration, gyroscope, and temperature data. Those were able to be read and output as usable data, below is a figure of the output to the Arduino serial monitor with all the data coming through.

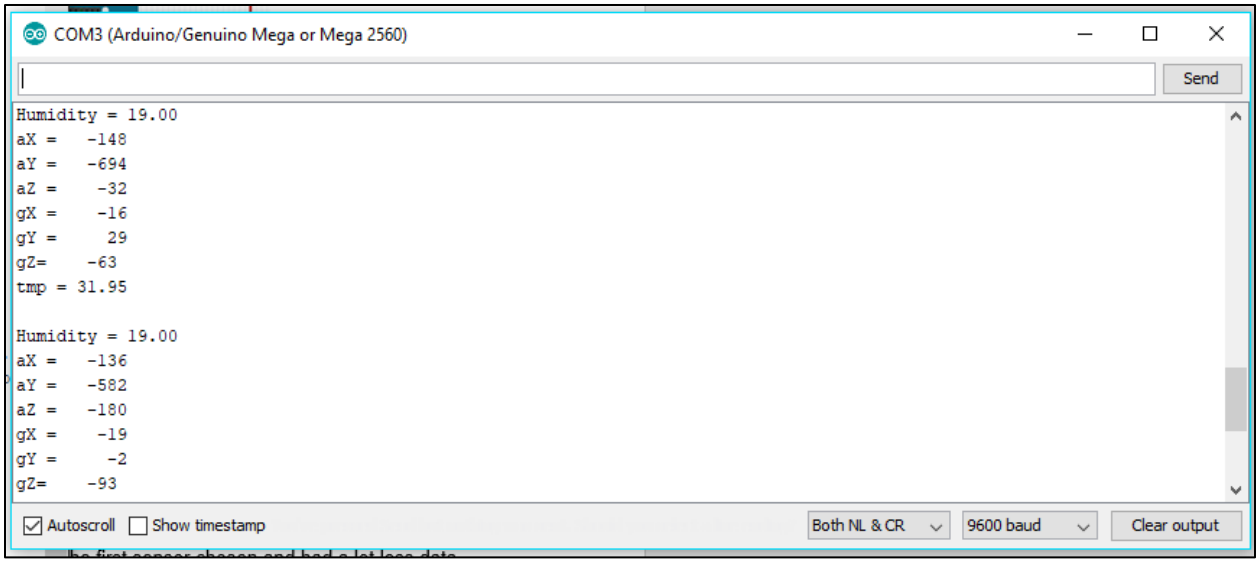


Figure 1.2 – Serial Monitor Data Output Example.

After figuring out how to make the sensors work, some calibration had to be done. This is being done by altering the data once it reads into the Arduino. This is a manual method of calibrating, but once enough data is taken, the team to narrow down what the exact offset of the sensors needs to be. During the testing that will be done between now and competition, the sensor readings from the rover sitting still before a trial run will be averaged, and those average will make up the baseline for the offset in the sensors. Temperature and humidity data will be checked against the known temperature and humidity of the environment that the rover is in, and that data will be averaged and used as the baseline for those sensors. Below is an example snippet of code for where the data is read and calibrated. The calibration numbers, the top 6 lines of the code, are subject to change as more data comes in.

```
AcXCalib = -400;
AcYCalib = 0;
AcZCalib = -14700;
GyXCalib = -2200;
GyYCalib = -200;
GyZCalib = -40;

Wire.beginTransmission(MPU_ADDR);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_ADDR, 7*2, true);


Ax = Wire.read()<<8 | Wire.read() + AcXCalib;
Ay = Wire.read()<<8 | Wire.read() + AcYCalib;
Az = Wire.read()<<8 | Wire.read() + AcZCalib;
temp2 = Wire.read()<<8 | Wire.read();
Gx = Wire.read()<<8 | Wire.read() + GyXCalib;
Gy = Wire.read()<<8 | Wire.read() + GyYCalib;
Gz = Wire.read()<<8 | Wire.read() + GyZCalib;
```

Figure1.3 – Example data read/calibration code

Once data was being read, some math had to be done to determine pitch, roll, and yaw from the gyroscope data. Raw gyroscope data does not itself give a good indicator of direction that the rover is going, but pitch and roll data does give us good information about the status of the vehicle. Yaw is less important, since the location of the buggy will already be known. Whether the buggy is in danger of flipping or rolling over is more important, which is why only the pitch and yaw data was taken. Pitch and roll calculations shown below.

```
pitch = 180 * atan (Ax/sqrt(Ay*Ay + Az*Az))/3.14159;
roll = 180 * atan (Ay/sqrt(Ax*Ax + Az*Az))/3.14159;
```

Figure1.4 – Pitch and roll calculations

Finally, the whole thing had to transmit data to the Internet of Things. This was done using thingspeak.com. After creating an account, one can upload up to 8 different fields of data at a time, and Thingspeak will compile it into graph for use in real time. Once the Arduino is connected to the internet, you can tell it to send data using api key given by thingspeak. You can choose what data you can going to what field, and what fields you want on each graph. Once the module was connected to the internet, the data can be plotted. Pair that with a live video stream, and you have all your data and video in one place. Below is an example of the layout when live.

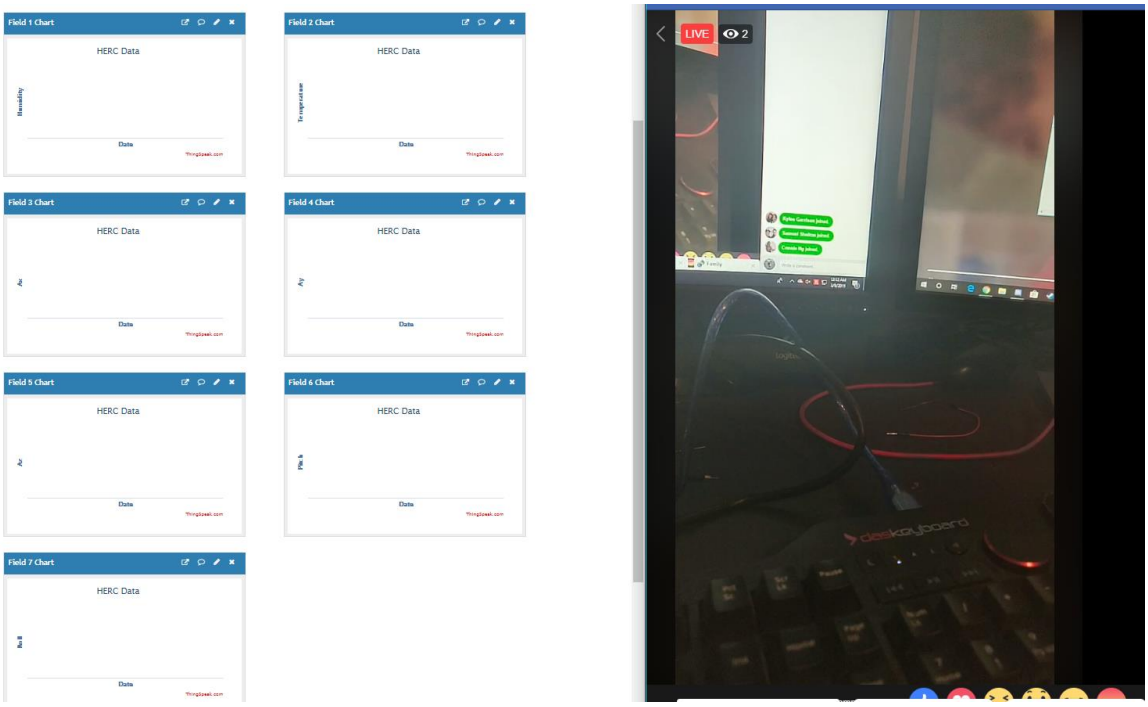


Figure 1.5 – GUI

Conclusion

While an actual manned extraterrestrial mission would have much more data than what is being transmitted here, this data is a small snapshot to show an example of what would be needed. Humidity and temperature sensors sample the atmosphere, while an accelerometer and a gyroscope keep track of the status of the buggy. Yes, pressure data would be nice to know more about the atmosphere of the planet, and a strain gauge on each of the a-arms for the wheels would be great to determine vehicle health, but on a limited budget these sensors work out just fine.

Acknowledgments

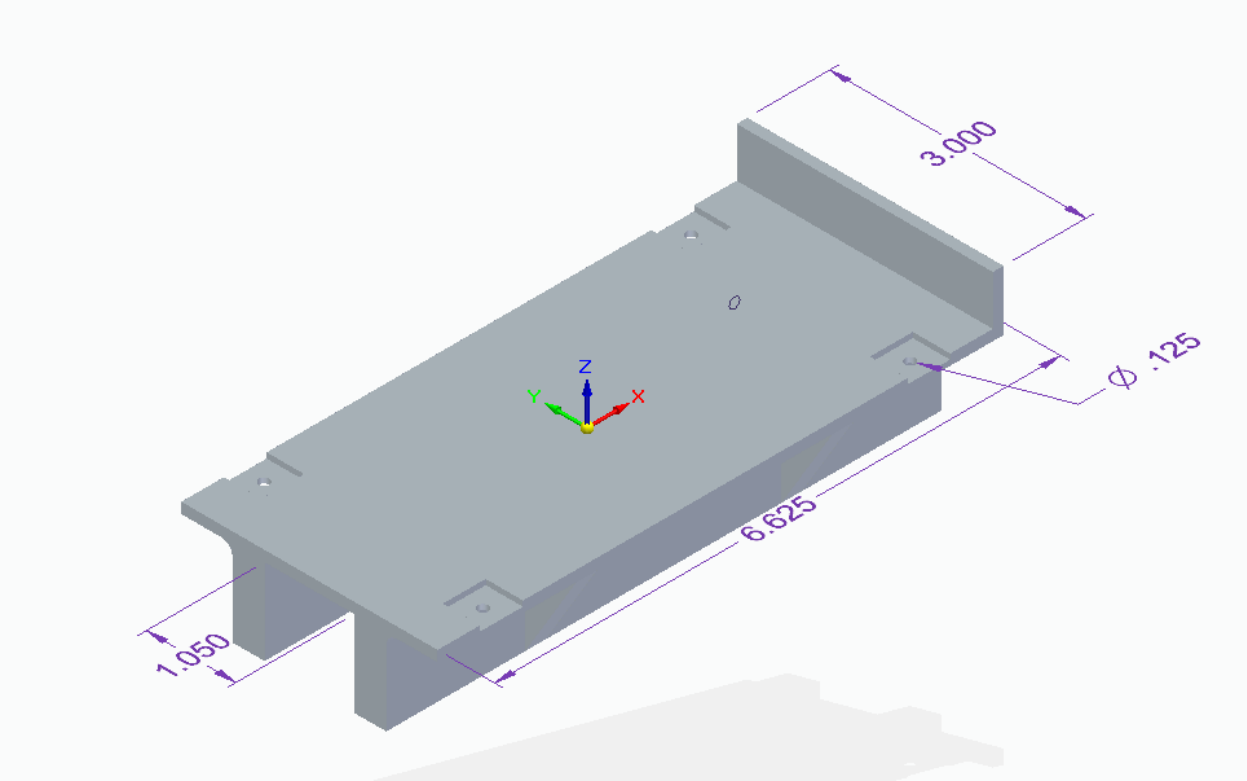
W.D. Author thanks the UAH Honors College for supporting this honors project, as well as Paul Collopy for advising.

References

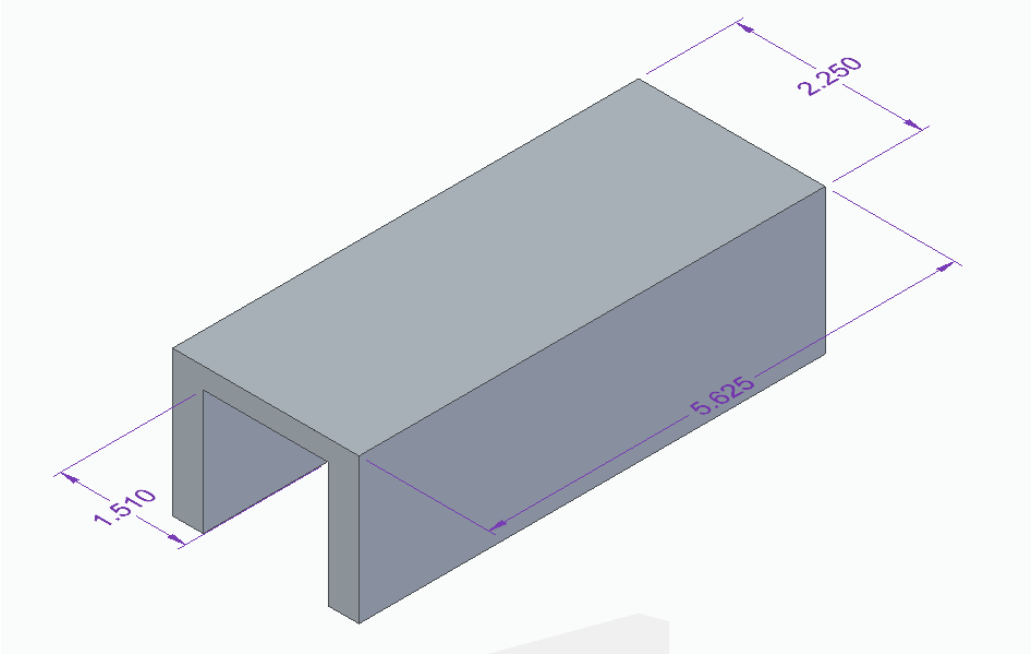
N. (2010, October 10). NASA Human Exploration Rover Challenge 2019 Guidebook. Retrieved from https://www.nasa.gov/sites/default/files/atoms/files/guide-human-exploration-rover-challenge-2019.pdf

Appendices

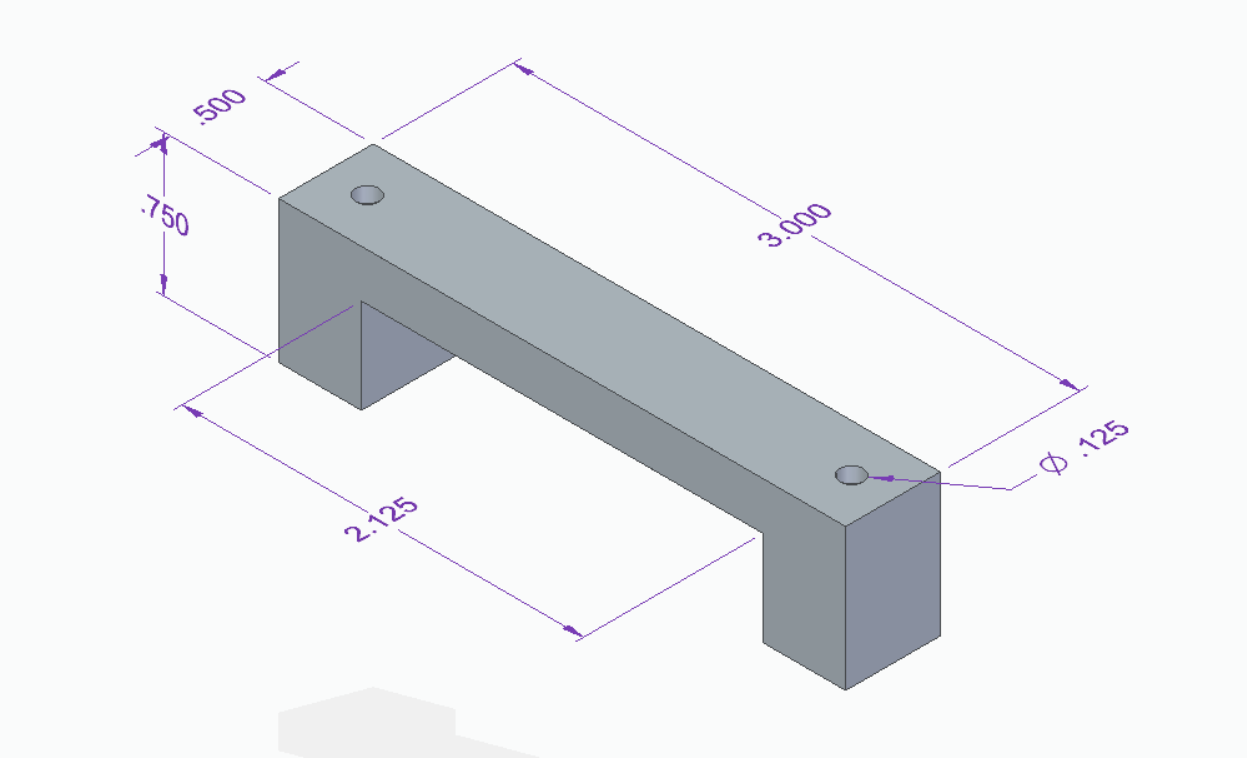
A. Restraint CAD models and photos



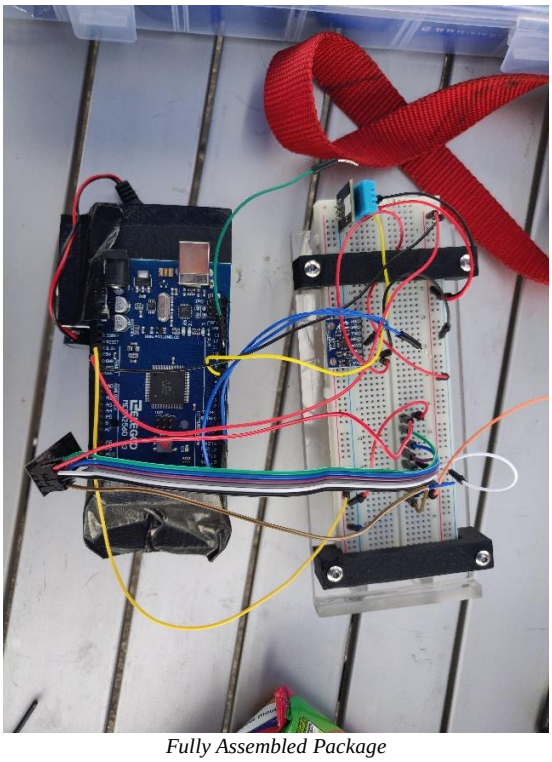
Breadboard mount, front



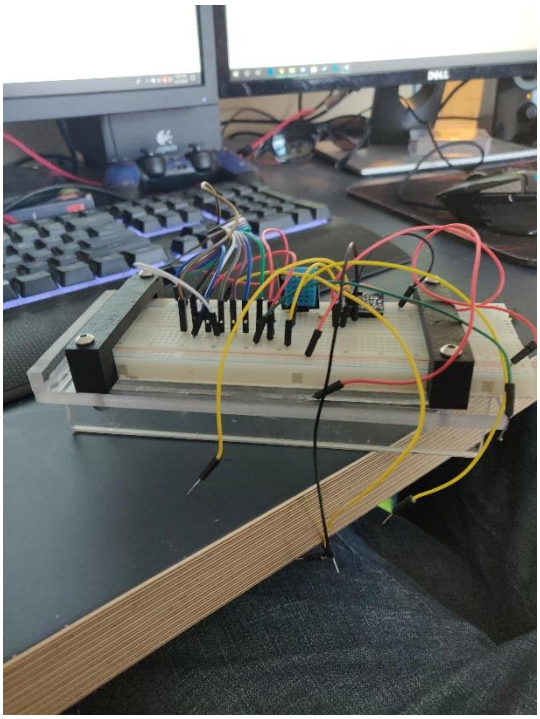
Arduino Mount



Breadboard Holder



Fully Assembled Package



Mounted Breadboard

B. Code

```
//Includes
#include "ThingSpeak.h"
#include "Wire.h" // This library allows you to communicate with I2C devices.
#include "math.h"
#include <SoftwareSerial.h>

const int MPU_ADDR = 0x68; // I2C address of the MPU-6050. If AD0 pin is set to HIGH, the I2C address will be 0x69.
int16_t Ax, Ay, Az; // variables for accelerometer raw data
int16_t Gx, Gy, Gz; // variables for gyro raw data
int16_t temp1, temp2; // variables for gy562temperature data
char tmp_str[7]; // temporary variable used in convert function
int AcXCalib, AcYCalib, AcZCalib, GyXCalib, GyYCalib, GyZCalib;
int16_t pitch,roll;
char* convert_int16_to_str(int16_t i) { // converts int16 to string. Moreover, resulting strings will have the same length in the debug monitor.
  sprintf(tmp_str, "%6d", i);
  return tmp_str;
}

#include <dht.h>

dht DHT;

#define DHT11_PIN 7
#define RX 10
#define TX 11

String AP = "WDOP6";            //Hotspot Name
String PASS = "Moonbuggy12345";  //Hotspot Password

String API = "U7BE6VU1QMN33PPQ"; //Thingspeak API
String HOST = "api.thingspeak.com";
String PORT = "80";
String field = "field1";
int countTrueCommand;
int countTimeCommand;
boolean found = false;
int valSensor = 1;
SoftwareSerial esp8266(RX,TX);

void setup() {
  Serial.begin(9600);
  esp8266.begin(115200);
  sendCommand("AT",5,"OK");
  sendCommand("AT+CWMODE=1",5,"OK");
  sendCommand("AT+CWJAP=\""+ AP +"\",\""+ PASS +"\"",20,"OK");
  Wire.begin();
```

```
  Wire.beginTransmission(MPU_ADDR); // Begins a transmission to the GY-521 board
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0);    // Wakes up the MPU-6050
  Wire.endTransmission(true);
}
void loop() {


 String getData = "GET /update?api_key="+ API +"&"+ field +"="+String(temp2);
 sendCommand("AT+CIPMUX=1",5,"OK");
 sendCommand("AT+CIPSTART=0,\"TCP\",\""+ HOST +"\","+ PORT,15,"OK");
 sendCommand("AT+CIPSEND=0," +String(getData.length()+4),4,">");
 esp8266.println(getData);delay(1500);countTrueCommand++;
 sendCommand("AT+CIPCLOSE=0",5,"OK");

int getSensorData() {
  return random(1000); // Replace with
}
  temp1=(DHT.temperature);
  int chk = DHT.read11(DHT11_PIN);
 // Serial.print("Temperature = ");
 // Serial.println(temp1);
  Serial.print("Humidity = ");
  Serial.println(DHT.humidity);
  //Calibration for Accelerometer/Gyro
  Ax = 0;
  Ay = 0;
  Az = 0;
  Gx = 0;
  Gy = 0;
  Gz = 0;
  AcXCalib = 252;
  AcYCalib = -1268;
  AcZCalib = -16400;
  GyXCalib = -543;
  GyYCalib = -51;
  GyZCalib = 7;

  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);                                  // starting with register 0x3B (ACCEL_XOUT_H)
[MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2, p.40]
  Wire.endTransmission(false);                       // the parameter indicates that the Arduino will
send a restart. As a result, the connection is kept active.
  Wire.requestFrom(MPU_ADDR, 7*2, true);             // request a total of 7*2=14 registers

                                                     // "Wire.read()<<8 | Wire.read();" means two registers are
read and stored in the same variable
  Ax = Wire.read()<<8 | Wire.read() + AcXCalib;      // reading registers: 0x3B (ACCEL_XOUT_H)
and 0x3C (ACCEL_XOUT_L)
  Ay = Wire.read()<<8 | Wire.read() + AcYCalib;      // reading registers: 0x3D (ACCEL_YOUT_H)
and 0x3E (ACCEL_YOUT_L)
```

```
  Az = Wire.read()<<8 | Wire.read() + AcZCalib;              // reading registers: 0x3F (ACCEL_ZOUT_H)
and 0x40 (ACCEL_ZOUT_L)
  temp2 = Wire.read()<<8 | Wire.read();                      // reading registers: 0x41 (TEMP_OUT_H)
and 0x42 (TEMP_OUT_L)
  Gx = Wire.read()<<8 | Wire.read() + GyXCalib;               // reading registers: 0x43
(GYRO_XOUT_H) and 0x44 (GYRO_XOUT_L)
  Gy = Wire.read()<<8 | Wire.read() + GyYCalib;               // reading registers: 0x45
(GYRO_YOUT_H) and 0x46 (GYRO_YOUT_L)
  Gz = Wire.read()<<8 | Wire.read() + GyZCalib;               // reading registers: 0x47
(GYRO_ZOUT_H) and 0x48 (GYRO_ZOUT_L)

  // print out data
  Serial.print("aX = "); Serial.println(convert_int16_to_str(Ax));
  Serial.print( "aY = "); Serial.println(convert_int16_to_str(Ay));
  Serial.print( "aZ = "); Serial.println(convert_int16_to_str(Az));
  // the following equation was taken from the documentation [MPU-6000/MPU-6050 Register Map and
Description, p.30]
  Serial.print( "gX = "); Serial.println(convert_int16_to_str(Gx));
  Serial.print( "gY = "); Serial.println(convert_int16_to_str(Gy));
  Serial.print( "gZ= "); Serial.println(convert_int16_to_str(Gz));
  Serial.print( "tmp = "); Serial.println(((temp1+temp2)/2)/340.00+36.53);
  Serial.println();
  // delay

  pitch = 180 * atan (Ax/sqrt(Ay*Ay + Az*Az))/3.14159;
  roll = 180 * atan (Ay/sqrt(Ax*Ax + Az*Az))/3.14159;
  Serial.print("pitch = "); Serial.println(pitch);
  Serial.print("roll = "); Serial.println(roll);
  delay(3000);
}
  void sendCommand(String command, int maxTime, char readReplay[]) {
  Serial.print(countTrueCommand);
  Serial.print(". at command => ");
  Serial.print(command);
  Serial.print(" ");
  while(countTimeCommand < (maxTime*1))
  {
    esp8266.println(command);//at+cipsend
    if(esp8266.find(readReplay))//ok
    {
      found = true;
      break;
    }

    countTimeCommand++;
  }

  if(found == true)
  {
    Serial.println("OYI");
    countTrueCommand++;
```

```
  countTimeCommand = 0;
 }

 if(found == false)
 {
  Serial.println("Fail");
  countTrueCommand = 0;
  countTimeCommand = 0;
 }

 found = false;
}
```

C. **Requirements Document**

AIAA Telemetry/Electronics Award (Optional)

The Greater Huntsville Section of the AIAA will present an award titled "AIAA Telemetry," which recognizes the development and operation of the most innovative and useful real-time telemetry system at the Rover Challenge.

Submission: Written report (no more than five pages, not including cover page) and oral presentation(no more than five minutes, not including judges' questions and answers)

Deadline: March 8, 2019 (written report); oral presentation at the Rover Challenge

Method of Submission: Email with PDF attachment of written report sent to distribution@hsv-aiaa.org *Please include the award name as the subject line.*

The telemetry system must do at least one of the following two options:

Transmit Real-Time Video

- Video must be broadcast and received by a central station, and video image must be available for viewing in real time by your team.
- Teams may use on-board data logging (recording) for later review.
- Report and presentation must justify the system design choices made by your team relative to weight, usefulness to crew, function, innovation and redundancy.

Transmit Real-Time Sensor Data

- Sensor data must be broadcast and received by a central station, and the raw data must be available for viewing in real time by your team.
- Teams may use on-board data logging (recording) for later review.
- A baseline for sensor data must be established prior to the excursion. Explain how your team will accomplish this.
- Report and presentation must justify the system design choices made by your team relative to weight, usefulness to crew, function, innovation and redundancy.

Additional notes:

- The report must include a cover page with the words "AIAA Telemetry Award" and the team name.
- The telemetry package may be designed and constructed by someone external to your team. If this method is used, your team must obtain permission to integrate the package from the developer. A PDF version of this permission must be included in your written report by the report submission deadline. This additional documentation will not count against your page limit.
- Telemetry systems that fail during operation on the course still qualify for award consideration.
- Functioning power system batteries and transmission antennae must be appropriately labeled as defined by the Rover Challenge rules.

- Teams who submit their written report after the deadline will not be considered for the AIAA Telemetry/Electronics Award.
- Judges are able to exclude a team for consideration of an award if the team's written report exceeds the permitted number of pages and/or if the oral presentation exceeds the time allowed.

Questions should be directed to: distribution@hsv-aiaa.org