FINEST SELECTION

1

AMPAD

Gold Fibre®

Computation Book

NAME Nate Poole + Wesley Cooke

COURSE Internship/Research Summer 2022 Book 1
Summer 2023

Numbered pages to track information

Fine quality heavyweight paper

11¾" x 9¼"
Ruled 4 squares/inch
76 Numbered Sheets

REV4

TOPS PRODUCTS

22-157

©2015 TOPS Products. 1001 Rialto Road, Covington, TN 38019 USA
Made in USA. • Ampad, Gold Fibre and Finest Selection are trademarks of TOPS Products. • Tops-Products.com/Ampad

0 74319 22157 9

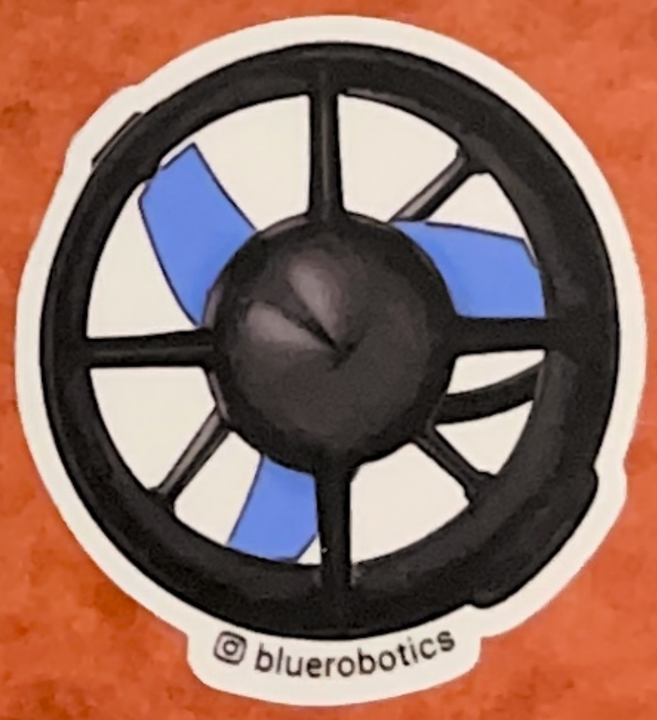


Table of Contents

Date	Project name	Page
July 12, 2022	MKR 1310 Board	6
July 12, 2022	MKR1310 Robotics	10
May 23, 2023	BNO055 Testing	13
May 26, 2023	DFRobot GNSS	23
June 8, 2023	Standalone GNSS Data Logger DFRobot	31
June 14, 2023	Level Shifting MOSFET	39
June 19, 2023	Programming ALS31313 I2C Addr	51
June 21, 2023	Reyax 8839RYS Parsing+logging	59
June 26, 2023	SparkFun ZED-F9P Logging	69
June 28, 2023	Setting up an RTK Base station	73
July 30, 2023	BNO055 Relative heading observation	87
July 30, 2023	Running the T200 underwater Motor	93
July 30, 2023	Autonomous WIL	97
Dec 2023	Writing an H-Bridge Library	117
Dec 2023	Writing an ESC Library	123
Dec 2023	Writing a GPS Library	127
Dec 2023	Writing a BNO Library	131
Dec 2023	Writing a Heading Controller Library	135
Dec 2023	Fixing the Arcs	139

Table Of Contents Continued

Date	Project Name	Page
Dec 2023	writing a controller Library	143
Dec 2023	putting it all Together - Land	145
Dec 2023	putting it All Together - Water	148

Nate Poole

July 12, 2022

Arduino MKR 1310 Board

- Overview
  - LORa connectivity w/ low power
  - Can connect to
    - + Arduino Iot Cloud
    - + LoRa network using Arduino LoRa PRO Gateway
    - + other boards
- Main Features
  - Cortex-M0 32-bit SAMD21
    - + low power processor
  - Murata CMWX1ZZABZ
    - + LoRa wirelss to connect to own network
    - + Connect to various LoRaWAN networks
  - ATECC508 crypto chip
    - + Keeps data secure and private
- Tech Specs
  - Operating Voltage : 3.3V
    - → Note: VIN runs off of 5V (A little weird)
  - Communication
    - + UART
    - + I2C
    - + SPI

Nate Poole

MKR 1310 Robotics

July 12, 2022

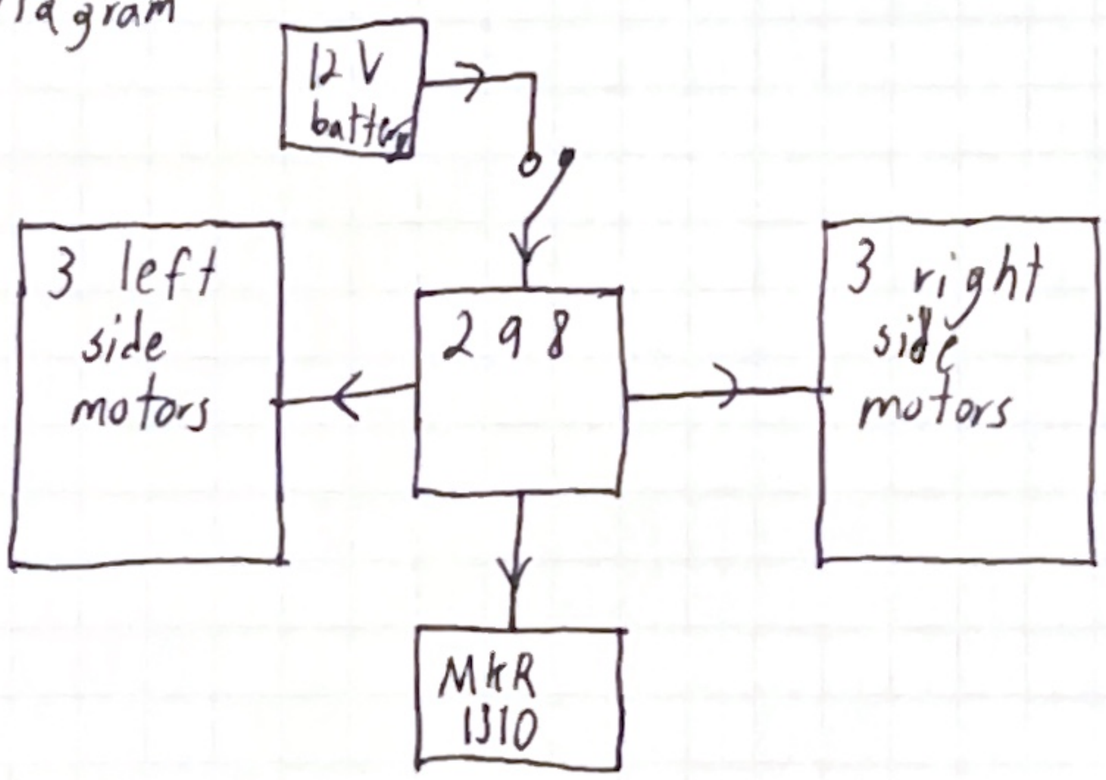
- Goal
  - To make a ~~self navagating~~ obstacle avoiding, GPS-navigated robot to collect samples in the future.

- ~~Components~~

  Project 1 - Movement

- Components
  - MKR 1310 board
  - LM 298 motor controller
  - PiBorg structure
  - 12V battery
  - 6 motors

- Diagram



- ~~R~~ Physical Connections
  - 2 wires from battery to LM298
    + red for Voltage
    + black for Ground
  - 4 wires from LM298 to motors (2 for each side)
    + red and back again

Nate Poole

July 12, 2022

- Physical Connections
  - LM298 to MKR 1310 6 wires
    - + 2 Enable pins
      - ↳ must be PWM, we use analog write command
    - + 2 for right side motors
      - ↳ digital pins, we use digitalwrite
    - + 2 for left side motors
      - ↳ digital pins

- Code

Nate Poole

Project 2: Getting a Heading and turning

BNO055 Testing

Overview

9-axis absolute orientation sensor
3-axis accelerometer
3-axis gyroscope
3-axis magnetometer
Capable of providing sensor fused data for euler angles and quaternions

Libraries

1. <Adafruit_Sensor.h>
2. <Adafruit_BNO055.h>
3. <utility/imumaths.h>

Library 3 should be defaultly installed with arduino.
To install Library 2, go to Tools → Manage Libraries.
Search "Adafruit BNO055". Click instal.
This should prompt you to install the other dependencies.

Example Code

More information at the following link:
learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/arduino-code

Additionally, video tutorials available from Top Tech Boy
toptechboy.com/arduino-based-9-axis_inertial-measurement-unit-imu-based-on-bno055-sensor/

- Instantiation

```
Adafruit_BNO055 myIMU = Adafruit_BNO055();
```

- In set up

```
myIMU.begin();
myIMU.setExternalCrystalUse(true);
```

BNO055 Testing

05/23/23
Wesley Cecko

- Getting a vector with 3 components.

$$\underbrace{\text{imu::Vector<3>}}_{1}\ \underbrace{\text{acc}}_{2} = \underbrace{\text{myIMU.getVector}}_{3}(\underbrace{\text{Adafruit_BNO055::Vector_Accelerometer}}_{4})$$

1. Data type of the variable. This is similar to int or float.

2. Variable Name. This name could be anything you chose. Make it descriptive.

3. This is the method to call.

4. This is the specific Vector that you want.

<u>Available Vectors</u>

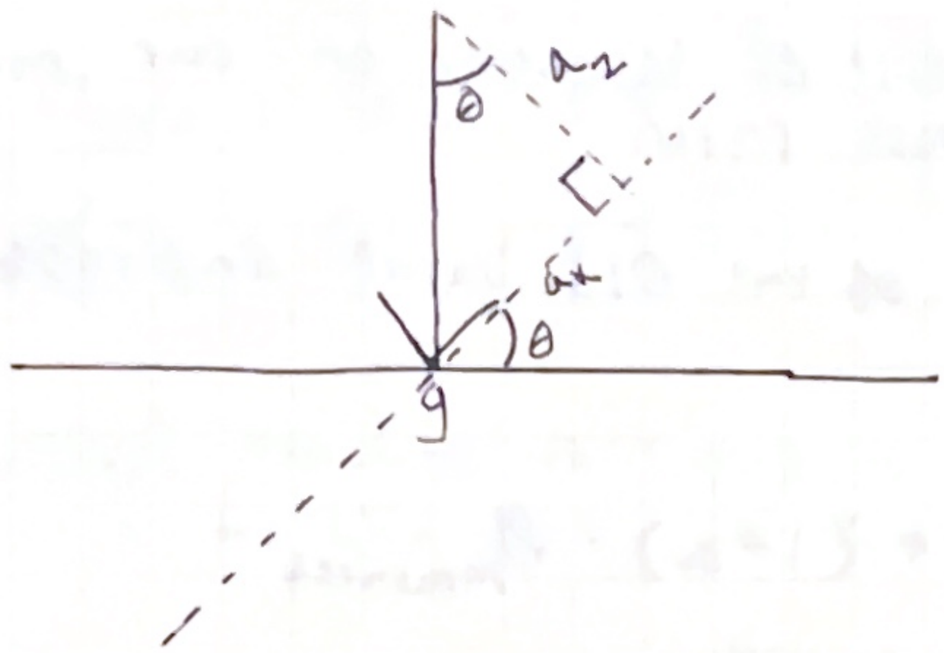
Adafruit_BNO055::VECTOR_MAGNETOMETER
Adafruit_BNO055::VECTOR_GYROSCOPE
Adafruit_BNO055::VECTOR_ACCELEROMETER
Adafruit_BNO055::VECTOR_EULER
Adafruit_BNO055::VECTOR_LINEARACCEL
Adafruit_BNO055::VECTOR_GRAVITY

<u>Quaternion Data</u>

```
imu::Quaternion quat = myIMU.getQuat();
```

Vectors have .X(), .Y(), and .Z() methods.
Quaternions have .X(), .Y(), .Z(), and .W() methods.

Measuring Pitch and Roll from Accelerometer



Measured Pitch

$$\theta_m = \tan^{-1}\left(\frac{a_x}{a_z}\right)$$

Measured Roll

$$\phi_M = \tan^{-1}\left(\frac{a_y}{a_z}\right)$$

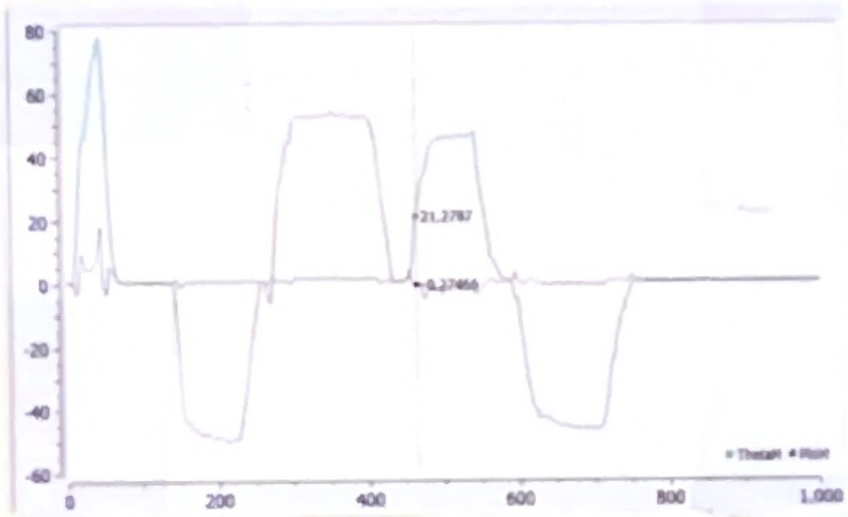


Figure 1

Testing our pitch and roll measurements. This is acomplished by rotating the bread board about the x any y axies. The measurements are good based on eyeball judgement.

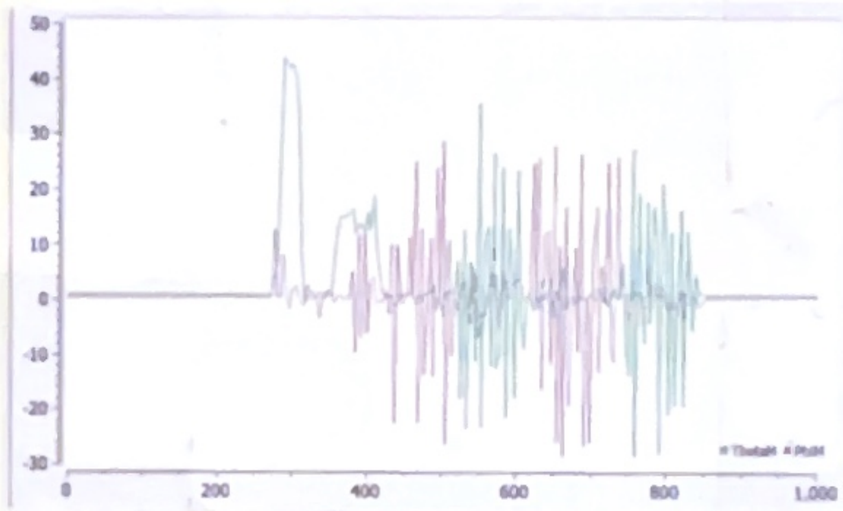


Figure 2

Testing our pitch and roll measurements when vibration is introduced. Notice that we didn't move the pitch and roll from 0. However, there were still changes due to the vibration.

BNO055 Testing

05/23/23
Wesley Cooka

Low Pass Filter on Pitch and roll

To combat the effects of Vibration on our measurements, we can use a low pass filter.

We will take 90% of the Old Value and 10% of the New.

$a = 0.9$

$$\theta_{\substack{Filtered\\New}} = a \cdot \theta_{\substack{Filtered\\Old}} + (1-a) \cdot \theta_{Measured}$$

$$\phi_{\substack{Filtered\\New}} = a \cdot \phi_{\substack{Filtered\\Old}} + (1-a) \cdot \phi_{measured}$$

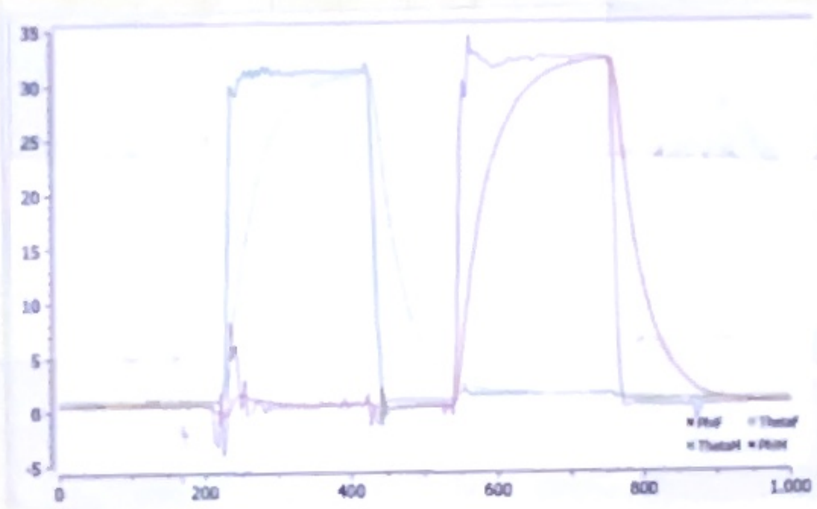


Figure 3

Showing the effects of the low pass filter on our measurements.

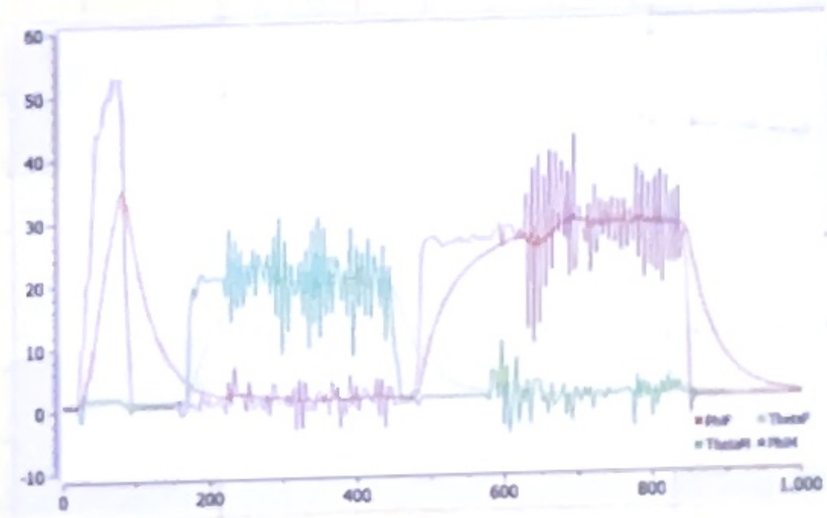


Figure 4

Showing vibration is not as bad on the filtered values compared to the ~~accel~~ measured.

Measuring pitch and Roll from the gyroscope

$$\theta_{gyro} = \theta_{gyro} + \omega_y \cdot dt$$

$$\phi_{gyro} = \phi_{gyro} + \omega_x \cdot dt$$

The gyroscope measures the angular velocity. If we keep up with the time delta, we can measure pitch and Roll.

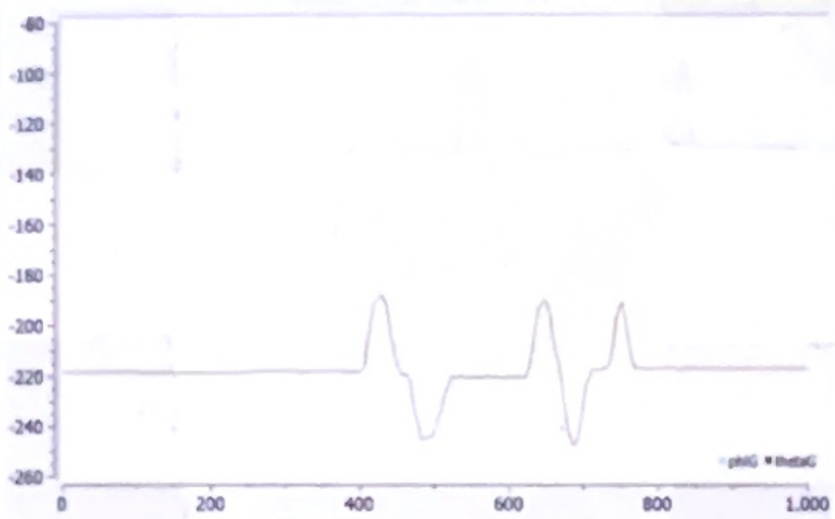


Figure 5
Testing the measurements from the gyroscope. Notice that the data in the short term is good, but overtime the readings began to drift further from the actual value.

Combining Accelerometer and Gyroscope data using a Complimentary Filter

$$\theta_{system} = [\theta_{system} + \omega_y \cdot dt] \cdot a + \theta_{measured} \cdot [1-a]$$

$$\phi_{system} = [\phi_{system} + \omega_x \cdot dt] \cdot a + \phi_{measured} \cdot [1-a]$$

If ω is changing, we will use more of the data from it and less of the data from the accelerometer. If ω is near 0, our system will approach the value from the accelerometer.

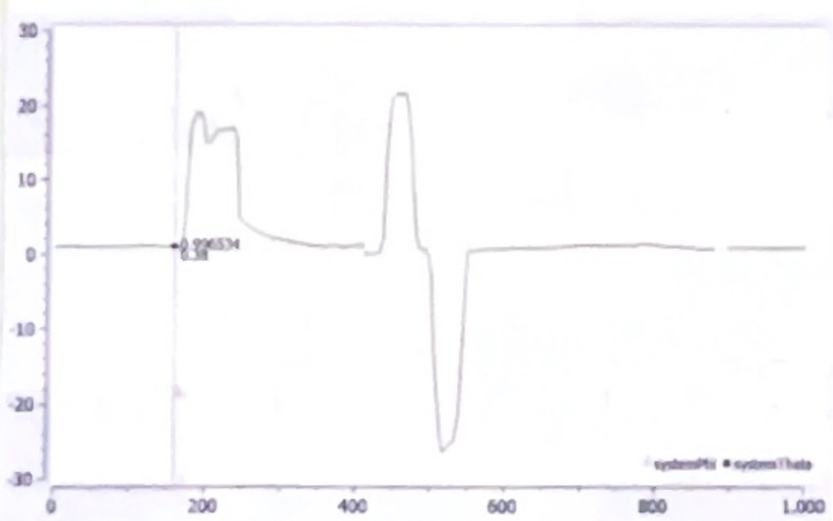


Figure 6

Testing the complimentary filter. Notice that the value changes realy quick, but when left still, it will approach a more stable value.

BNO055 Testing

05/23/23
Wesley Cooke

Basic Yaw Measurement

Assuming we stay in the XY plane, we can estimate our heading using $\tan^{-1}(M_y/M_x)$ where M_y and M_x are the y and x readings from the magnetometer. However, this stops working once you add Pitch and Roll. The derivation of the tilt compensated equation is left as an exercise for the reader.

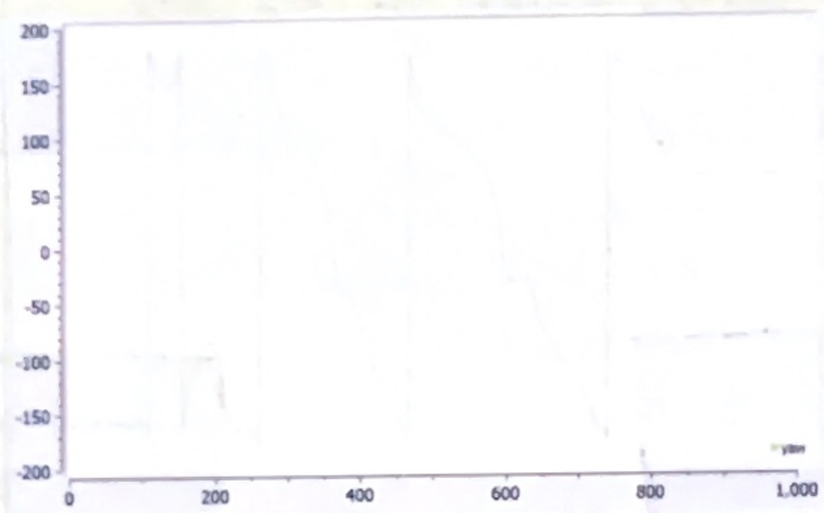


Figure 7
Testing the basic yaw measurement.

Comparing Complimentary Filter to onboard Euler and Quaternion data

The previous exercises are important to help build your intuition about how euler angles work. However, this sensor has registers that are keeping track of these values already.

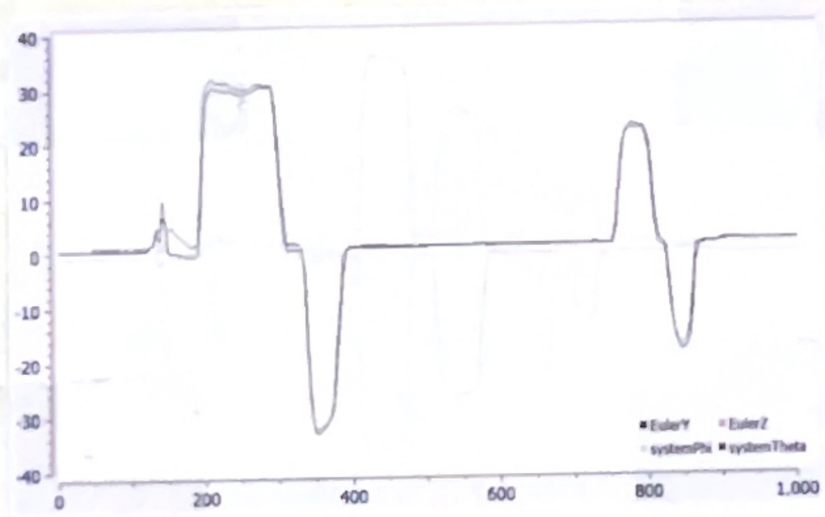


Figure 8
Comparing the onboard Euler angles to our filtered angles. Aside from having an axis flipped, we notice that the values are very similar.

~~The yaw isn't shown because it assumed a different forward direction.~~ → See page 21

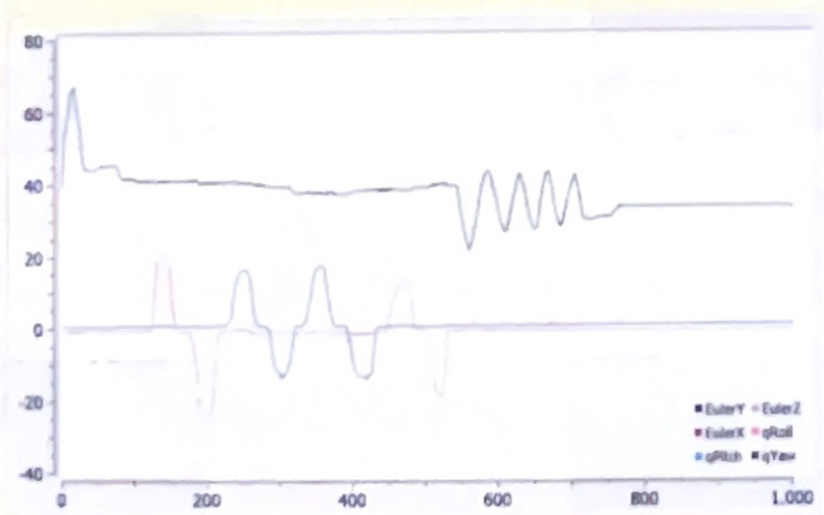


Figure 9
If we read the quaternion data and convert them to euler angles, we notice they almost exclusively overlap the eular angles of the board.

Code

Not all ~~pea~~ pieces of the code are shown for spacial reasons. Most is available on top tech dog's website.

```
void setup()
{
  Serial.begin(115200);
  offerEncouragement();

  myIMU.begin();
  delay(1000);

  // Tell the sensor to make a temperature measurement.
  // Some measurements may be based on this
  int8_t temp = myIMU.getTemp();
  Serial.print("Temp: ");
  Serial.println(temp);

  // Don't use the crystal on the chip itself.
  // Instead use the crystal on the board. -- More accurate
  myIMU.setExtCrystalUse(true);

  millisOld = millis();
}
```

```
Void offerEncouragement()
{
  Serial.Println("Stand by to Get Some!");
  Serial.Println("#GOPhysics");
  Serial.Println("Water is Life");
  Serial.Println("-----------");
}
```

BNO055 Testing

05/23/23
Wesley Costa

```
void loop()
{

  // Calibration Qualities. Measured on a scale of 0-3.
  uint8_t system, gyro, accel, mg = 0;
  myIMU.getCalibration(&system, &gyro, &accel, &mg);

  // Using the unified library
  // Go the IMU and bring me a vector with 3 components
  imu::Vector<3> acc = myIMU.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
  imu::Vector<3> gyr = myIMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
  imu::Vector<3> mag = myIMU.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
  imu::Vector<3> euler = myIMU.getVector(Adafruit_BNO055::VECTOR_EULER);
  imu::Quaternion quat = myIMU.getQuat();

  // Measured values from the accelerometer
  thetaM = atan2(acc.x()/9.8, acc.z()/9.8)/2/3.141592654*360; // Pitch
  phiM = atan2(acc.y()/9.8, acc.z()/9.8)/2/3.141592654*360;   // Roll
```

```
  // Calculate the new filtered values
  phiFnew = a * phiFold + (1-a) * phiM;
  thetaFnew = a * thetaFold + (1-a) * thetaM;

  // Get the current dt, and remember the old millis.
  dt = (millis() - millisOld)/1000.0;
  millisOld = millis();

  // System theta calculation using a complimentary filter
  theta = (theta-(gyr.y()*dt))*a + thetaM*(1-a); // Pitch
  phi = (phi+(gyr.x()*dt))*a + phiM*(1-a);       // Roll

  // theta = thetaold + omega * dt. Omega comes from gryo.
  thetaG = thetaG-gyr.y()*dt;
  phiG = phiG+gyr.x()*dt;

  phiRad = phi/360*(2*3.14);
  thetaRad = theta/360*(2*3.14);

```

```
  thetaFold = thetaFnew;
  phiFold = phiFnew;
  delay(BNO055_SAMPLERATE_DELAY_MS);
```

```
  // Calculate Euler angles from quaternions.
  // using https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
  double qRoll = atan2(2*(quat.w()*quat.x() + quat.y()*quat.z()), 1-2*(quat.x()*quat.x()+quat.y()*quat.y()));
  double qPitch = -3.14/2 + 2*atan2(sqrt(1+2*(quat.w()*quat.y() - quat.x()*quat.z())), sqrt(1-2*(quat.w()*quat.y() - quat.x()*quat.z()))); 
  double qYaw = atan2(2*(quat.w()*quat.z()+quat.x()*quat.y()), 1-2*(quat.y()*quat.y() + quat.z()*quat.z()));
```

Relative vs. Absolute Orientation

The BNO is capable of providing a yaw reading relative to where the sensor turned on at, or a yaw reading relative to the earths magnetic field. (ie absolute).

To enable the Absolute readings, the BNO needs to be set to the compass mode. More Modes and details are on page 22 and 23 of the datasheet.
https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/~~bno~~bst-bno055-ds000.pdf

```
void setup()
{
  Serial.begin(115200);
  myIMU.begin();
  delay(1000);
  myIMU.setMode(adafruit_bno055_opmode_t::OPERATION_MODE_COMPASS);
  myIMU.setExtCrystalUse(true);
  Serial.println("BNO Connected!");
}

void loop()
{
  euler = myIMU.getVector(Adafruit_BNO055::VECTOR_EULER);
  Serial.println(euler.x(), 2);
  delay(500);
}
```

The set mode method allows us to set the BNO to the compass mode. If this isn't here, the yaw we recieve from the BNO is relative to the position it powered up in.

Overview

Single or Multiple positioning module
Quick delivery of position data like lat. & lon.
I2C or UART
Positioning Accuracy: 2.0 m CEP

Library

"DFRobot_GNSS.h"
https://github.com/DFRobot/DFRobot_GNSS

Sketch → include Library → Add .Zip Library

Experiment

Given two gps coordinates, we would like to determine the heading between the two points. IE What heading do I need to travel from point A to Arrive at point B. There are two methods to calculate the heading. One assumes the earth is flat and the other assumes the earth is a sphere.

Procedure

I hard coded a destination GPS coordinate. Then, I wrote code to get my current location. These two gps locations are used in the two algorithms to determine the heading from my current location to the destination location.

Calculate Heading assuming the earth is a sphere

The following derivation was found at:
https://web.archive.org/web/20171219235009/http://mathforum.org/library/drmath/view/55417.html

Let's define three unit vectors, each in the direction of the line from the center of the earth to a point on the surface: N in the direction of the north pole, A in the direction of the initial point, and B in the direction of the final point on the course. Then the bearing we seek is the angle between the plane containing N and A, and the plane containing A and B. Thus it equals the angle between vectors perpendicular to these planes, namely, NxA and BxA.

Let point A have latitude lat1 and longitude 0 (we can rotate our coordinate system so this is true), and let point B have latitude lat2 and longitude dlon (the difference between the actual longitudes of A and B). Then we can calculate NxA and BxA:

```
N = (0, 0, 1)
A = (cos(lat1), 0, sin(lat1))
B = (cos(lat2)*cos(dlon), cos(lat2)*sin(dlon), sin(lat2))

NxA = (0, cos(lat1), 0)
BxA = (sin(lat1)*cos(lat2)*sin(dlon),
       cos(lat1)*sin(lat2)-sin(lat1)*cos(lat2)*cos(dlon),
       -cos(lat1)*cos(lat2)*sin(dlon))
```

The usual way to find the angle would be to take the dot product of these vectors, divide by the product of the magnitudes of the vectors and take the arccosine. But that would be quite a mess. Instead we can take advantage of the fact that NxA is parallel to the y axis. The tangent of the angle between BxA and the y axis is the component of BxA in the x-z plane (the square root of the sum of the squares of the x and z components) divided by the y component:

```
tan(theta) = sqrt((sin(lat1)*cos(lat2)*sin(dlon))^2 +
                  (-cos(lat1)*cos(lat2)*sin(dlon))^2) /
             (cos(lat1)*sin(lat2)-sin(lat1)*cos(lat2)*cos(dlon))
```

The numerator simplifies to

```
sqrt(cos(lat2)^2 * sin(dlon)^2) * sqrt(sin(lat1)^2 + cos(lat1)^2)
= cos(lat2)*sin(dlon)
```

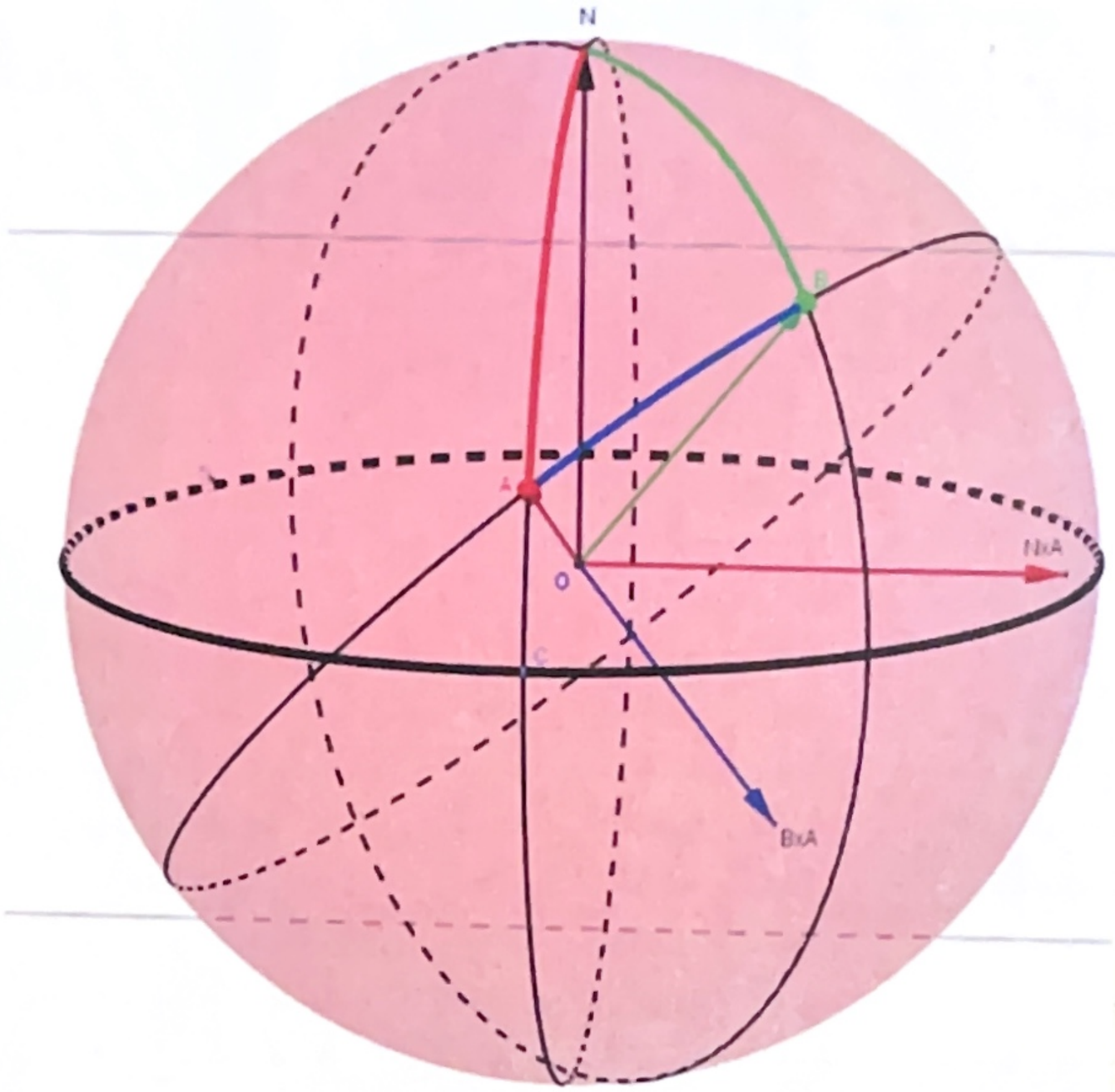
Thus

```
tan(theta) = cos(lat2)*sin(dlon) /
             (cos(lat1)*sin(lat2)-sin(lat1)*cos(lat2)*cos(dlon))
```

That's the formula. If we take the arctan of both sides, we get a value in the range from -pi/2 to pi/2 radians. The problem is, this doesn't distinguish between opposite directions, NE vs. SW for instance. But the function atan2(y,x) returns the arctan of y/x with adjustments for the signs of x and y so that the angle returned is the angle of the cartesian point (x,y) in polar coordinates -- just what we need here.

The only problem is that it returns a value in the range (-pi, pi]. The function mod(..., 2*pi) moves negative angles up to the range (pi, 2*pi). To get a final answer in degrees (0 to 360), you must multiply by 180/pi.

- Doctor Rick, The Math Forum
  http://mathforum.org/dr.math/

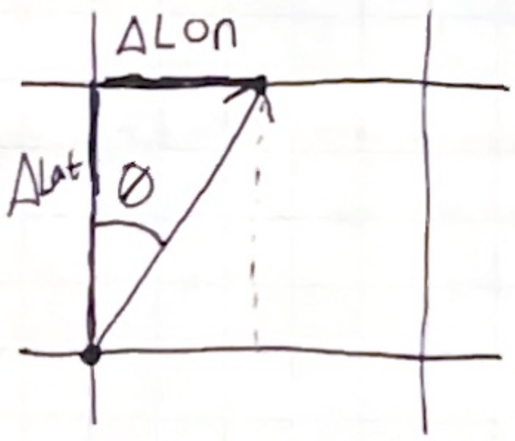


This shows a unit sphere. The heavy black circle is the equator; vector N is perpendicular to that (pointing to the north pole). Vectors A and B point toward the points A and B on the earth's surface. A blue arc is drawn between A and B along a great circle. The bearing from A to B is the angle at A between the red arc (north) and the blue arc.

Vector NxA is the red vector pointing right; it is perpendicular to the great circle through A and N. Vector BxA is the blue vector pointing down and to the right; it is perpendicular to the great circle through A and B. The angle between NxA and BxA is thus the same as the angle we seek (the bearing from A to B).

The rest of the work shown in my derivation of the bearing formula is how to find the tangent of the angle between NxA and BxA, in terms of the vector components of BxA.

Calculate heading assuming the earth is a ~~sphere~~ flat



$$\tan(\theta) = \frac{\Delta Lon}{\Delta Lat}$$

$$\theta = \tan^{-}\left(\frac{\Delta Lon}{\Delta Lat}\right)$$

Assuming the earth is flat allows us to view lat and lon as a square grid. Then we can apply some geometry to determine our heading.

Results

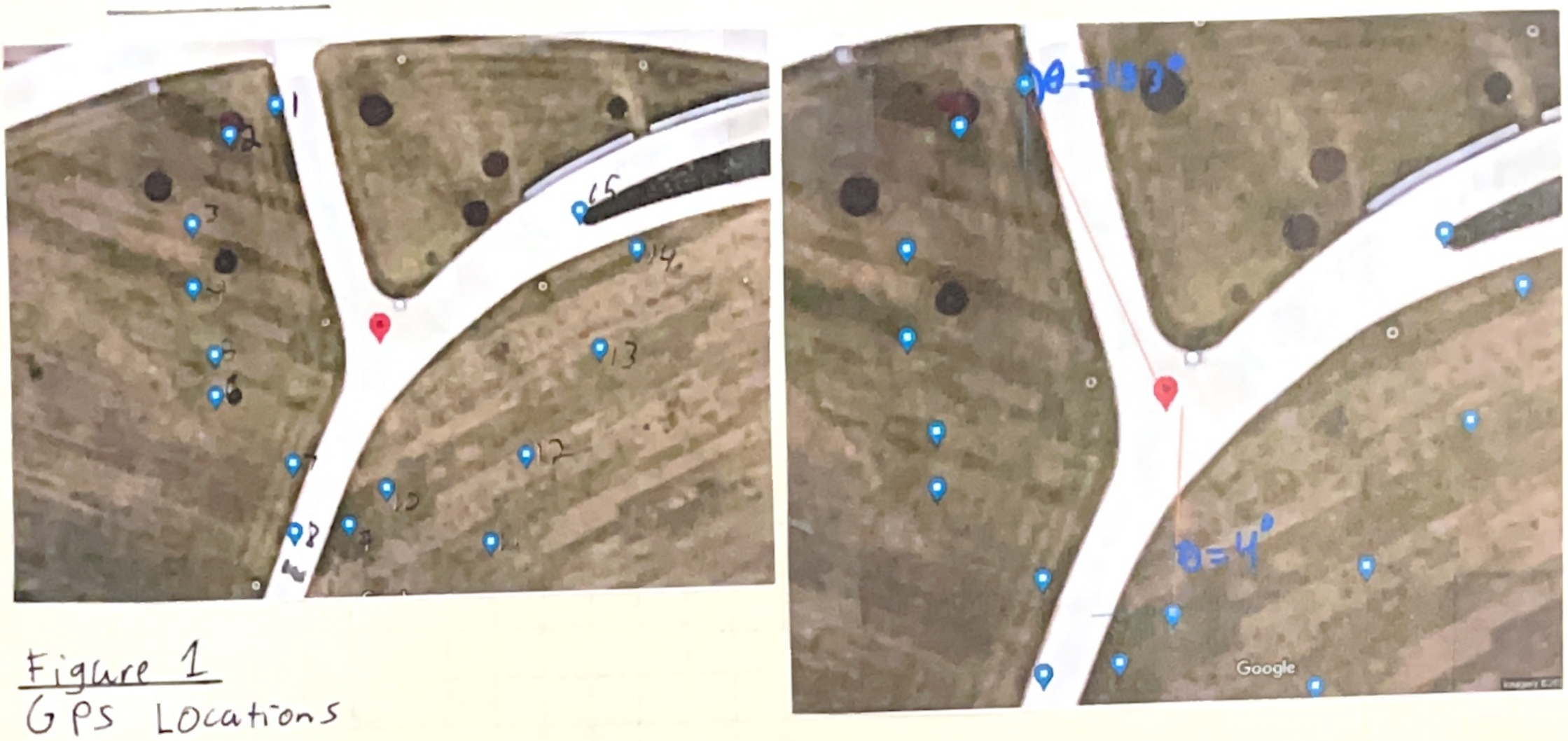


Figure 1
GPS Locations

Sample	Lat	Lon	Spherical Earth	Flat Approximation
1	33.467697	-81.991012	153.759719	150.57257
2	33.467678	-81.99105	140.93544	136.735748
3	33.46762	-81.991081	120.937568	117.699501
4	33.467578	-81.991081	100.338203	99.926269
5	33.467533	-81.991065	81.477973	83.659828
6	33.467506	-81.991065	67.972747	73.009193
7	33.46746	-81.991004	35.216751	41.009098
8	33.467414	-81.991004	24.646091	29.74489
9	33.467418	-81.990959	12.225576	13.240524
10	33.467441	-81.990928	3.080645	4.085618
11	33.467403	-81.990844	334.51358	329.931396
12	33.46746	-81.990814	312.511322	307.476165
13	33.467529	-81.990753	280.193603	276.20343
14	33.467594	-81.990722	255.440231	257.47116
15	33.46762	-81.990768	241.887588	245.658874

Table 1
Spherical heading calculation vs. flat approximation heading calculations.

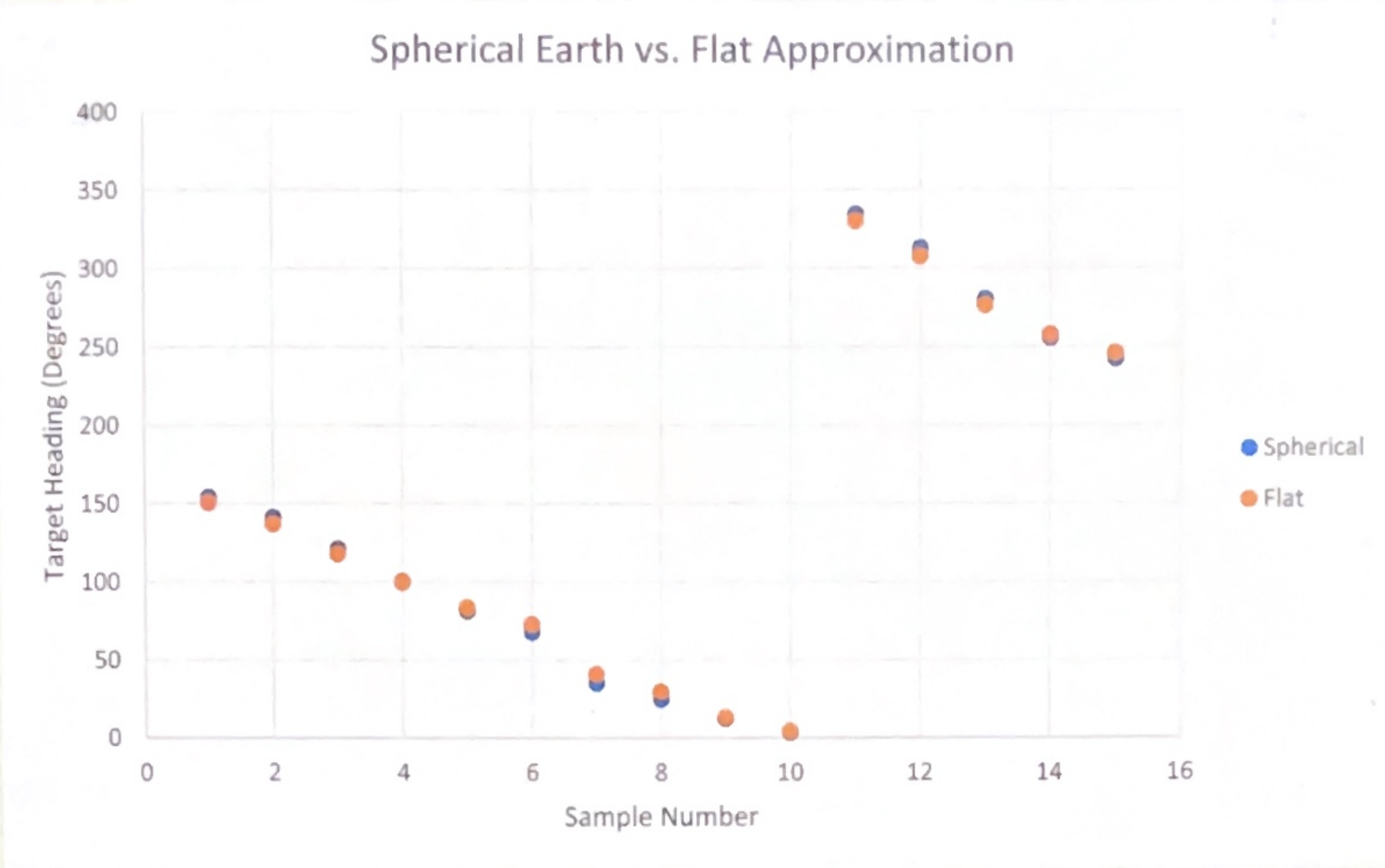
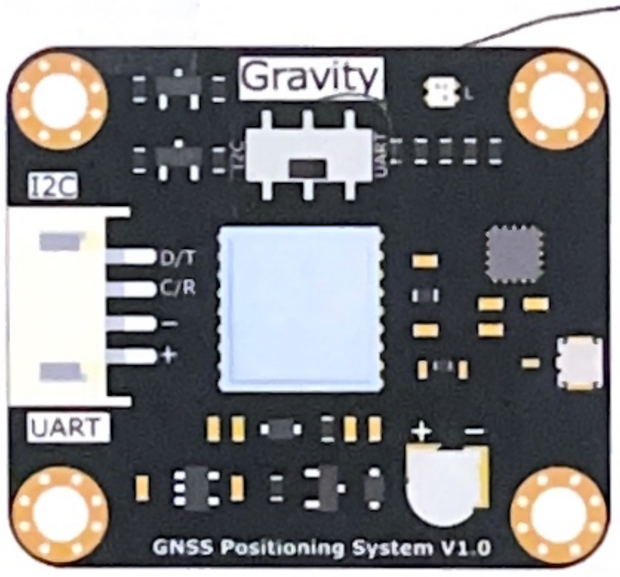


Figure 2

Comparing the spherical heading to the flat heading, Notice that since we were at small distances, the approximation isn't far off. However, if the distance is much bigger, the difference will also be bigger.

Board Overview



Choose I2C or UART

Num	Label	Description
1	D/T	I2C data line SDA/UART Data Transmitting- TX
2	C/R	I2C clock line SCL/UART Data Receiving- RX
3	-	GND
4	+	VCC

Wiring

You can use either I2C or UART.

If using UART:

TX of GNSS → Rx of Arduino
Rx of GNSS → Tx of Arduino

If using I2C:

SDA → SDA
SCL → SCL

* Make sure the switch is flipped the right way

DF Robot GNSS

05/26/23
Wesley Crotts

Code

```
#include <math.h>
#include <SoftwareSerial.h>
#include "DFRobot_GNSS.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <wire.h>

SoftwareSerial mySerial(10, 11);
DFRobot_GNSS_UART gnss(&mySerial, 9600);

#define RAD_TO_DEG (180/3.141592)
#define DEG_TO_RAD (3.141592/180)

sLonLat_t currentLat;
sLonLat_t currentLon;

float cLat;
float cLon;

char cLatDir;
char cLonDir;

float dLat = 33.467547;
float dLon = -81.990931;
float targetHeading;
float flatEarthHeading;
uint8_t satsUsed;
float getTargetHeading(float cLat, float cLon, float dLat, float dLon)
{
  // Calculate the heading using something based on haversines.
  // https://web.archive.org/web/20171219235009/http://mathforum.org/
  // library/drmath/view/55417.html
  cLat = cLat * DEG_TO_RAD;
  cLon = cLon * DEG_TO_RAD;
  dLat = dLat * DEG_TO_RAD;
  dLon = dLon * DEG_TO_RAD;

  float bearing = atan2(
    sin(dLon - cLon) * cos(dLat),
    cos(cLat) * sin(dLat) - sin(cLat) * cos(dLat) * cos(dLon - cLon)
    );

  // Return bearing in radians
  return bearing;
}
```

Rx of arduino
Tx of arduino

05/26/23
Wesley Cook

DFRobot GNSS


```
49 void setup()
50 {
51   Serial.begin(115200);
52   while(!gnss.begin())
53   {
54     Serial.println("NO Devices!");
55     delay(1000);
56   }
57
58   gnss.enablePower();
59
60   /** Set the galaxy to be used
61    *   eGPS              USE gps
62    *   eBeiDou           USE beidou
63    *   eGPS_BeiDou       USE gps + beidou
64    *   eGLONASS          USE glonass
65    *   eGPS_GLONASS      USE gps + glonass
66    *   eBeiDou_GLONASS   USE beidou +glonass
67    *   eGPS_BeiDou_GLONASS USE gps + beidou + glonass
68    */
69    gnss.setGnss(eGPS_BeiDou_GLONASS);
70    gnss.setRgbOn();
71 }
```

Should probably only use eGPS

```
72 void loop()
73 {
74   // Get our current GPS coords
75   currentLat = gnss.getLat();
76   currentLon = gnss.getLon();
77   satsUsed = gnss.getNumSatUsed();
78
79   // Get the decimal degree's representation of the Lat and Lon.
80   cLat = currentLat.latitudeDegree;
81   cLon = currentLon.lonitudeDegree;
82
83   // Get the directions
84   // These directions are flipped in the library???
85   cLatDir = currentLon.lonDirection;
86   cLonDir = currentLat.latDirection;
87
88   // Correct the lattitude and longitude based on directions
89   if (cLatDir == 'S')
90   {
91     cLat = -cLat;
92   }
93   if (cLonDir == 'W')
94   {
95     cLon = -cLon;
96   }
97
98   // Get Target heading in degrees using the fancy equation
99   targetHeading = getTargetHeading(cLat, cLon, dLat, dLon) * RAD_TO_DEG;
100
101   // Get the heading using an approximation in degrees
102   flatEarthHeading = atan2((dLon-cLon)*DEG_TO_RAD, (dLat-cLat)*DEG_TO_RAD) * RAD_TO_DEG;
103
104   // Make sure headings are positive
105   if (targetHeading < 0) {targetHeading += 360;}
106   if (flatEarthHeading < 0) {flatEarthHeading += 360;}
107
108   printStuff();
109   delay(1000);
110 }
```

06/08/23
Wesley Cooper

Standalone GNss Data Logger

Objective

Develop a small + portable GNss data logger to determine the accuracy of the gps data points.

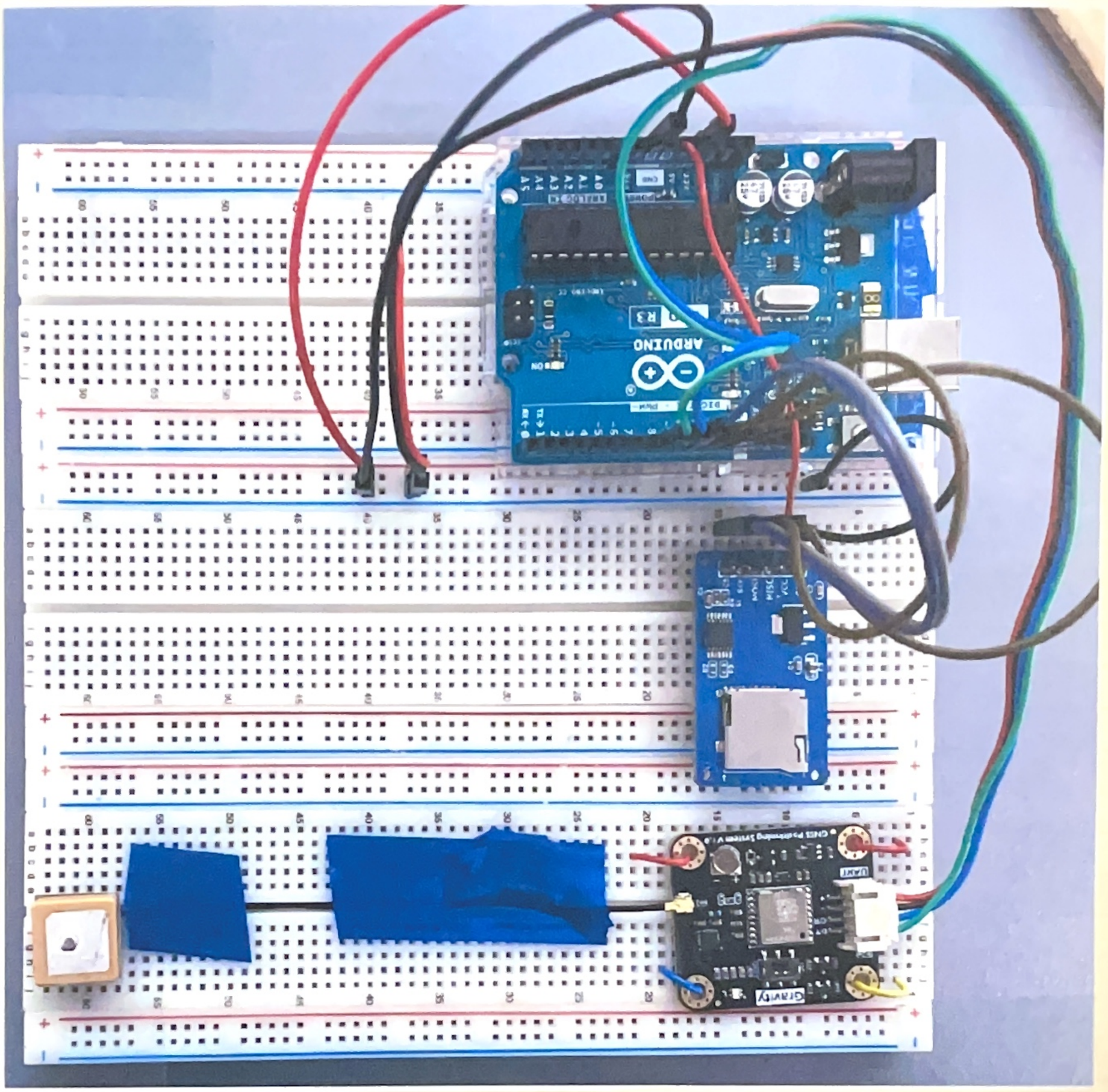


Figure 1 - Example Circuit

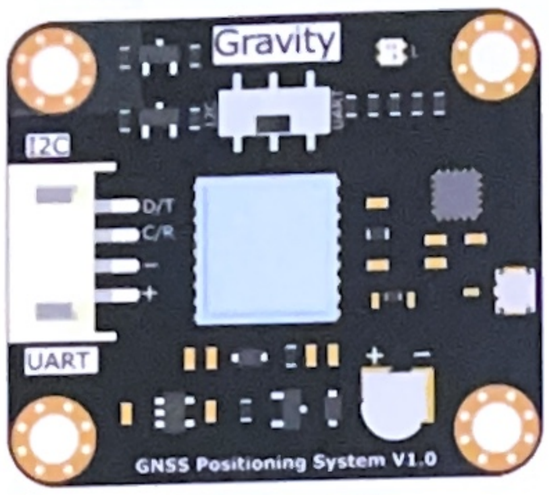
Here, we have an arduino uno connected to a gnss module using UART, and an SD card module using SPI.

Standalone GNSS Data Logger

06/09/23
Wesley Cooper

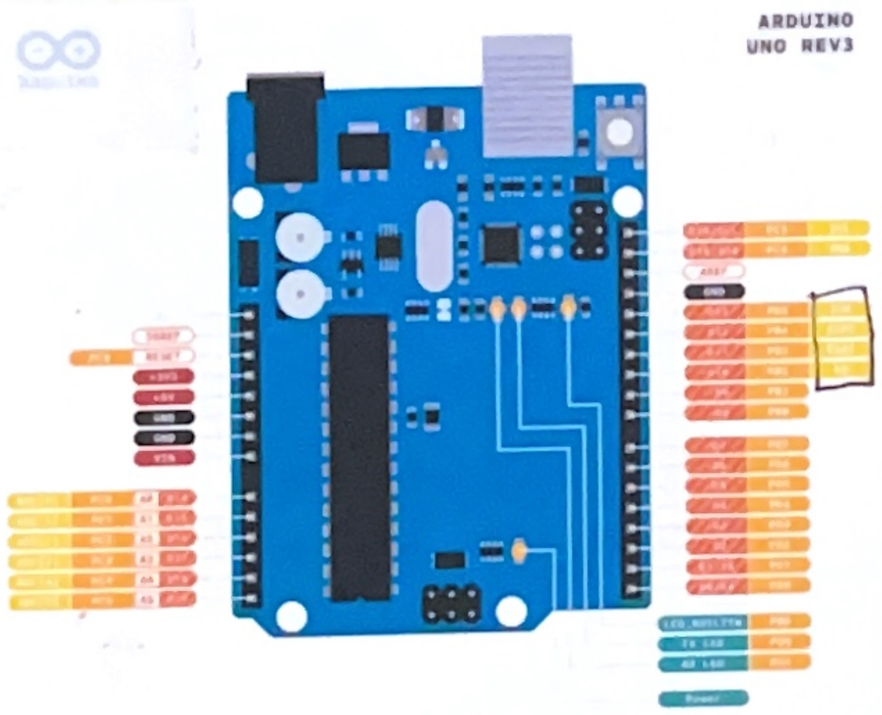
Wiring

Board Overview



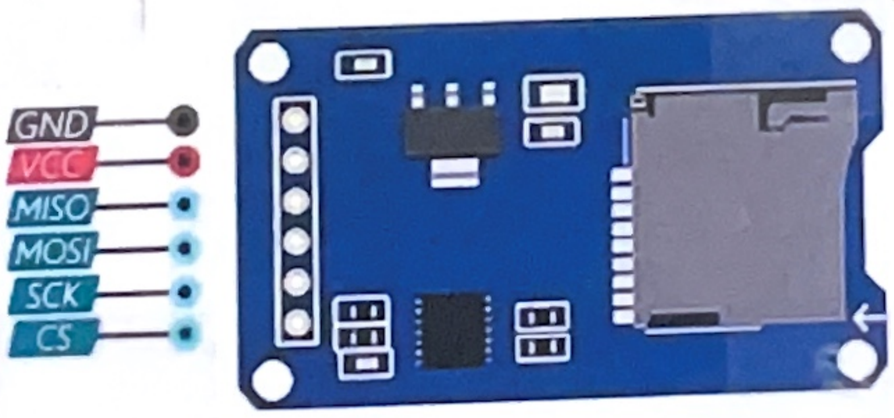
Num	Label	Description
1	D/T	I2C data line SDA/UART Data Transmitting- TX
2	C/R	I2C clock line SCL/UART Data Receiving- RX
3	-	GND
4	+	VCC

GNSS Pin	Arduino Pin
1 (Tx)	8 (Rx)
2 (Rx)	9 (TX)



SPI pins on arduino

PIN Name	Arduino Pin
MISO	12
MOSI	11
SCK	13
CS/SS	10



Plotting Data

Copy and paste data into excel.
Click the first column and press ctrl + a.
This should select all the data.
Click on "Data" → "Text to columns"
Select the "Delimited" buble.
Click Next.
Check the comma box.
Click Finish.
Make sure each column is properly labeled.
Go to "my maps" on google and import your excel file.

Data

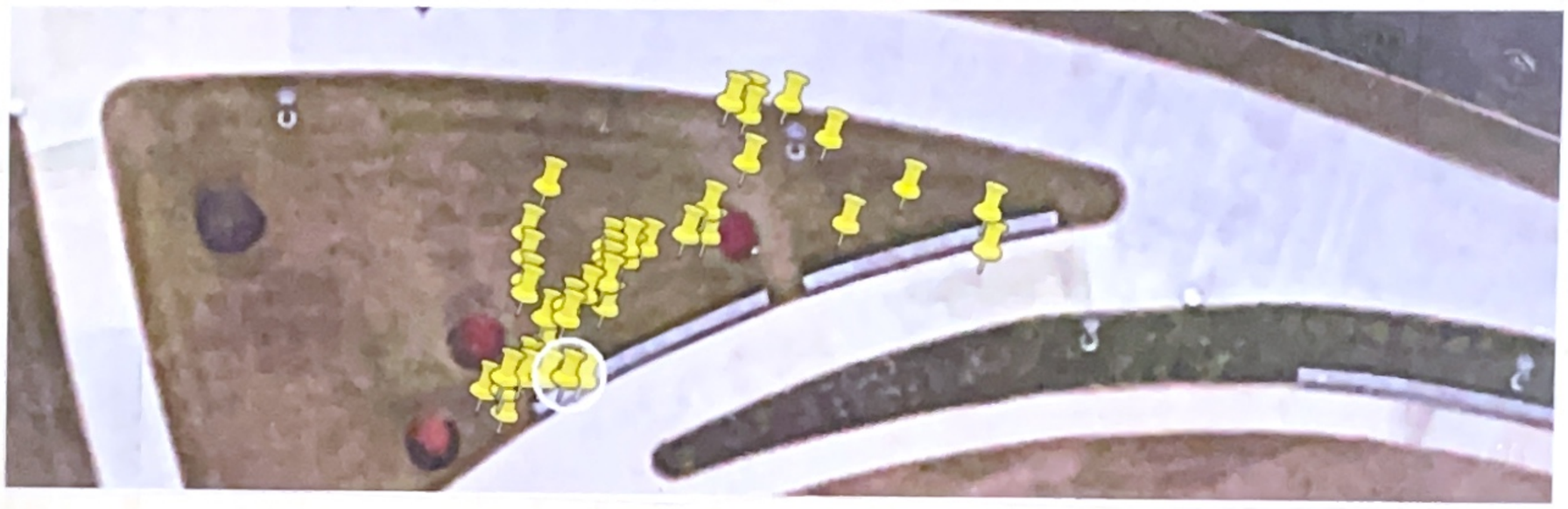


Figure 2 - Sitting on a bench
These points were collected in the same location, on the edge of a bench. Notice how spread out they are.

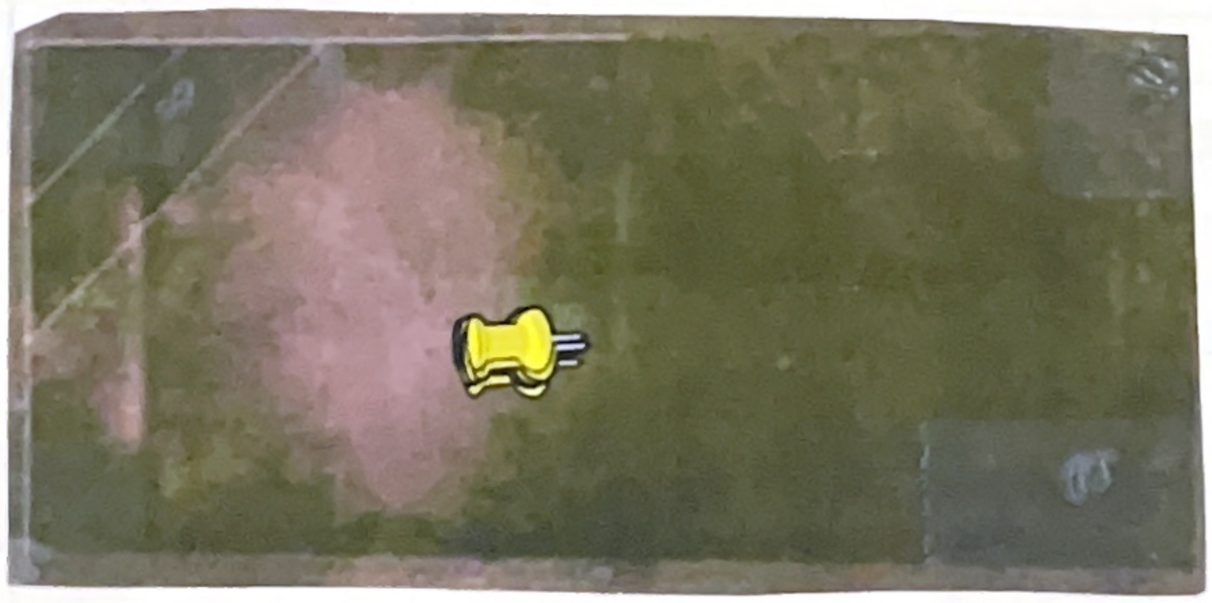


Figure 3 - in the middle of a field
Notice how these points aren't spread out as bad as the bench was...

Standalone GNSS Data Logger

06/08/23
wesley
cecta

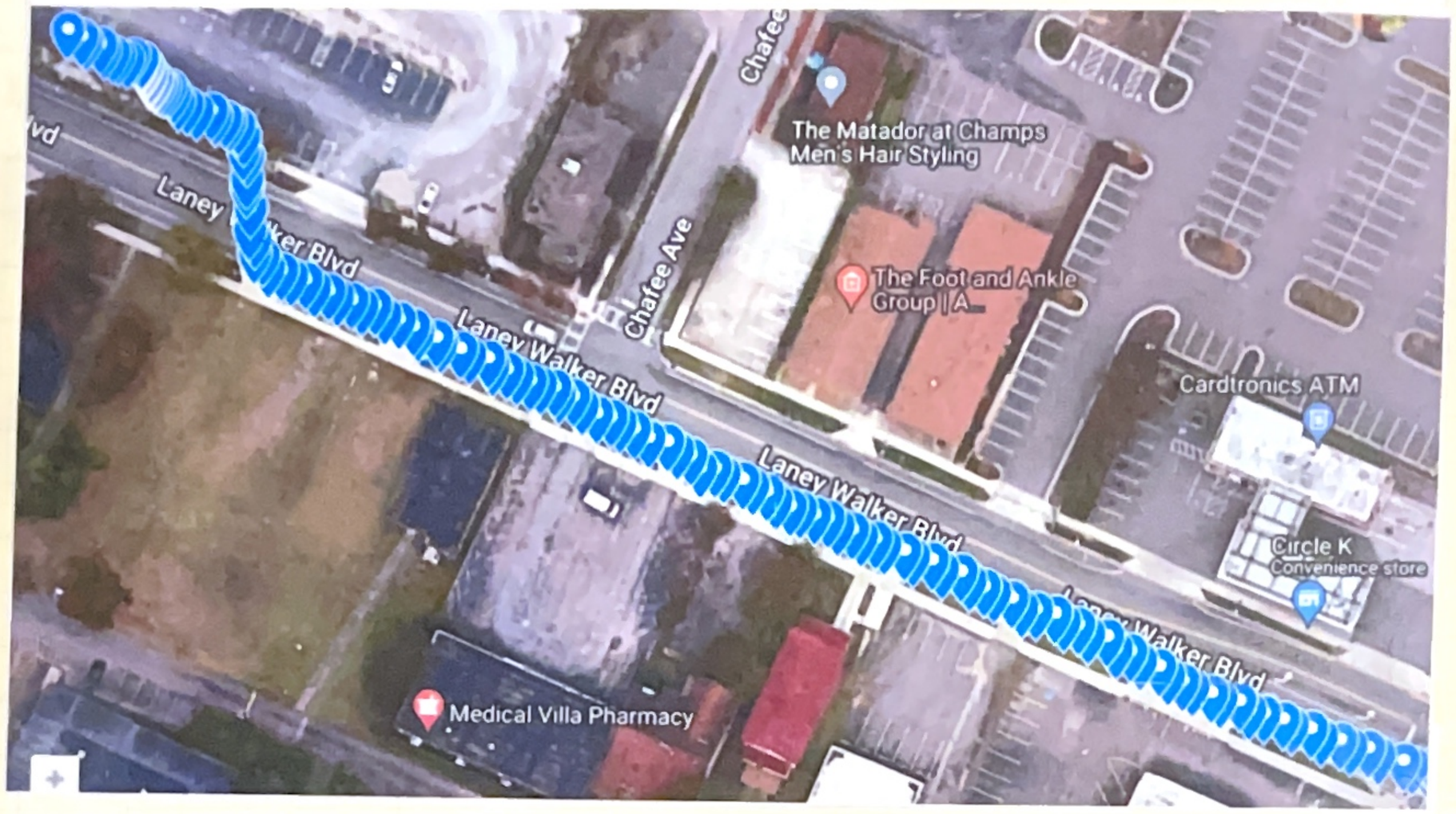


Figure 4 - on a brisk walk

These points were collected while walking on a sidewalk.

Figure 5 - another brisk walk

We experimented with a few brisk walks to try to see if they would mess up as bad as ~~the first~~ in Figure 2. None of the brisk walks messed up very far.

Figure 6 - A slow "robot paced" walk

After expirencing difficulties tracking the path of a robot on a side walk, we followed the same path seen in Figure 5. However, we moved very slowly. Notice how it doesn't even complete a ~~or~~ loop.

Conclusions

After several trials similar to the ones above, we concluded that the GNSS module is best suited for a walking pace or faster. Even a leisurily stroll is slow enough to introduce inaccuracies in the GNSS module. It has consistently been unable to track slow moving objects.

Standalone GNSS Data Logger

06/08/23
Wesley Cooper

Code

```
#include <SPI.h>
#include <SD.h>
#include <SoftwareSerial.h>
#include "DFRobot_GNSS.h"

SoftwareSerial mySerial(8, 9);
DFRobot_GNSS_UART gnss(&mySerial, 9600);

sLonLat_t currentLat;
sLonLat_t currentLon;

// These variables will hold our
// current Lat and current Lon.
float cLat;
float cLon;

// These variables will give us the
// direction of our cLat and cLon
char cLatDir;
char cLonDir;

uint8_t satsUsed;

uint16_t num = 1;

const int chipSelect = 10;

void setup()
{
  Serial.begin(115200);
  pinMode(chipSelect, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect))
  {
    Serial.println("Card failed, or not present");
    while (1);
  }
  Serial.println("card initialized.");

  // Put a separator in the SD card
  File outputfile = SD.open("data.txt", FILE_WRITE);
  outputfile.println("=================");
  outputfile.close();

  while(!gnss.begin())
  {
    Serial.println("Check the gnss mode and wiring...");
    while(1);
  }
  gnss.enablePower();
  gnss.setGnss(eGPS);
  gnss.setRgbOn();
  Serial.println("GPS Connected!");
}
```

```
void loop()
{
    // Update our Lat and Lon variables
    currentLat = gnss.getLat();
    currentLon = gnss.getLon();
    satsUsed = gnss.getNumSatUsed();

    if (satsUsed > 0)
    {
      // Turn the lat/long data into decimal degrees
      cLat = currentLat.latitudeDegree;
      cLon = currentLon.lonitudeDegree;

      // Get the cardinal direction associated with our decimal degrees.
      // These directions are flipped in the library ???
      cLatDir = currentLon.lonDirection;
      cLonDir = currentLat.latDirection;

      // Correct the lattitude and longitude based on their cardinal directions
      if (cLatDir == 'S')
      {
        cLat = -cLat;
      }
      if (cLonDir == 'W')
      {
        cLon = -cLon;
      }

```

```

      // Open the data file in append mode
      File outputfile = SD.open("data.txt", FILE_WRITE);

      if (outputfile)
      {
        outputfile.print(cLat, 6);
        outputfile.print(", ");
        outputfile.print(cLon, 6);
        outputfile.print(", ");
        outputfile.print(num);
        outputfile.print(", test");
        outputfile.println(", 111");
      }
      outputfile.close();
      num++;
    }
    else
    {
      Serial.println("No sats yet...");
    }
}
```

06/14/23
Wesley Cooper

Level Shifting MOSFET

Objective

To create a bidirectional level shifter using a MOSFET.

Level Shifting

Take a logic level voltage and convert it to another logic level voltage. Useful to convert UART/I2C/SPI communications between a device that operates at a 5V logic level and another at 3.3V logic level...

/23, 4:03 PM Logic Level Shifting Basics | DigiKey

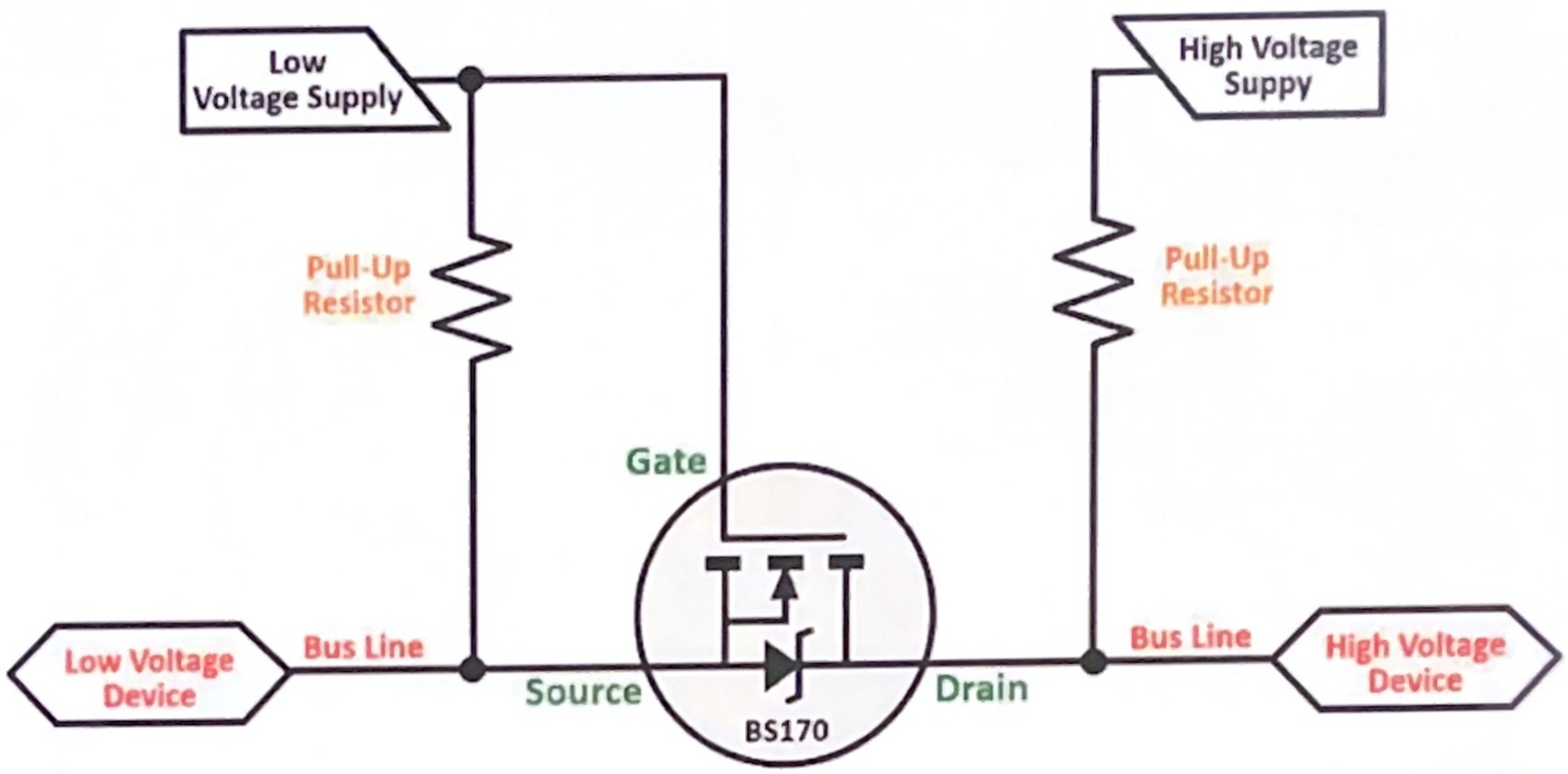


Figure 1: Basic, single bus, level translation MOSFET circuit.

Level Shifting MOSFET

06/14/23
Wesley Cooke

The logic high levels on each side of the MOSFET are achieved by pull-up resistors to their respective supplies providing conversion of fast mode (400 kHz) I²C signals or other similarly fast digital interfaces. The gate of the MOSFET is held at the low voltage supply level. When no device is pulling down the bus line, the bus line at the MOSFET's source is pulled up by the low voltage pull-up resistors. The MOSFET's Gate/Source voltage (VGS) is below the threshold, and the MOSFET is not conducting. This allows the bus line at the MOSFET's drain to be pulled up by the higher-voltage pull-up resistor. The bus lines on each side of the MOSFET are held HIGH but at different voltage levels. See Figure 2.

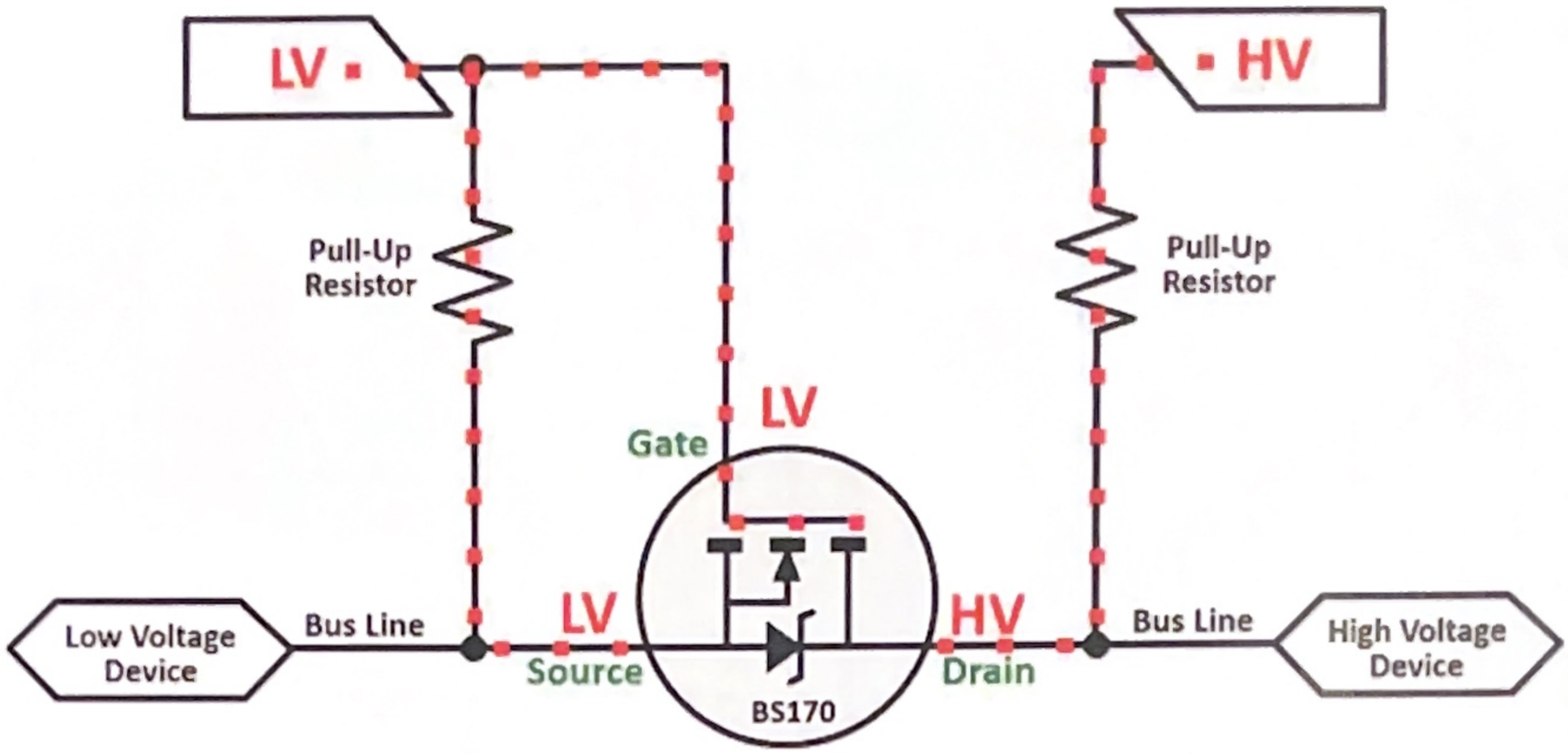


Figure 2: Logical HIGH voltage translation.

If the low voltage device pulls down the bus line at the MOSFET's source and the gate remains at the low voltage supply, VGS rises above the threshold and the MOSFET starts to conduct. The bus line at the MOSFET's drain is now pulled down as well. Refer to Figure 3.

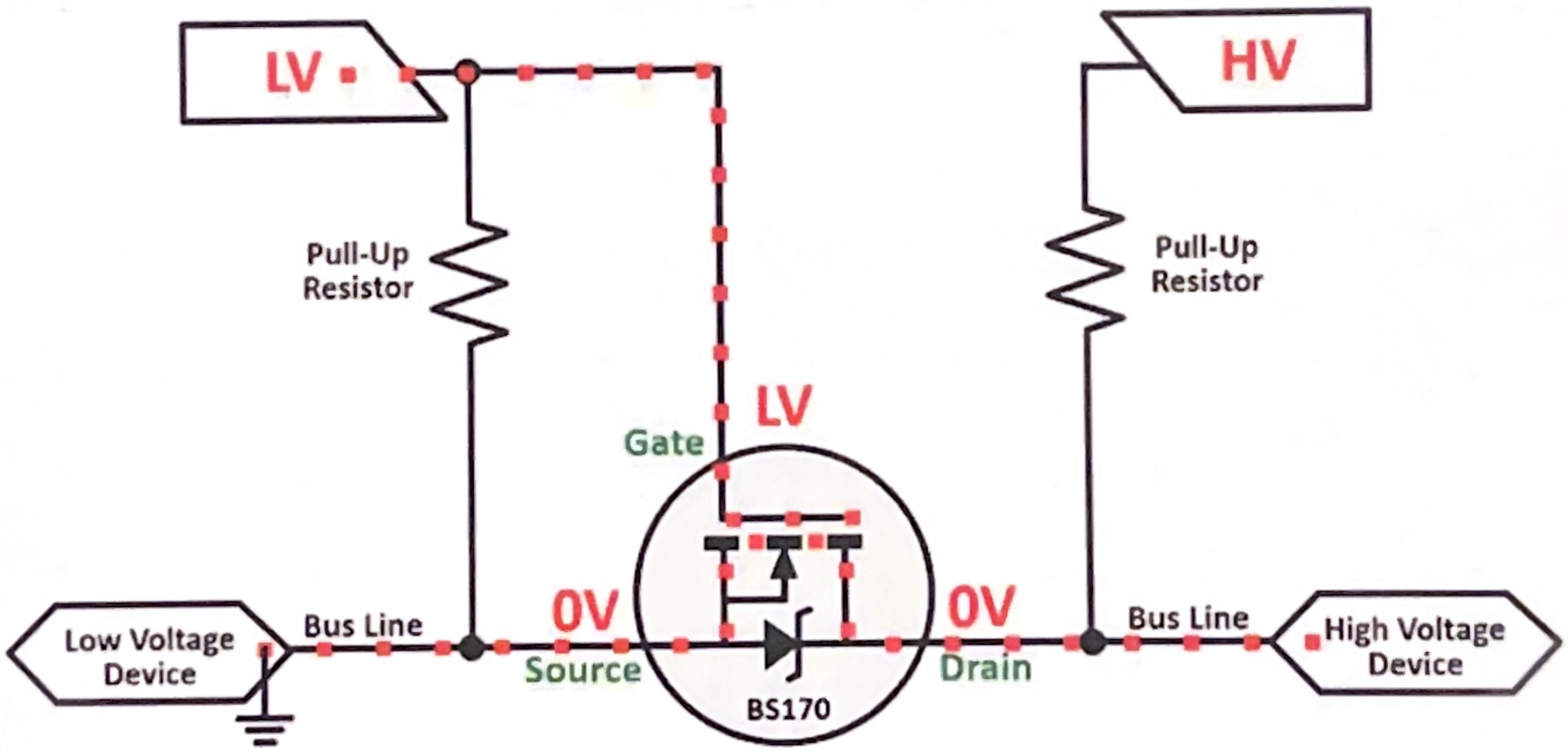


Figure 3: Logical LOW voltage translation initiated by low voltage device.

If the high voltage device pulls down the bus line at the MOSFET's drain, the MOSFET's substrate diode allows the source to also be partially pulled down due to a small amount of voltage dropped across the diode. See Figure 4.

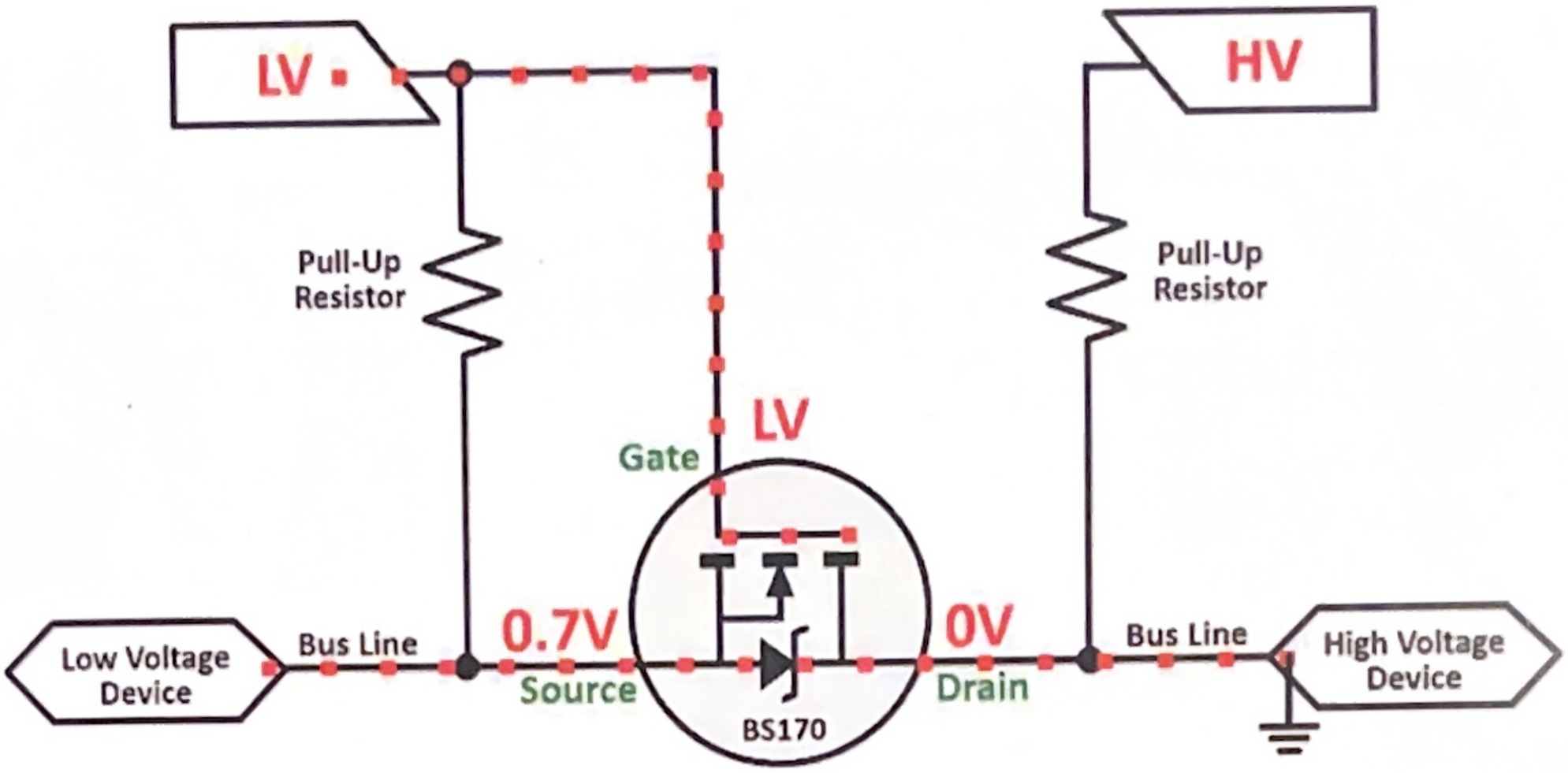


Figure 4: Near logical LOW voltage translation initiated by a high voltage device.

When the MOSFET's source is partially pulled down, the VGS rises above the threshold and the MOSFET starts to conduct effectively bypassing the substrate diode. See Figures 5.

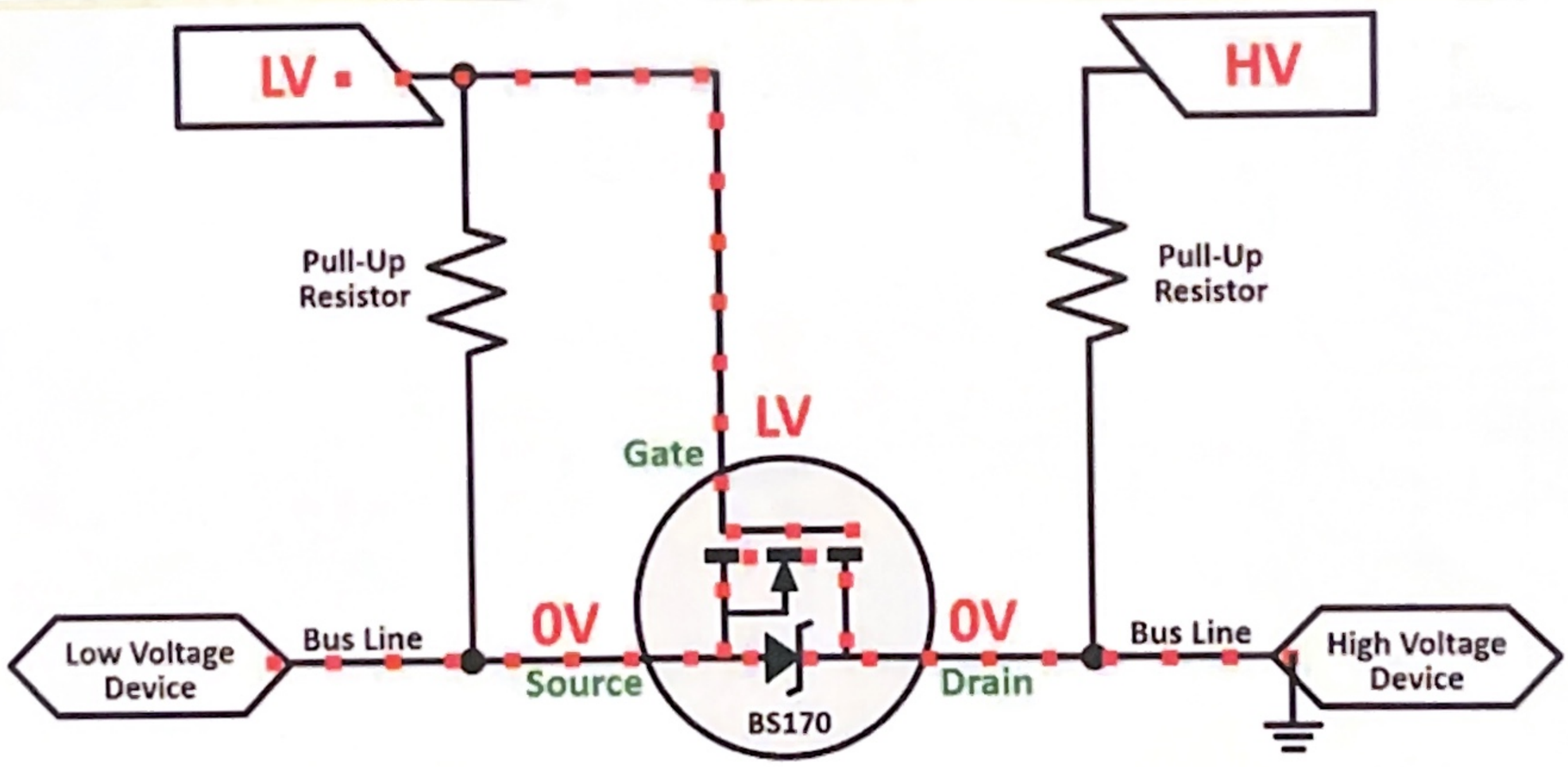


Figure 5: Full logical LOW voltage translation initiated by high voltage device.

The three states show transferred logic levels in both directions of the bus system, independent of the driving section. Many combinations of high and low voltage supplies are possible depending upon the capabilities of the MOSFET. Whether the logic level conflict involves point-to-point GPIO, sensor output, or bidirectional multi-line communication, MOSFET level shifters are useful tools. Figure 5 demonstrates the implementation of a translated, two-line, bidirectional communication circuit using two MOSFETs.

Level shifting MOSFET

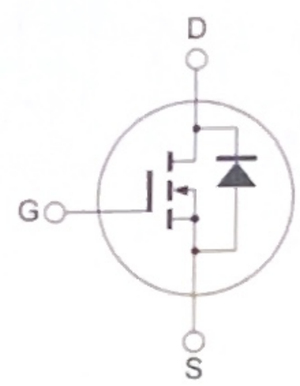
06/14/23
Wesley Cocka

Enhancement Mode N-channel MOSFET

These will allow current to be pulled through the drain when a voltage difference exists between the gate and source. If $V_{GS}=0$, then the mosfet is off and not conducting current through the drain.

RJK 5030

RENESAS Package code: PRSS0003AF-A
(Package name: TO-220FL)



1. Gate
2. Drain
3. Source

Absolute Maximum Ratings

(Ta = 25°C)

Item	Symbol	Value	Unit
Drain to source voltage	V_{DSS}	500	V
Gate to source voltage	V_{GSS}	±30	V
Drain current	I_D	5	A
Drain peak current	$I_{D\ (pulse)}$ Note1	20	A
Avalanche current	I_{AP} Note3	5	A
Channel dissipation	Pch Note 2	28.5	W
Channel to case thermal Impedance	θch-c	4.38	°C/W
Channel temperature	Tch	150	°C
Storage temperature	Tstg	–55 to +150	°C

Notes: 1. Pulse width limited by safe operating area.
2. Value at Tc = 25°C
3. STch = 25°C, Tch ≤ 150°C

06/14/23 Wesley Cecil

Level Shifting MOSFET

RJK5030DPP-M0

Electrical Characteristics

(Ta = 25°C)

Item	Symbol	Min	Typ	Max	Unit	Test Conditions
Drain to source breakdown voltage	$V_{(BR)DSS}$	500	—	—	V	I_D = 1 mA, V_{GS} = 0
Zero gate voltage drain current	I_{DSS}	—	—	10	μA	V_{DS} = 500 V, V_{GS} = 0
Gate to source leak current	I_{GSS}	—	—	±0.1	μA	V_{GS} = ±30 V, V_{DS} = 0
Gate to source cutoff voltage	$V_{GS(off)}$	3.5	—	4.5	V	V_{DS} = 10 V, I_D = 1 mA
Static drain to source on state resistance	$R_{DS(on)}$	—	1.3	1.6	Ω	I_D = 2 A, V_{GS} = 10 V Note 4
Input capacitance	Ciss	—	550	—	pF	V_{DS} = 25 V V_{GS} = 0 f = 1 MHz
Output capacitance	Coss	—	60	—	pF	
Reverse transfer capacitance	Crss	—	10	—	pF	
Turn-on delay time	$t_{d(on)}$	—	15	—	ns	V_{DD} = 200 V I_D = 2 A V_{GS} = 10 V Rg = 25 Ω
Rise time	t_r	—	20	—	ns	
Turn-off delay time	$t_{d(off)}$	—	90	—	ns	
Fall time	t_f	—	30	—	ns	
Total gate charge	Qg	—	13	—	nC	V_{DD} = 400 V V_{GS} = 10 V I_D = 5 A
Gate to source charge	Qgs	—	3.3	—	nC	
Gate to drain charge	Qgd	—	6.6	—	nC	
Body-drain diode forward voltage	V_{DF}	—	0.9	1.5	V	I_F = 5 A, V_{GS} = 0 Note 4
Body-drain diode reverse recovery time	t_{rr}	—	250	—	ns	I_F = 5 A, V_{GS} = 0 V_{DD} = 250 V di_F/dt = 100 A/μs

Note: 4. Pulse test

Notice that this mosfet requires somewhere between 3.5V and 4.5 V to conduct current.

This mosfet won't be very useful as a level shifter, but we can still try it out.

Level Shifting MOSFET

06/14/23
Wesley Lecru

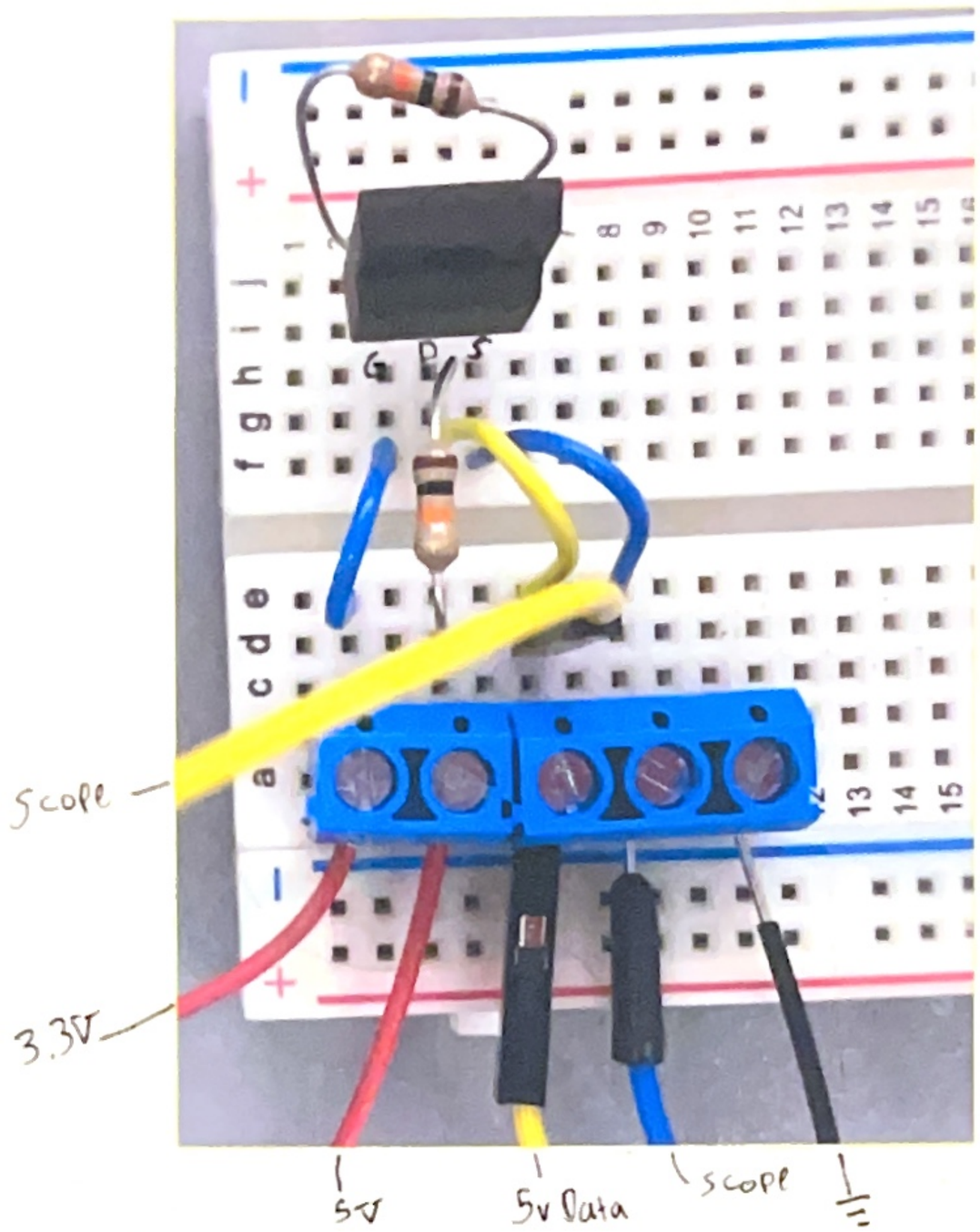
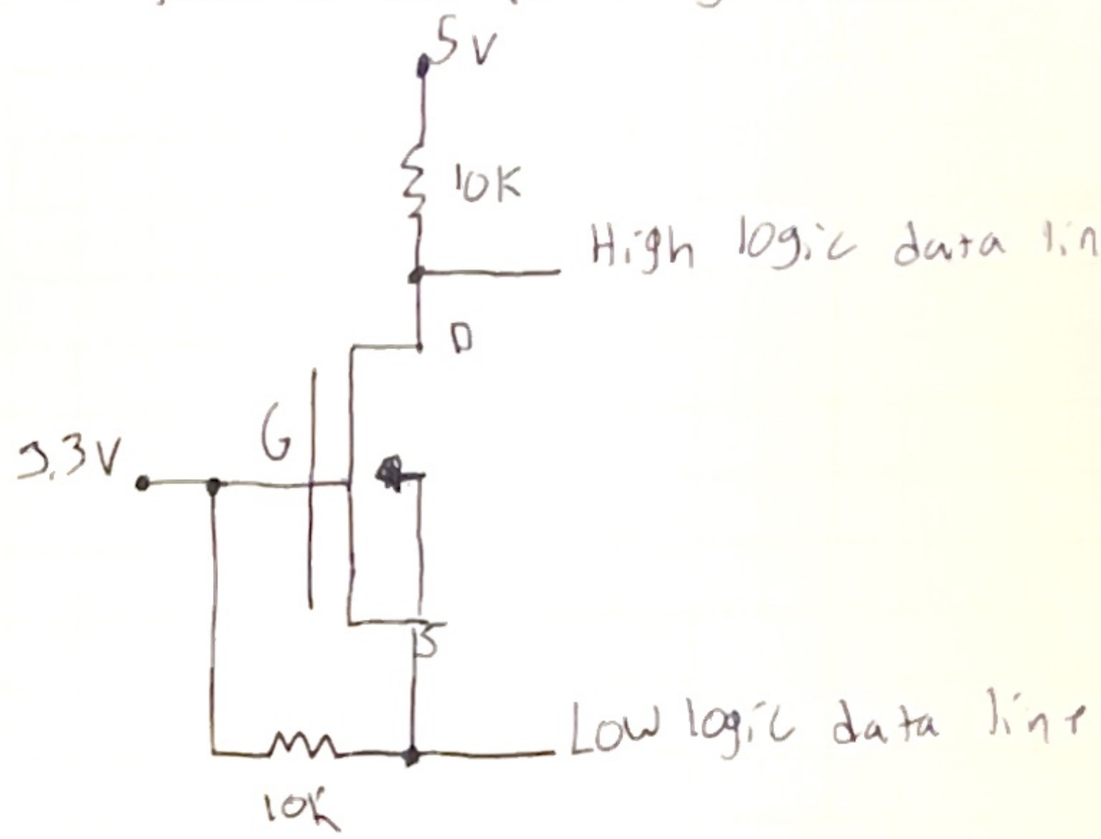


Figure 1
Creating the bidirectional level shifter.

It follows the following circuit diagram:



* This works to convert 5v down to 3.3, but will not work to convert 3.3 up to 5 because of V_{gs} specified by the data sheet.

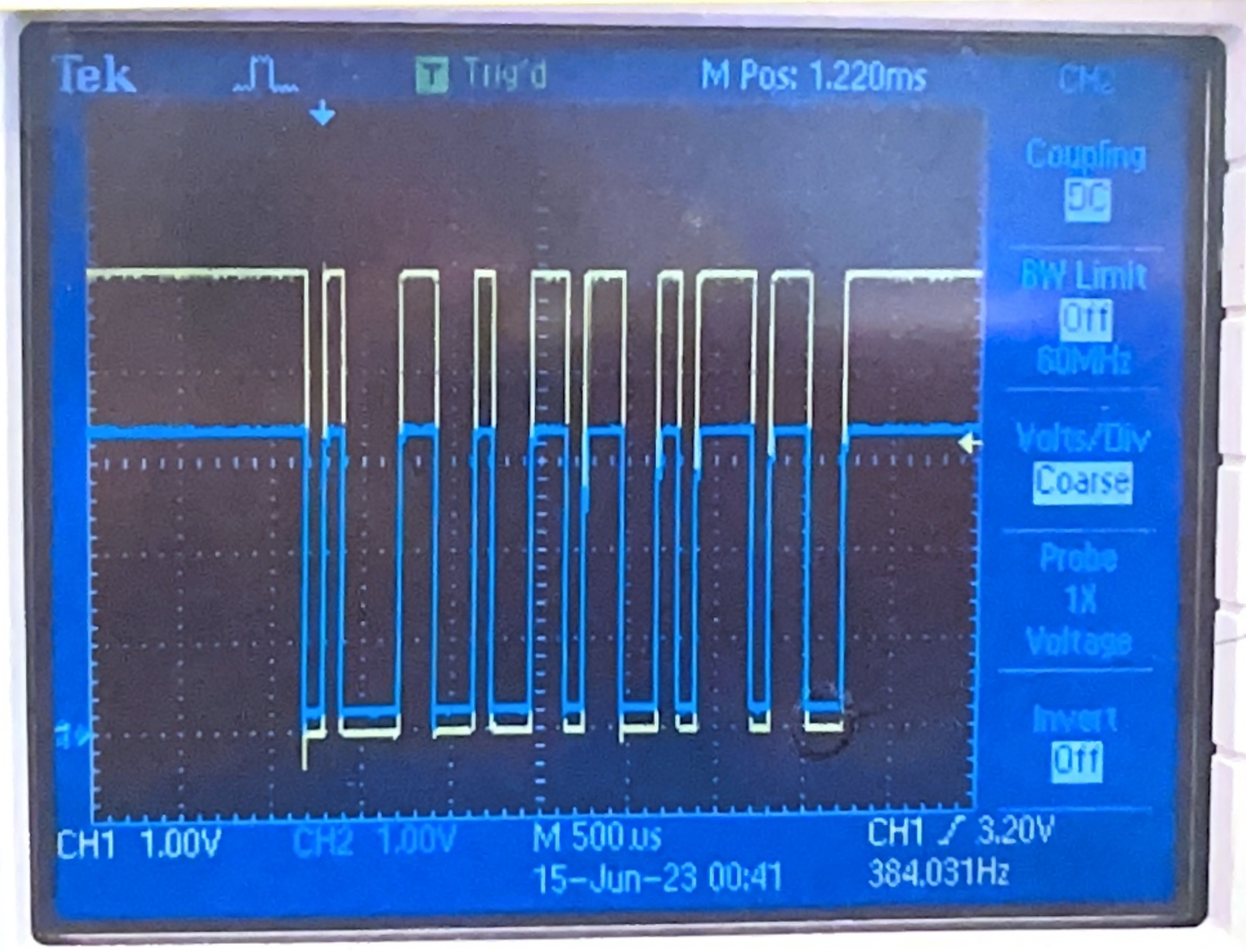
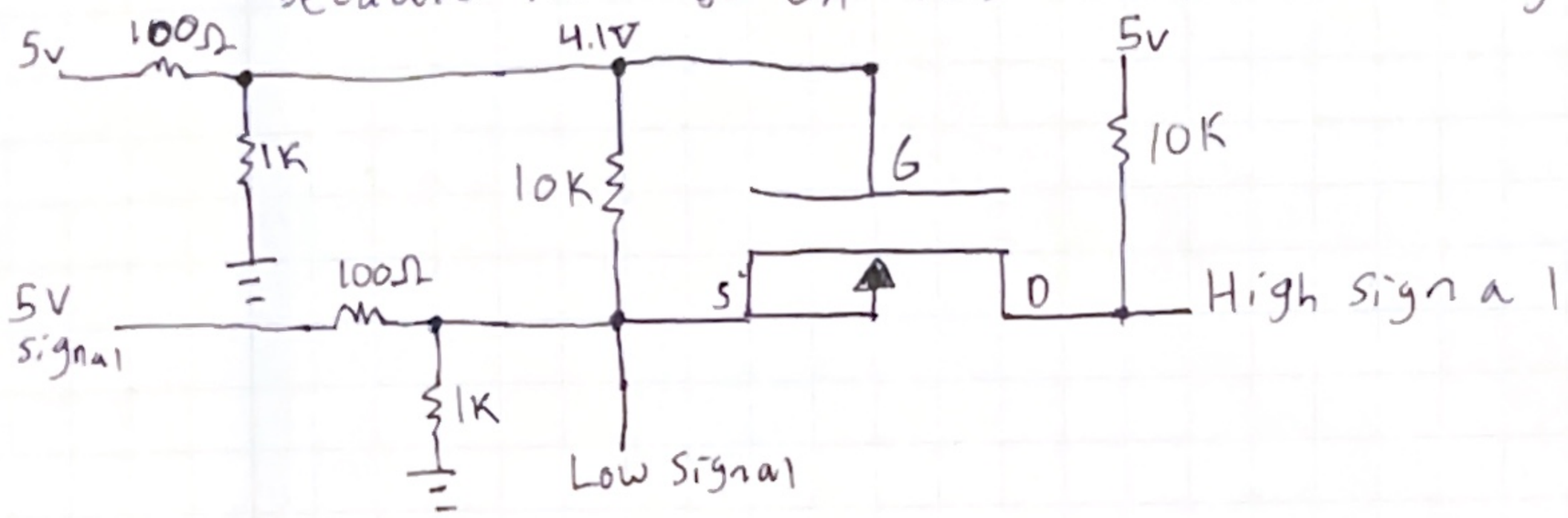
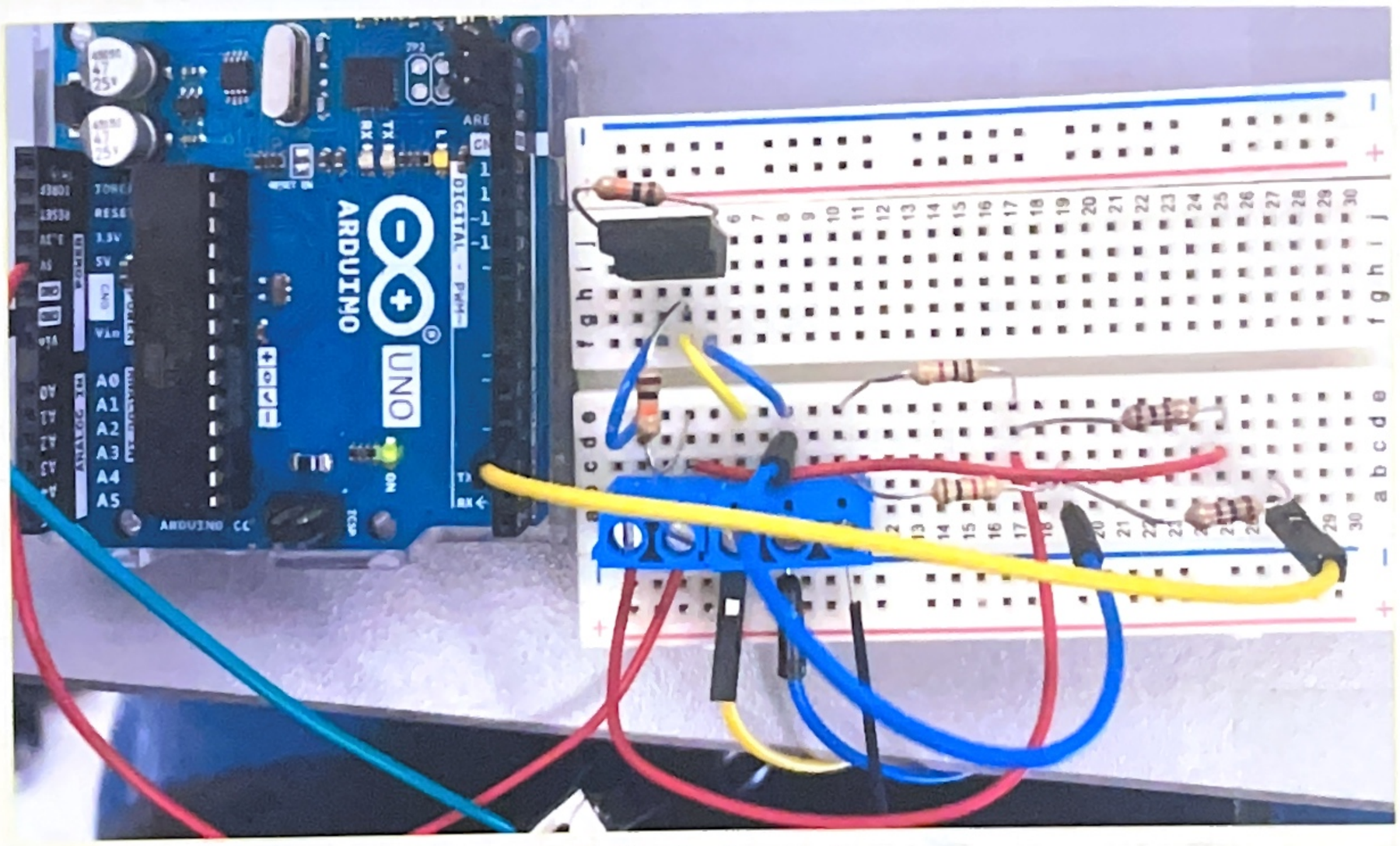


Figure 2
Shifting 5V data down to 3.3V.

Notice the voltage from the substrate diode when the high data line pulls the com low!

To show that the bidirectional feature works, we can first convert the 5v data to around 4.1v data using a voltage divider. Then allow the mosfet to bring it back up to 5v. This is a little bit of a bad example since we already have the 5v data line. However, it should be noted that normally we wouldn't do this. This is only because the V_{gs} on this mosfet is so high.





4.1v 5v 5v data 4.1v data GND

Figure 3

Creating the Circuit to shift a 4.1V signal to 5V.

Level Shifting MOSFET

06/14/23
Wesley Cooke

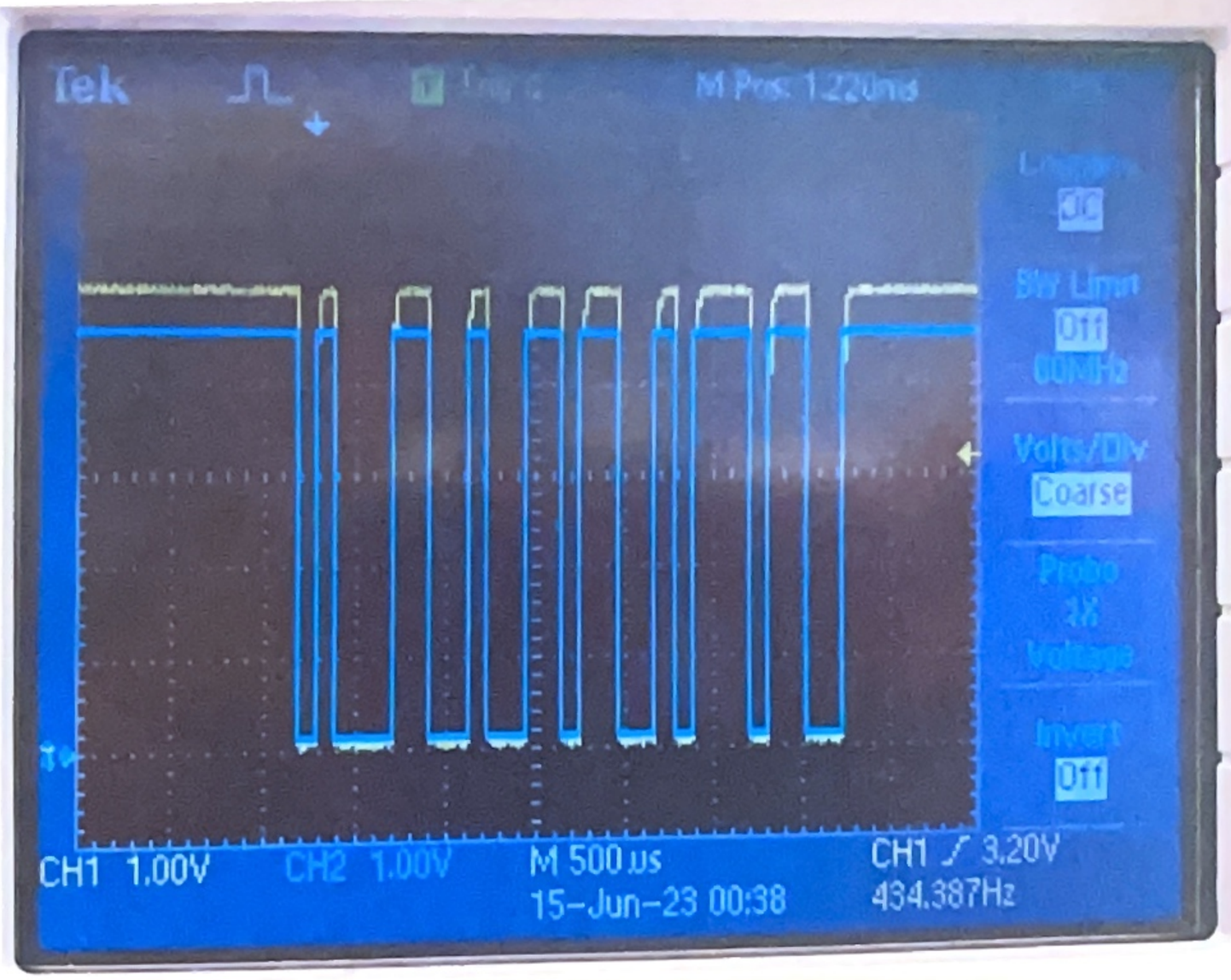
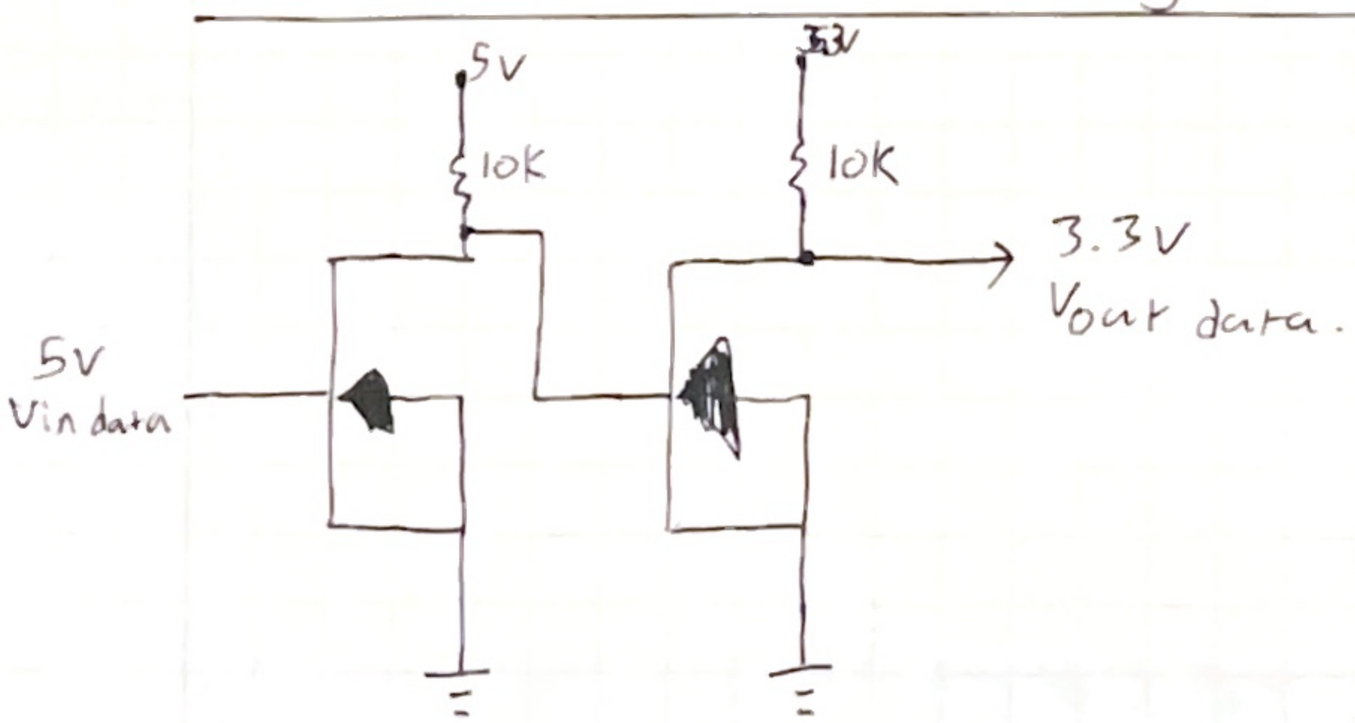


Figure 4
Showing the mosfet can shift a lower signal to a higher one.

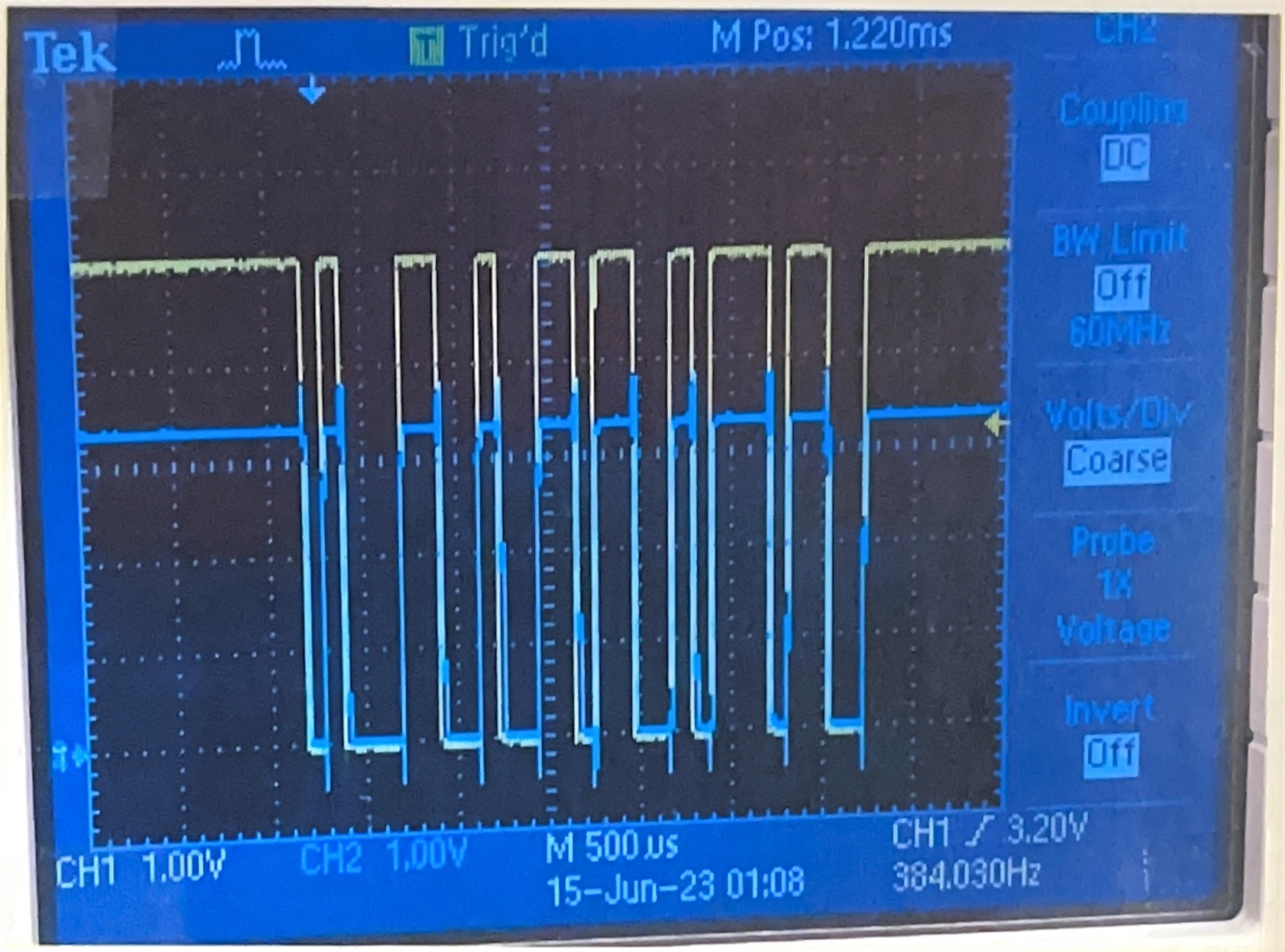
One Direction Level Shifting using two mosFets

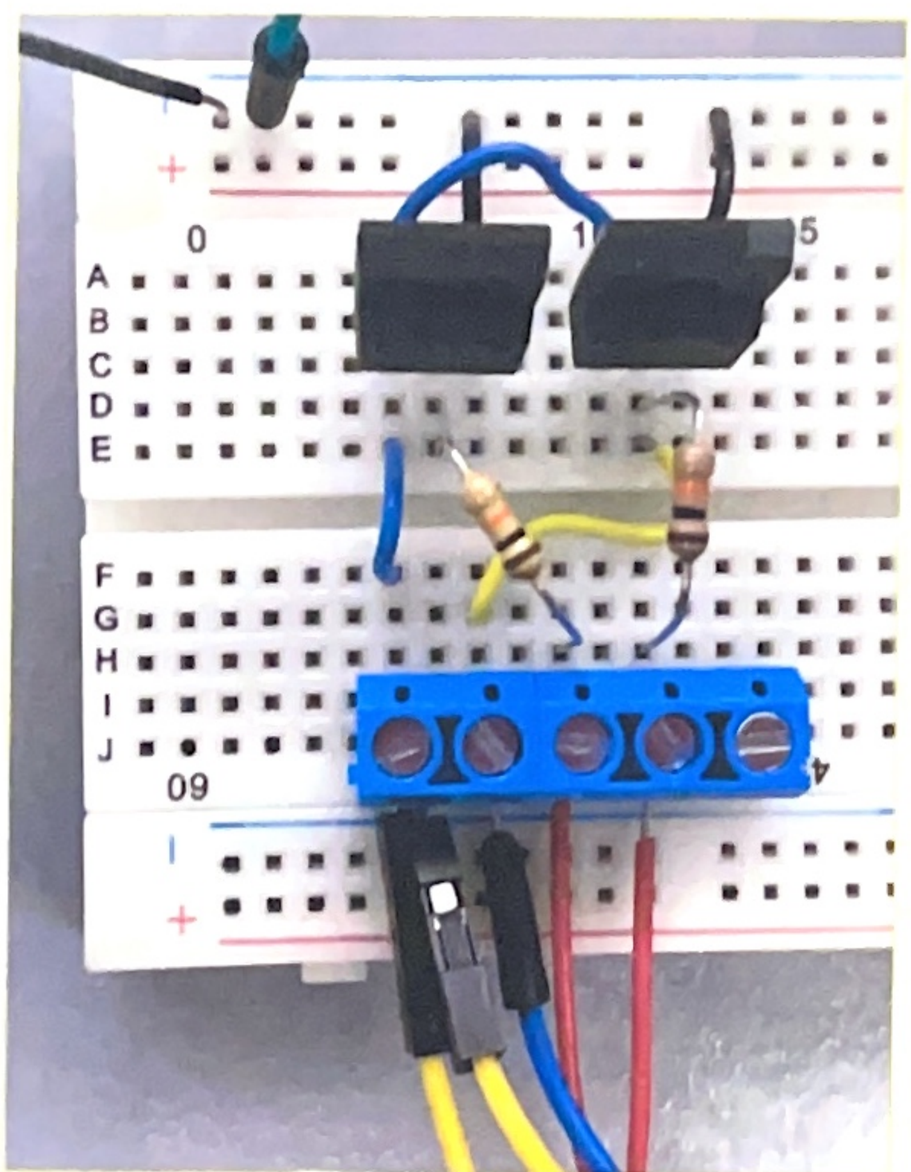


In this circuit, when the Vin data goes high, current is pulled across the 5v and 10K. This will turn off the second mosfet. This makes the 3.3v not drop across the resistor, therefore Vout data is also high.

06/14/23
Wesley Cook

Level Shifting MOSFET





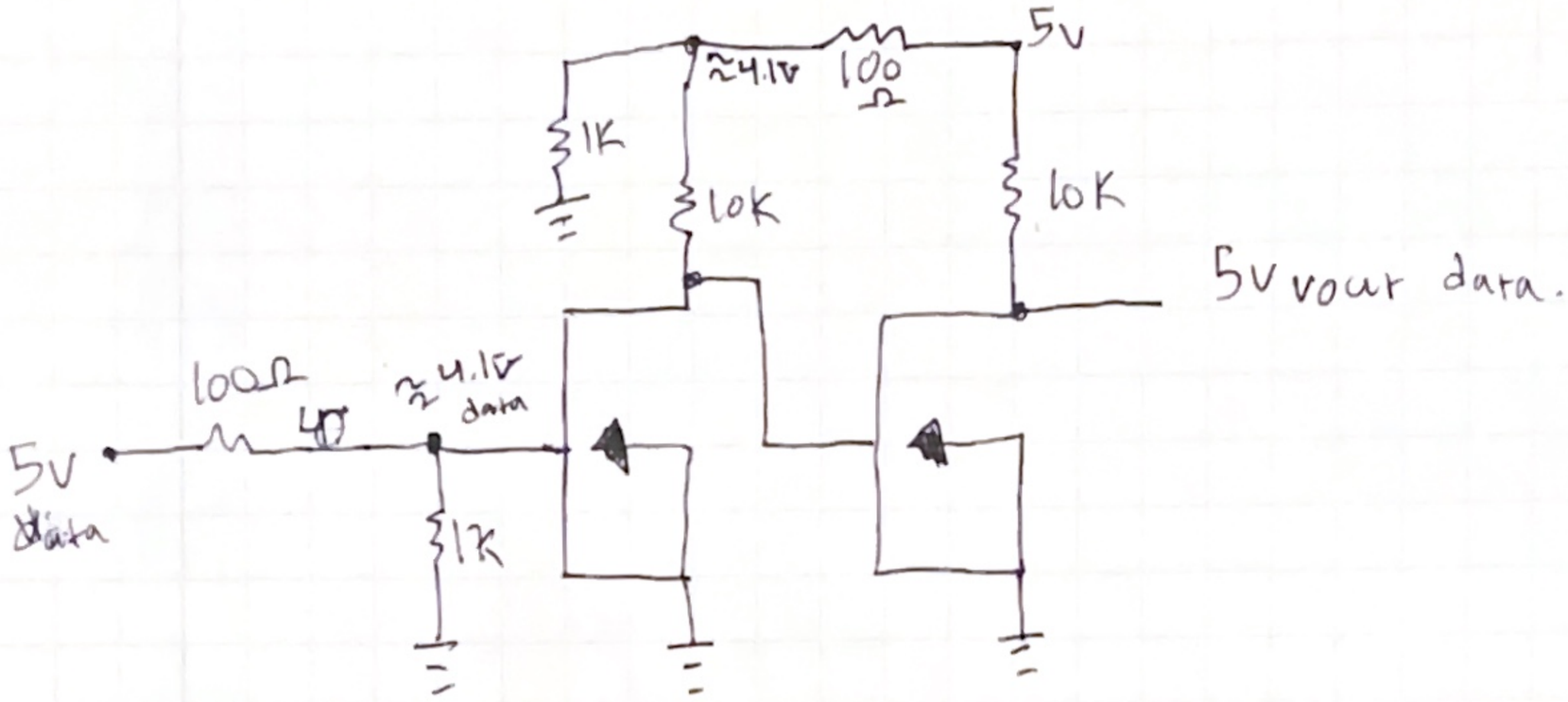
Figures 5 & 6
using two mosfets to level shift.

Level Shifting MOSFET

06/14/23
Wesley Cooper

* One problem with the previous circuit is that if you are trying to convert from a lower voltage to a higher voltage, you are limited by V_{gs}. In this case we need around 4.1V to activate the MOSFET.

To see the shift we can use the same trick as with the bidirectional level shifter. That is to use a voltage divider to convert a 5v signal to 4.1V signal.



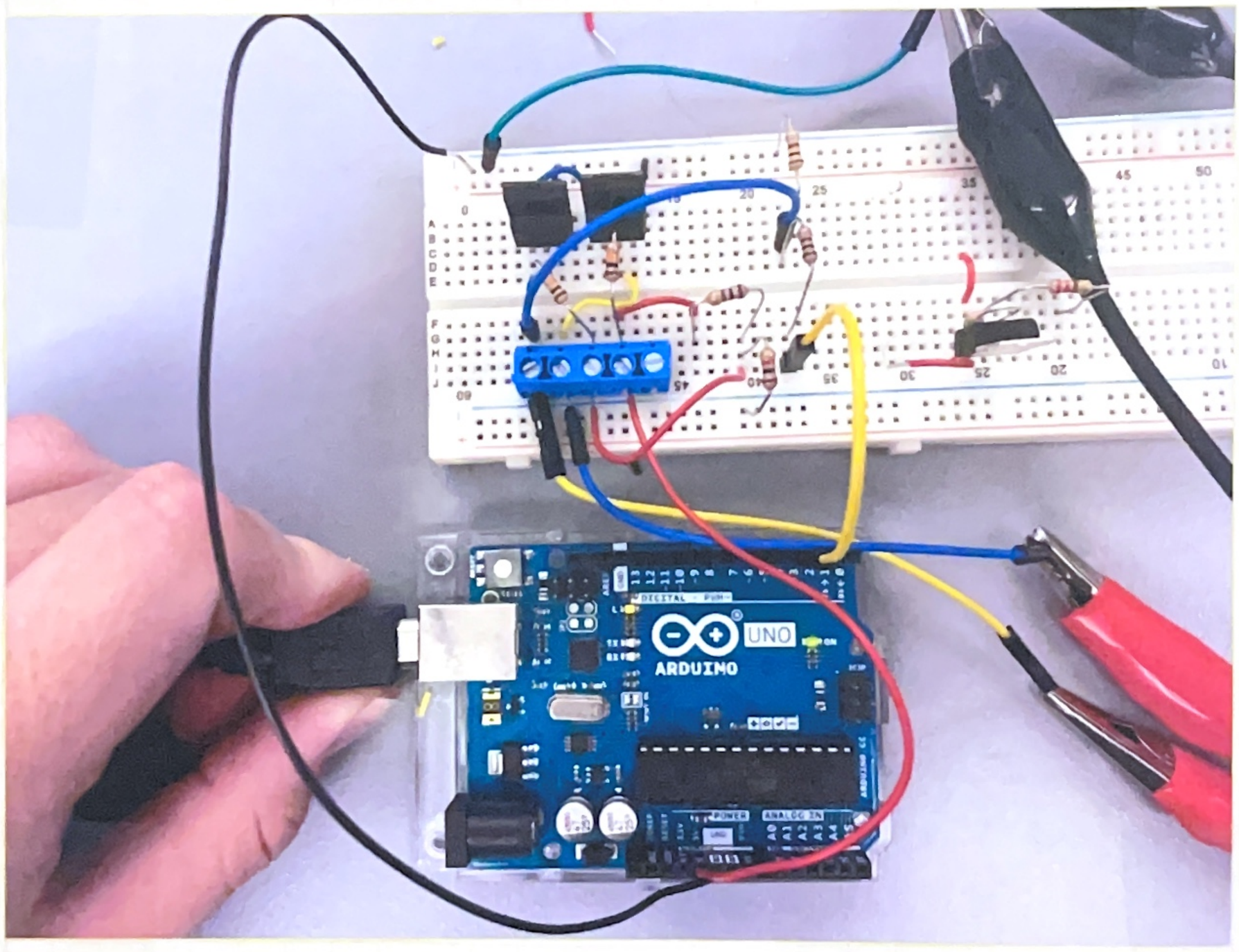


Figure 7
circuit.

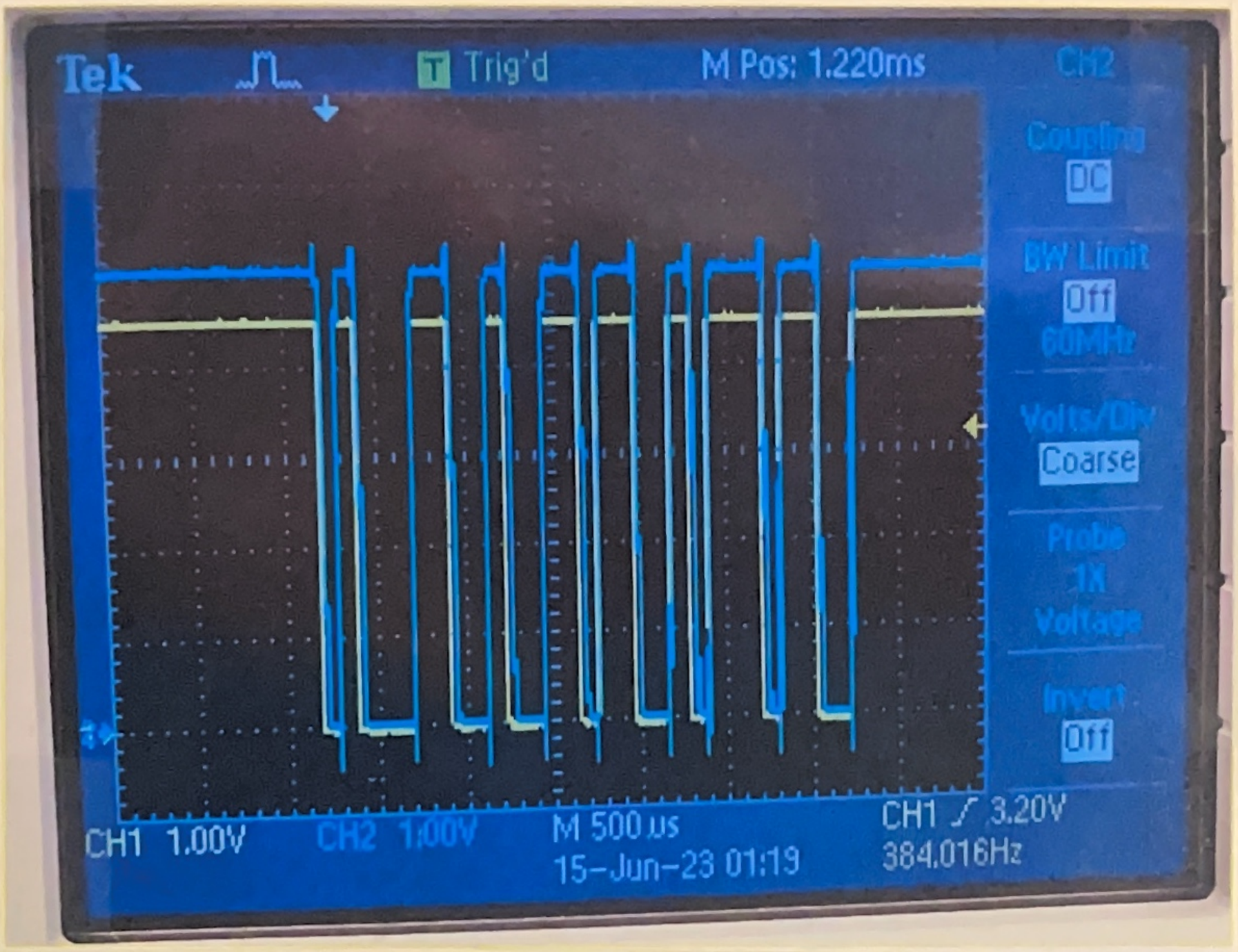


Figure 8

We see the 4.1v data is shifted to 5v.

Conclusions

Level Shifting is possible using the MOSFETs. However, it would be better to us a mosfet that has a lower Vgs. This would allow us a bigger range of shifting.

06/19/2023 Wesley Cadre

Programming the ALS31313 I2C address

Overview

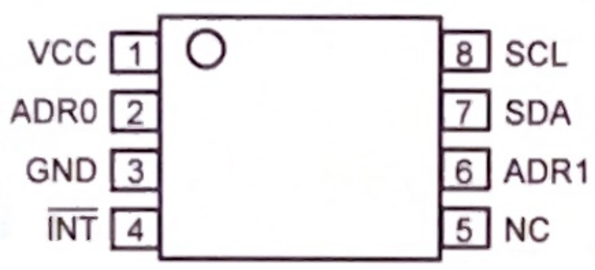
The ALS31313 is a linear hall effect sensor

Objective

Program the I2C address to be 0x41

Pinout and Circuit

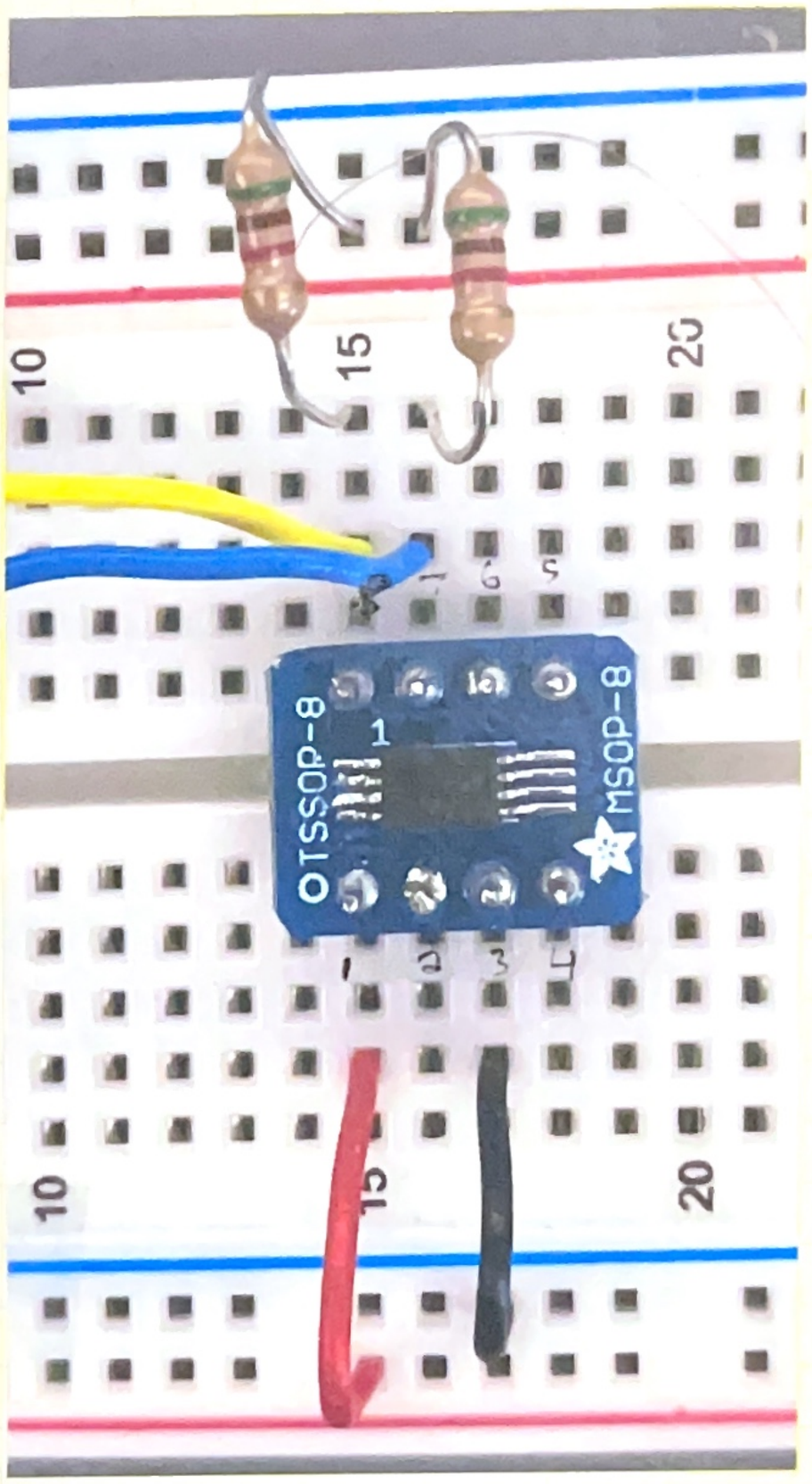
PINOUT DIAGRAM AND TERMINAL LIST TABLE



Package LE, 8-Pin TSSOP Pinout Diagram

Terminal List Table

Number	Name	Function
1	VCC	Power supply input. Bypass VCC to GND with a 0.1 µF capacitor.
2	ADR0	I²C Address Select 0. Connect a resistive divider to ADR0 to select the device address. See Application Information section on addressing for more information.
3	GND	Ground signal terminal.
4	$\overline{\text{INT}}$	Interrupt output. See Application Information section on interrupt function for more information.
5	NC	Not internally connected. Connect to GND.
6	ADR1	I²C Address Select 1. Connect a resistive divider to ADR1 to select the device's address. See Application Information section on addressing for more information.
7	SDA	I²C serial data input/output. Open-drain.
8	SCL	I²C serial clock input



Programming the ALS31313 I2C Address

06/19/2023
Wesley Cain

I²C Addressing Options

I²C Addressing

Table 19 outlines the different addresses available to the ALS31300. In the special case where AD0 and AD1 are both tied to VCC, the device will respond to the slave address stored in register 0x02 (bits 10:16). From the factory, this is set to 7b000000, with the bit following the address indicating a read or write per the I²C specification. Note: Different values for the three MSBs of the address bits (A6, A5, and A4) are available for factory programming if a conflict with other units occurs in the application design.

Table 19: I²C Slave Address Decoding

Voltage on AD1, V_{A1} (× V_{CC})	Voltage on AD0, V_{A0} (× V_{CC})	4-Bit Code from ADR1 and ADR0 Voltages				Slave Address Bits							Slave Address
		E3	E2	E1	E0	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	1	1	0	0	0	0	0	96
	0.33	0	0	0	1	1	1	0	0	0	0	1	97
	0.67	0	0	1	0	1	1	0	0	0	1	0	98
	1	0	0	1	1	1	1	0	0	0	1	1	99
0.33	0	0	1	0	0	1	1	0	0	1	0	0	100
	0.33	0	1	0	1	1	1	0	0	1	0	1	101
	0.67	0	1	1	0	1	1	0	0	1	1	0	102
	1	0	1	1	1	1	1	0	0	1	1	1	103
0.67	0	1	0	0	0	1	1	0	1	0	0	0	104
	0.33	1	0	0	1	1	1	0	1	0	0	1	105
	0.67	1	0	1	0	1	1	0	1	0	1	0	106
	1	1	0	1	1	1	1	0	1	0	1	1	107
1	0	1	1	0	0	1	1	0	1	1	0	0	108
	0.33	1	1	0	1	1	1	0	1	1	0	1	109
	0.67	1	1	1	0	1	1	0	1	1	1	0	110
	1	1	1	1	1	x	x	x	x	x	x	x	Set to 0 at the factory

There are a few ways to set the address on the ALS31313 chip.

The first way is based on the voltage on the ADDR 0 and ADDR 1 pins. In our case we left them open or "0" voltage. In this case our address will be 0b 0110 0000 or 0x60.

The other method involves using a programed address from the memory of the chip. To use this number, set ADDR0 and ADDR1 to high.

06/19/2023 Programming the ALS31313 I2C Address

Wesley Cada

MEMORY MAP

The memory map below lists the locations of accessible registers on the ALS31313. See the following sections on EEPROM and Primary Registers for detailed information.

Reserved	Read Only	Read/Write Volatile	Read/Write EEPROM	Read/Write 1 to Clear	Clear on Read

Table 1: Memory Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x02	RESERVED								BW Select			Hall Mode		I²C CRC Enable	Disable Slave ADC	Slave Address							I²C Threshold	Channel Z Enable	Channel Y Enable	Channel X Enable	INT Latch Enable	Customer EE				
0x03	RESERVED							Signed INT Enable	INT Mode	INT EEPROM Status	INT EEPROM Enable	Z INT Enable	Y INT Enable	X INT Enable	Z INT Threshold					Y INT Threshold							X INT Threshold				
0x0D	RESERVED							Customer EEPROM																								
0x0E	RESERVED							Customer EEPROM																								
0x0F	RESERVED							Customer EEPROM																								
0x27	RESERVED																									Low Power Counter		I²C Loop Mode		Sleep		
0x28	X_Axis_MSBs								Y_Axis_MSBs							Z_Axis_MSBs								New Data	INT	Temperature MSBs						
0x29	RESERVED											INT Write	X_Axis_LSBs				Y_Axis_LSBs				Z_Axis_LSBs				Hall Status	Temperature LSBs						
Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Notice that the slave address is in register 0x02 and occupies bits 16:10. Note that this is split between two bytes.

31:24 23:16 15:8 7:0

~~Byte~~ MSB LSB

00000000 00000000 00000000 00000000

16th bit 10th bit

Programming the ALS31313 I2C Address

06/19/2023
Wesley Cook

EEPROM

The following EEPROM addresses are customer accessible and may be read at any time, with or without entering the customer access code. Customer Access mode must be enabled to write to any of these registers.

Reserved	Read Only	Read/Write Volatile	Read/Write EEPROM	Write 1 to Clear	Clear on Read

Table 2: EEPROM 0x02

Address	Bits	Default	Name	Description
0x02	31:24	0	Reserved	Reserved
	23:21	0	BW Select	Used to control the sample rate of the device. Resolution can be traded for faster samples. 0 = Slowest sample rate, highest resolution 7 = Fastest sample rate, lowest resolution See *Bandwidth Selection* section.
	20:19	See Selection Guide	Hall Mode	Controls the operation mode of the Hall plates. 0 = Single-Ended Hall Mode 1 = Differential Hall Mode 2 = Common Hall Mode 3 = Alternating Hall mode (switches between Differential and Common Hall Modes for each conversion per enabled axis. See *Hall Modes* section.
	18	0	I²C CRC Enable	I²C Cyclic Redundancy Check (CRC) output byte enabled. Enable CRC for applications that require high data integrity. 0 = Disabled 1 = Enabled See *CRC* section
	17	0	Disable Slave ADC	Disable the external slave address pins. When set, the EEPROM setting in Slave Address is used to determine the slave address. See *I²C Addressing* section.
	16:10	0	Slave Address	Used to set the slave address for the device when either Disable Slave ADC is set, or the voltages on the slave address pins are set to V_{CC}. See *I²C Addressing* section.
	9	1	I²C Threshold	Enables 1.8 V or 3 V compatible I²C. 0 = 3 V compatible mode (Increases threshold for logic input high level) 1 = 1.8 V compatible mode
	8	1	Channel Z Enable	Enables the Z channel. Disable for faster update rate if this axis is not needed.
	7	1	Channel Y Enable	Enables the Y channel. Disable for faster update rate if this axis is not needed.
	6	1	Channel X Enable	Enables the X channel. Disable for faster update rate if this axis is not needed.
	5	0	$\overline{\text{INT}}$ Latch Enable	Enables volatile latching of the $\overline{\text{INT}}$ signal. When set, if an interrupt event occurs, the $\overline{\text{INT}}$ status bit and $\overline{\text{INT}}$ output will both remain latched even after the event goes away. See *Interrupt* section.
	4:0	0	Customer EEPROM	Customer non-volatile EEPROM. Can be used to store any customer information. Does not affect device operation.

Notice that to write to the 0x02, we have to use the Customer Access Mode...

Customer Write Access

An access code must be sent to the device prior to writing any of the volatile registers or EEPROM in the ALS31313. If customer access mode is not enabled, then no writes to the device are allowed. The only exception to this rule is the sleep register, which can be written regardless of the access mode. Furthermore, any register or EEPROM location can be read at any time regardless of the access mode.

To enable customer access mode, we have to send ~~a~~ some data to a specific register...

MSB 2C
41
35
LSB 34

To enter customer access mode, an access command must be sent via the I²C interface. The command consists of a serial write operation with the address and data values shown in Table 16. Once the customer access mode is entered, it is not possible to change access modes without power-cycling the device. After power up, there is no time limit to when the access code may be entered.

Table 16: Customer Access Code

Access Mode	Address	Data
Customer Access	0x35	0x2C413534

We can use the following function to set the device to customer access mode.

```
void enableWrite()
{
  // Enable the Customer Access mode from the data sheet

  // Data from the data sheet
  // uint32_t data = 0x2C413534;

  Wire.beginTransmission(devAddr);
  Wire.write(0x35);       // Customer Access Register
  Wire.write(0x2C);       // Customer Access Code
  Wire.write(0x41);
  Wire.write(0x35);
  Wire.write(0x34);
  Wire.endTransmission();
}
```

Programming the ALS31313 I2C Address

06/19/2023
Wesley Cooper

To change the address, we need to set bits 16:10 of Register 0x02.

If we read the register befor changing it contains the following:

0x 00 00 03 C0

To change it to 0x41 we need to set it to

0x 00 01 07 C0

0000 0000 00000001 0000 111 11000000

16th bit 10th bit

To do this, we can use the following method:

```
void changeAddr()
{
  uint32_t data = 0b00000000000000010000011111000000; // 31-24, 23-16, 15-8, 7-0

  Serial.println("Attempting to send...");
  Serial.println(data, HEX);

  Wire.beginTransmission(devAddr); // Tell the chip we are writing to it
  Wire.write(0x02);                // Tell the chip the cusster access address

  Wire.write(data>>24 & 0xFF);
  Wire.write(data>>16 & 0xFF);
  Wire.write(data>>8 & 0xFF);
  Wire.write(data & 0xFF);

  Wire.endTransmission();
}
```

0x 00 01 07 C0

06/19/2023
Wesley Coon

Programming the ALS31313 I2C Address

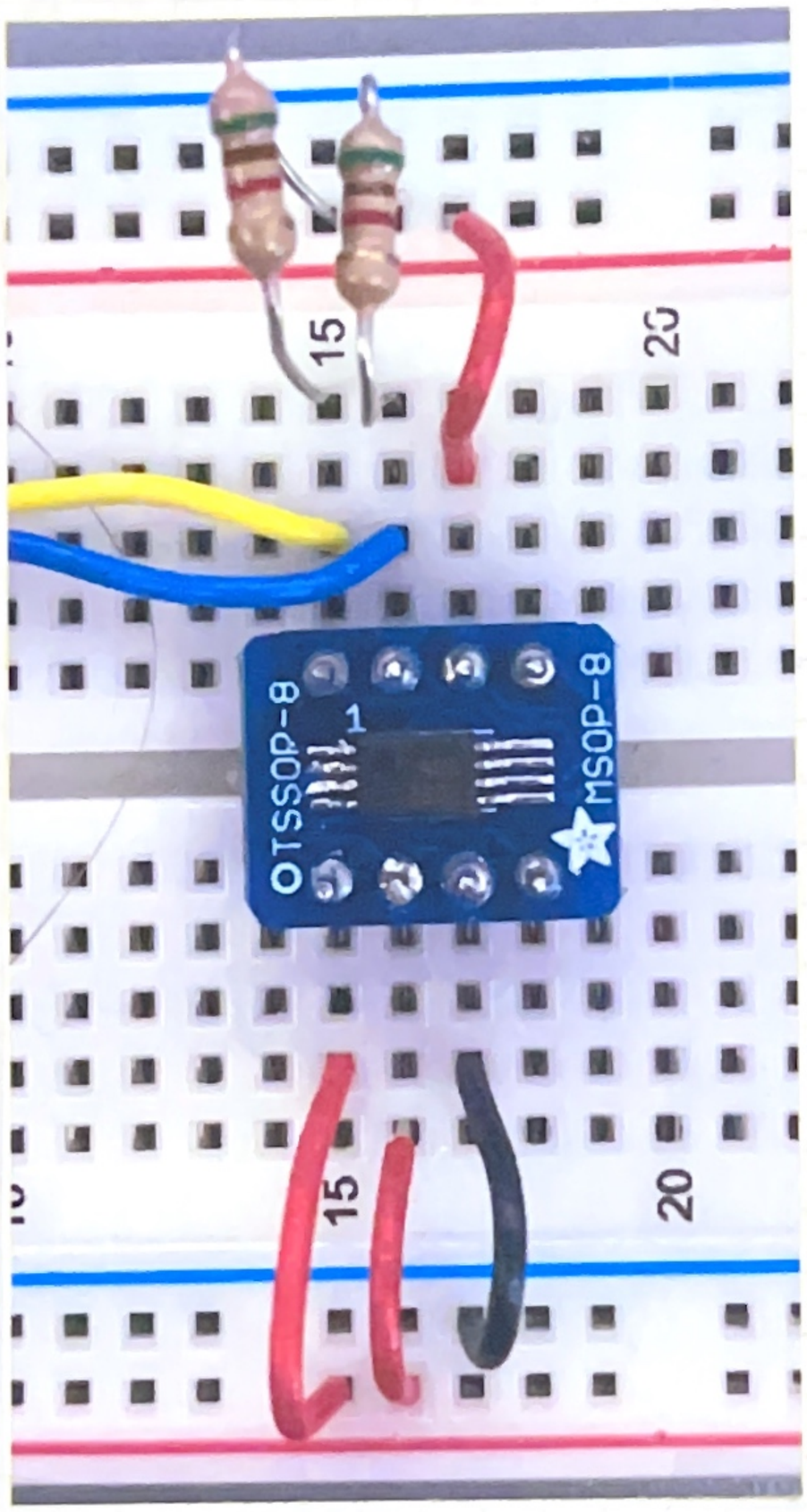

```
#include <Wire.h>

const byte devAddr = 0x60; // From data sheet when ADDR0 and ADDR1 are low
                           // Or use an I2C scanner

void setup()
{
  Wire.begin();          // join i2c bus
  Serial.begin(9600);    // start serial for output
}

void readAddr(byte regAddr)
{

  Wire.beginTransmission(devAddr);     // Communicate with this I2C addr
  Wire.write(regAddr);                 // Go to this register on the chip
  Wire.endTransmission();

  Wire.requestFrom(devAddr, 4);        // Request 4 Bytes from the device

  while (Wire.available()) {         // slave may send less than requested
    byte c = Wire.read();            // receive a byte
    Serial.print(c, HEX);            // print the byte
    Serial.print(", ");
    Serial.print(c, BIN);
    Serial.print(", ");
    Serial.print(c);
    Serial.println();
  }
}
```

```
void loop()
{
  // See what 0x02 holds
  readAddr(0x02);
  Serial.println();

  // Turn on customer access mode
  enableWrite();
  Serial.println();

  // Change the 0x02 register
  changeAddr();

  // See what 0x02 holds now...
  readAddr(0x02);
  Serial.println();

  // Sometimes the first read has all 0's
  // So we read it again
  delay(1000);
  readAddr(0x02);
  Serial.println();

  // End of program
  while(1);
}
```

Programming the ALS 31313 I2C Address

06/19/2023
Wesley Cook

After running the code on the previous page, you should be able to pull ADDR0 and ADDR1 high. Then you should be able to communicate with it using I2C at 0x41.



06/~~19~~ 22/2023
Cullen Cooke

Reyax RYS8839 NMEA parsing and Logging

Overview

The Reyax 8839 is a multi bandwidth GNSS reciever. It uses the L1 and L5 band.

Objective

Create a GPS data logger utilizing the multi bandwidth capabilites of the Reyax 8839.

Figure 1 – circuit

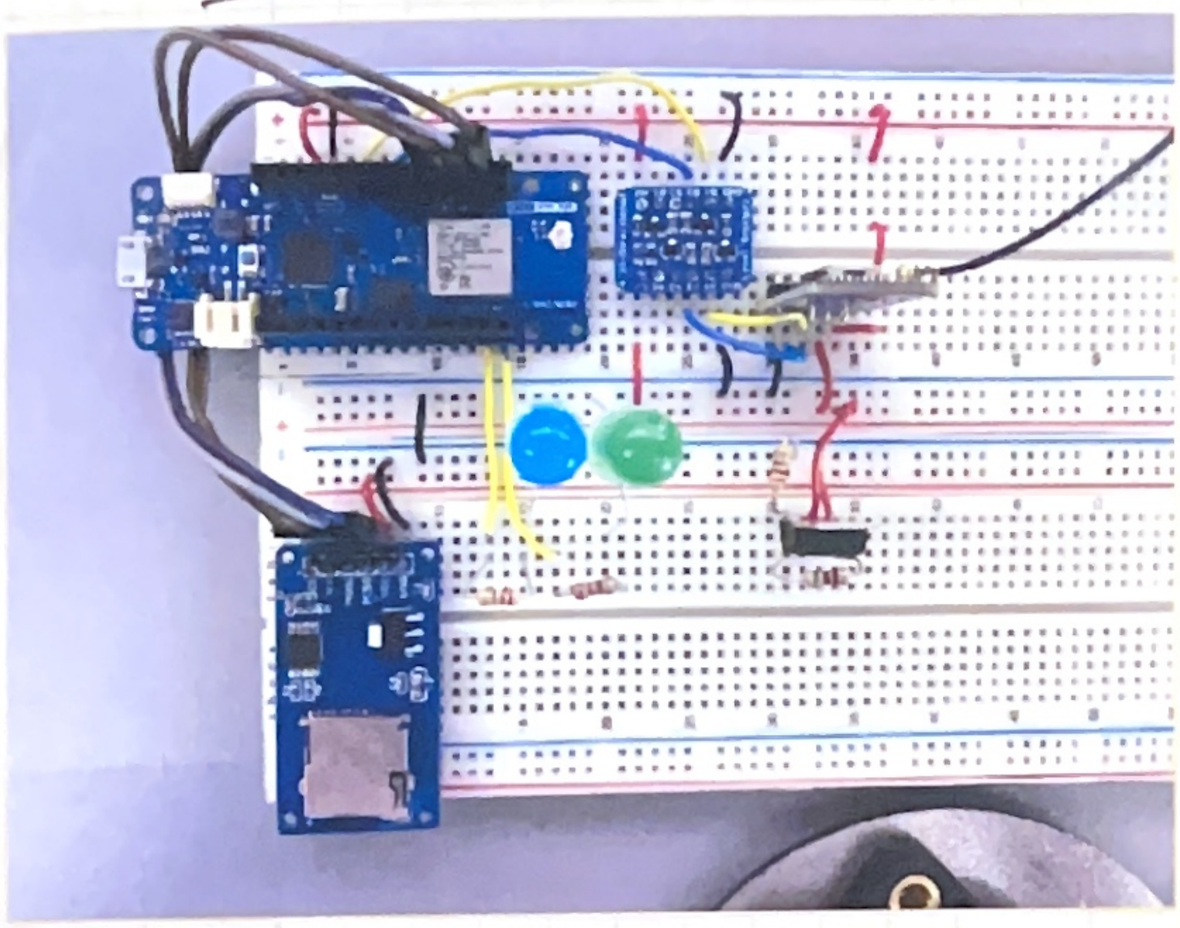


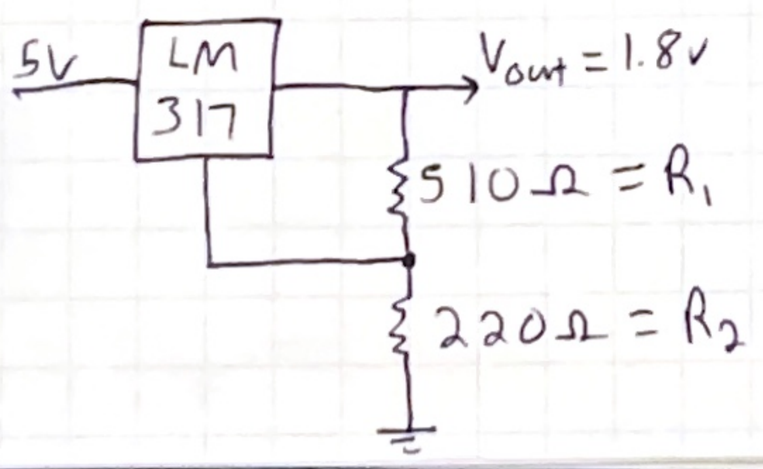
Figure 2 – Antenna

Circuit

The SD card reader is connected via SPI.

The Reyax operates on 1.8 v. To ensure the Rx and Tx of the MKR doesn't damage it, we need a level shifter.

The 1.8V for the level shifter is provided by a LM 317. The power for the 1310 came from an ANKER battery pack.



$$V_{out} = 1.25\left(1 + \frac{R_2}{R_1}\right.$$

$$V_{out} = 1.25\left(1 + \frac{220}{510}\right) = 1.78\,V$$

Reyax Rys 8839 NMEA Parsing and Logging

06/21/2023
W essy
Cooper

The Reyax module uses Commands Similar to AT commands. Here are som examples from their Software guide: reyax.com/products/RYS 8839

3.3.51 @GSTP: Positioning stop

This command is used to stop the positioning. The RYS8839 transfers to the Idle state.

Format: `@GSTP<CR><LF>`

Argument: None

Response:

Sentence	Description
"[GSTP] Done"	This indicates that the command has been executed successfully.
"[GSTP] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

3.3.67 @SBSP: SBAS satellite select

This command is used to select which SBAS satellite to be used when SBAS is used. Only one satellite from

Copyright © 2022, REYAX TECHNOLOGY CO., LTD.

GAGAN, WAAS, EGNOS, MSAS, SDCM and BDSBAS can be selected. The RYS8839 will search just the specified SBAS satellite and not search the other SBAS satellites.

Format: `@SBSP <arg 1><CR><LF>`

Argument:

Field	Description
arg 1	The SBAS satellite is specified. 0: GAGAN (default) 1: WAAS 2: EGNOS 3: MSAS 4: SDCM 5: BDSBAS

Response:

Sentence	Description
"[SBSP] Done"	This indicates that the command has been executed successfully.
"[SBSP] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

06/21/2023 Wesley Gopka

Reyax RYS8839 NMEA Parsing and Logging

3.3.37 @GNS: Positioning-use satellite setting

This command is used to select the satellite systems to be used for positioning.

The satellite systems are assigned to the bits of the argument. "1" is set for the bits of the systems which are to be used and "0" is set for the bits of the systems which are not be used. Arguments can be specified in decimal or hexadecimal notation. With hexadecimal notation, add "0x" in front of the numeral.

This command must be issued at "Idle" mode.

Format: `@GNS <arg 1><CR><LF>`

Argument:

Field	Description
arg 1	The satellite systems used for positioning are set on a bit by bit basis (0: system not used, 1: system used). bit 0: GPS L1-C/A bit 1: GLONASS L1OF bit 2: SBAS bit 3: QZSS L1-C/A bit 5: QZSS L1-S bit 6: BeiDou B1I bit 7: Galileo E1B/C bit 8: GPS L5 bit 9: QZSS L5 bit 10: BeiDou B1C bit 11: BeiDou B2a bit 12: Galileo E5a bit 13: NavIC (Default value: 0x01)

Response:

Sentence	Description
"[GNS] Done"	This indicates that the command has been executed successfully.
"[GNS] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

3.3.33 @GGNS: Acquire the positioning-use satellite setting

This command is used to acquire the positioning-use satellite systems setting by @GNS command. It returns the argument of @GNS.

Format: `@GGNS<CR><LF>`

Argument: None

Response:

Sentence	Description
"[GGNS] Done"	This indicates that the command has been executed successfully.
"[GGNS] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

3.3.5 @BSSL: Output sentence select

This command is used to select the NMEA sentence to be output.

The sentences are assigned to each of the bits of the argument. "1" is set for the bits of the sentences which are to be output, and "0" is set for the bits of the sentences whose output is not required. Arguments can be specified in decimal or hexadecimal notation. With hexadecimal notation, add '0x' in front of the numeral.

Format: `@BSSL <arg 1><CR><LF>`

Argument:

Field	Description
arg 1	Output NMEA sentence bit0 : GGA bit1 : GLL bit2 : GSA bit3 : GSV bit4 : GNS bit5 : RMC bit6 : VTG bit7 : ZDA bit8 : Reserved bit9 : Reserved bit10 : Reserved bit11 : Reserved bit12 : Reserved bit13 : Reserved bit14 : Reserved bit15 : Reserved bit16 : Reserved bit17 : Reserved bit18 : Reserved bit19 : Reserved bit20 : GST (Default value: 0x000000EF)

Response:

Sentence	Description
"[BSSL] Done"	This indicates that the command has been executed successfully.
"[BSSL] Err n"	This indicates that an error has occurred.

3.3.13 @GCD: Cold start

This command is used to start the positioning with cold start. Ephemeris and almanac are erased. Different from @GDCD, time, receiver position and TCXO offset are not erased.

Format: `@GCD<CR><LF>`

Argument: None

Response:

Sentence	Description
"[GCD] Done"	This indicates that the command has been executed successfully.
"[GCD] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

06/21/2023 Wesley Costa

Reyax RYS8839 NMEA Parsing and Logging

3.3.55 @GTIM: Time setting

This command is used to set the time of the receiver in the RYS8839. The UTC time standard is used for the receiver time which employs the format of year, month, day, hours, minutes and seconds.

The receiver position, current time and TCXO offset value are required in order to initiate a hot start so the time must have been set in the RYS8839 prior to hot start using this command.

Format: `@GTIM <arg 1> <arg 2> <arg 3> <arg 4> <arg 5> <arg 6><CR><LF>`

Field	Description
arg 1	This specifies the UTC time (year) using an integer.
arg 2	This specifies the UTC time (month) using an integer.
arg 3	This specifies the UTC time (day) using an integer.
arg 4	This specifies the UTC time (hour) using an integer.
arg 5	This specifies the UTC time (minutes) using an integer.
arg 6	This specifies the UTC time (seconds) using an integer.

Response:

Sentence	Description
"[GTIM] Done"	This indicates that the command has been executed successfully.
"[GTIM] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

3.3.39 @GPPS: 1PPS output setting

This command is used to control 1PPS output.

When 1PPS output is enabled, timing pulse output is 1s period from 1PPS output port. When "1" is set to the argument, the timing pulse outputs after clock information being received from GNSS. When "2" is set to the argument, the timing pulse outputs always during positioning operation. When "3" is set to the argument, the timing pulse outputs only during position is fixed.

When 1PPS output is disabled, timing pulse does not output from 1PPS output port.

Format: `@GPPS <arg 1><CR><LF>`

Argument:

Field	Description
arg 1	1PPS output control 0 : Disable 1PPS output (default value) 1 : Enable 1PPS output (output after clock information is received) 2 : Enable 1PPS output (output always while positioning operation) 3 : Enable 1PPS output (output only during position fix)

Response:

Sentence	Description
"[GPPS] Done"	This indicates that the command has been executed successfully.
"[GPPS] Err n"	This indicates that an error has occurred. "n" is where the error code is entered.

Reyax Rys883a NMEA Parsing and Logging

06/21/2023
Wesley Costa

Once the cold start command is issued, the reyax module will begin outputting ~~NEMA~~ NMEA strings.

To parse the ~~NEMA~~ NMEA sentences, we will use the ~~<Arduino NMEA~~ <ArduinoNmeaParser.h>

Tools → Library Manager → Search "Arduino NMEA"
Install "107-Arduino-NMEA-Parser"

```
#include <SD.h>
#include <SPI.h>
#include <ArduinoNmeaParser.h>

int myIndex = 0;            //What index is our string at?
char mySentence[200];       // Our string has up to 200 chars.
const int chipSelect = 7; // For the SPI SD card.
const int sdLED = 4;
const int fixLED = 5;

void onGgaUpdate(nmea::GgaData const gga)
{
  // This method is called inside of the Arduino Nema Parser
  // when there is new gga data.

  // First we check if there were enough satelities
  // and a good enough signal
  if (gga.fix_quality != nmea::FixQuality::Invalid)
  {
    // Turn on an LED so we can see that the fix is good.
    digitalWrite(fixLED, HIGH);

    // Log some important values from the parser.
    File outputfile = SD.open("data.txt", FILE_WRITE);
    if (outputfile)
    {
      outputfile.print(gga.latitude, 6);
      outputfile.print(", ");
      outputfile.print(gga.longitude, 6);
      outputfile.print(", ");
      outputfile.println(gga.num_satellites);
    }
    outputfile.close();
  }
  else
  {
    // If our fix was invalid, turn off the LED.
    digitalWrite(fixLED, LOW);
  }
}

// This is the parser object
// We pass in the function for onGgaUpdate
// so the parser knows what function to call
// when there is new GGA data. Notice that this
// satement needs to be below the definition of the
// function.
ArduinoNmeaParser parser(NULL, onGgaUpdate);

```

Regax RYS8839 parsing and Logging | 65

06/2/2023 Wyatt Cook

```
void setup()
{
  // These pins are outputs
  pinMode(fixLED, OUTPUT);
  pinMode(sdLED, OUTPUT);
  pinMode(chipSelect, OUTPUT);

  Serial.begin(115200);  // Connection to Serial monitor
  Serial1.begin(115200); // Connection to Reyax Board

  // If the sd card fails to init
  if (!SD.begin(chipSelect))
  {
    Serial.println("Card Failed, or not present... Freezing");
    digitalWrite(sdLED, HIGH); // Show us there was a problem with the SD.
    while(1);
  }
  else { digitalWrite(sdLED, LOW); }

  // Start the file for the data from the parser
  File outputfile = SD.open("data.txt", FILE_WRITE);
  outputfile.println("Latitude, Longitude, NumSats");
  outputfile.close();

  // Start the file for the full nema sentences
  File outputnema = SD.open("nema.txt", FILE_WRITE);
  outputnema.println("====");
  outputnema.close();

  // Reyax Commands

  // Stop Positioning
  Serial1.print("@GSTP\r\n");
  delay(500);
  readResponse();

  // Turn on WAAS
  Serial1.print("@SBSP 1\r\n");
  delay(500);
  readResponse();

  //Set to use all possible sats
  // 0x85 -> GPS L1, GPS L5, and SBAS (WAAS) only.
  Serial1.print("@GNS 0xFF\r\n");
  delay(500);
  readResponse();

  // Get the Satelities being used
  Serial1.print("@GGNS\r\n");
  delay(500);
  readResponse();

  // Turn on only GGA sentences 0x1
  // Turn on GGA, GSA, and RMC 0x25
  // Turn on GLL 0x02
  // Turn on all 0xFF
  Serial1.print("@BSSL 0x000000FF\r\n");
  delay(500);
  readResponse();

  // Inject UTC time for hot start
  Serial1.print("@GTIM 2023 06 28 08 20 00\r\c");
  delay(500);
  readResponse();

  // Enable 1PPS (pulse per second) function
  Serial1.print("GPPS 1\r\c");
  delay(500);
  readResponse();

  // Cold Start
  Serial1.print("@GCD\r\n");
  delay(500);
  readResponse();
}
```

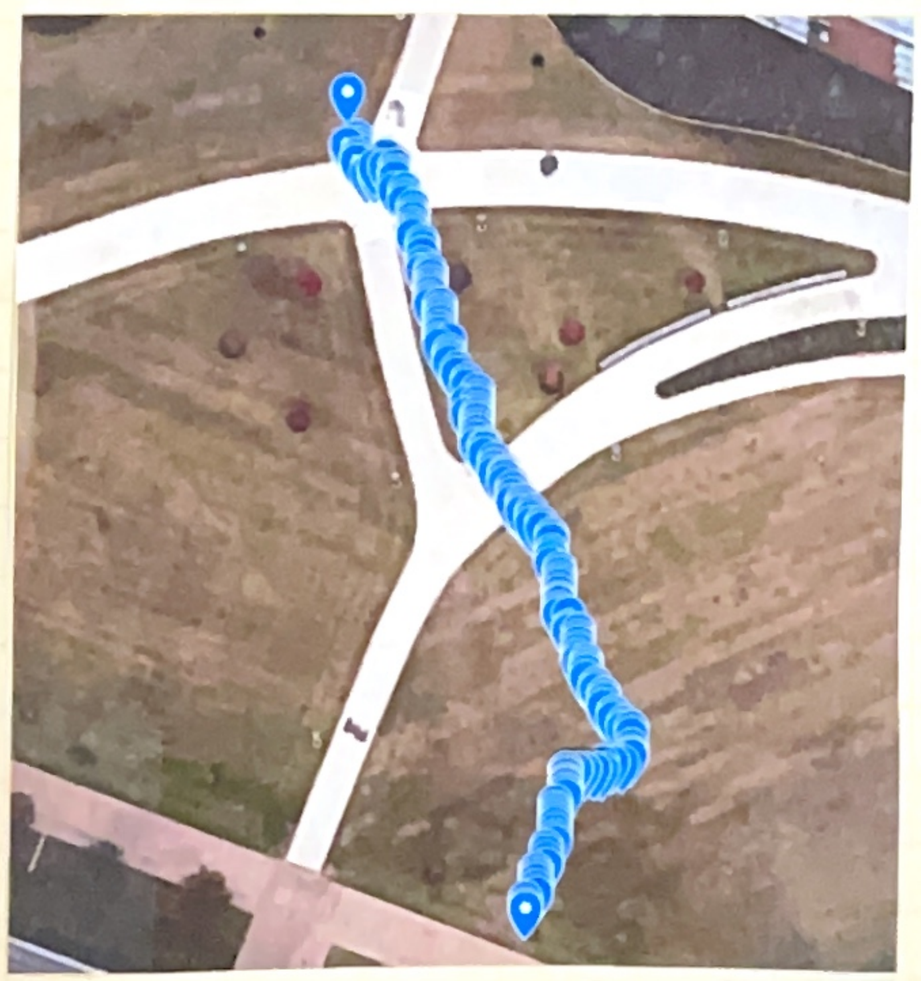
Reyax Rys8839 NMEA Parsing and Logging

06/21/2023
Wesley Cooper

```
void readResponse()
{
  // This method will print out the response
  // of the reyax module to the serial monitor

  // A better method would be to save this
  // in a char array and save it to a file
  // so you can debug if something went wrong in the field.

  delay(10);
  while(Serial1.available()>0)
  {
    Serial.print((char)Serial1.read());
  }
}
void loop()
{
  // While we can read something
  while(Serial1.available())
  {
    // Read the char and pass it to the parser
    char car = (char)Serial1.read();
    parser.encode(car);

    // If we see the start of a new sentence
    if (car == '$')
    {
      // Log the last sentence

      // The last char must be a null char.
      mySentence[myIndex] = '\0';

      // Open the nema file we created in setup
      File outputnema = SD.open("nema.txt", FILE_WRITE);
      if (outputnema)
      {
        // Print the sentence
        outputnema.print(mySentence);
      }
      outputnema.close();

      // Get ready for the next sentence
      mySentence[0] = '$';
      mySentence[1] = '\0';
      myIndex = 1;
    }
    else
    {
      // Otherwise, add this char to the end of
      // our current sentence

      mySentence[myIndex] = car;
      myIndex++;
    }
  }
}
```

06/21/2023
Wesley Costa

Reyax RYS8839 NMEA Parsing and Logging



Figures 3 + 4

Example data plotted onto google maps.

Walked around the side walks outside CSM.

Walked into the grass since this area was a problem area with the DFRobot GPS.

```
33.467503, -81.990936, 25
33.467499, -81.990944, 25
33.467495, -81.990944, 25
33.467491, -81.990952, 25
33.467487, -81.990952, 25
33.467484, -81.990959, 25
33.467480, -81.990959, 25
33.467472, -81.990967, 25
33.467468, -81.990967, 25
33.467464, -81.990967, 25
33.467461, -81.990974, 25
33.467457, -81.990974, 25
33.467453, -81.990974, 25
33.467449, -81.990982, 25
```

Figure 5

Example SD card data

Reyax RYS 8839 NMEA Parsing and Logging

06/21/2023
Wesley Cocka

```
$GNGSA,A,3,11,14,27,28,33,41,43,,,,,,1.1,0.6,1.0,4*G,142,40,14,46,039,42,27,26,317,35,28,76,300,50,1*481,47,33,52,330,42,,,,,,,,7*4E
$GNGNS,201637.00,3328.N,0.8,K,A*1C
$GNZDA,201637.00,28,06,2023,,*76
$GPGGA,201638.00,3328.0569,N,08159.4557,W,1,24,0.6,42.0,M,-31.7,M,,*68
$GNGLL,3328.0569,N,08159.4557,W,201638.00,A,A*6D
$GNGSA,A,3,05,11,13,15,18,23,29,,,,,,1.1,0.6,1.0,1*36
$GNGSA,A,3,65,71,72,85,86,87,,,,,,,1.1,0.6,1.0,2*39
$GNGSA,A,3,07,19,26,33,,,,,,,,,1.1,0.6,1.0,3*3F
$GNGSA,A,3,11,14,27,28,33,41,43,,,,,,1.1,0.6,1.0,35,142,39,14,46,039,43,27,26,317,36,28,76,300,50,1*082,47,33,52,330,41,,,,,,,,7*4E
$GNGNS,201638.00,3328N,0.4,K,A*16
$GNZDA,201638.00,28,06,2023,,*79
$GPGGA,201639.00,3328.0567,N,08159.4555,W,1,24,0.6,42.1,M,-31.7,M,,*64
$GNGLL,3328.0567,N,08159.4555,W,201639.00,A,A*60
$GNGSA,A,3,05,11,13,15,18,23,29,,,,,,1.1,0.6,1.0,1*36
$GNGSA,A,3,65,71,72,85,86,87,,,,,,,1.1,0.6,1.0,2*39
$GNGSA,A,3,07,19,26,33,,,,,,,,,1.1,0.6,1.0,3*3F
$GNGSA,A,3,11,14,27,28,33,41,43,,,,,,1.1,0.6,1.0,35,142,38,14,46,039,42,27,26,317,35,28,76,300,50,1*82,47,33,52,330,41,,,,,,,,7*4E
$GNGNS,201639.00,3328.N,0.6,K,A*15
$GNZDA,201639.00,28,06,2023,,*78
$GPGGA,201640.00,3328.0567,N,08159.4555,W,1,24,0.6,42.1,M,-31.7,M,,*6A
$GNGLL,3328.0567,N,08159.4555,W,201640.00,A,A*6E
$GNGSA,A,3,05,11,13,15,18,23,29,,,,,,1.1,0.6,1.0,1*36
$GNGSA,A,3,65,71,72,85,86,87,,,,,,,1.1,0.6,1.0,2*39
$GNGSA,A,3,07,19,26,33,,,,,,,,,1.1,0.6,1.0,3*3F
```

Figure 6

Example NMEA data

Overview

The ZED-F9P is a multibandwidth GNSS receiver. It uses the L1 and L2 bands.

Objective

Create a gps data logger utilizing the multiple bands of the ZED-F9P.

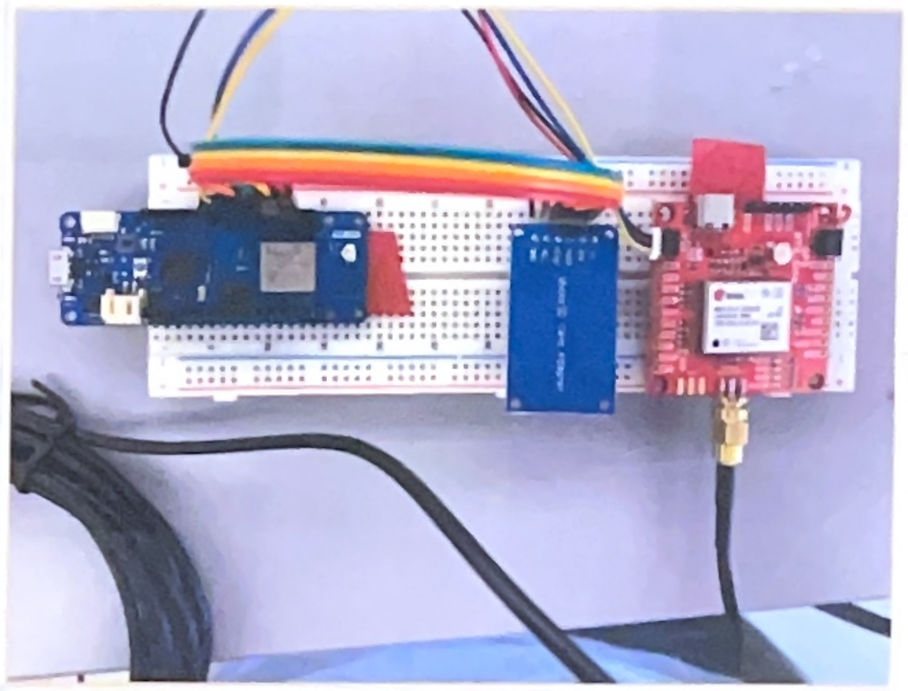


Figure 1

Circuit. The SD card reader is connected via SPI. The Spark fun board is connected via I2C.

The Pinout for the Qwicc connector is:

GND - Top
3.3V
SDA
SCL - Bottom

If using the 3.3V to power the board, it needs to be a very clean source. I chose to power the board using the USB-C cable and an Anker battery pack.

SparkFun ZED-F9P Logging

06/26/2023
Wesley Cooke

Install V3 of the SparkFun u-blox GNSS Library from the Library manager.

```
#include <Wire.h> //Needed for I2C to GNSS
#include <SPI.h>
#include <SD.h>

#include <SparkFun_u-blox_GNSS_v3.h> //http://librarymanager/All#SparkFun_u-blox_GNSS_v3
SFE_UBLOX_GNSS myGNSS;

long lastTime = 0; //Simple local timer. Limits amount if I2C traffic to u-blox module.

const int chipSelect = 7;
const int sdLED = 5;

void setup()
{
  Serial.begin(115200);
  Serial.println("SparkFun u-blox Example");

  pinMode(chipSelect, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);

  Wire.begin();

  if (!SD.begin(chipSelect))
  {
    Serial.println("Card failed, or not present");
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  }
  else
  {
    digitalWrite(LED_BUILTIN, LOW);
  }

  File outputfile = SD.open("data.txt", FILE_WRITE);
  outputfile.println("Latitude, Longitude, SIV");
  outputfile.close();

  //myGNSS.enableDebugging(); // Uncomment this line to enable helpful debug messages on Serial

  if (myGNSS.begin() == false) //Connect to the u-blox module using Wire port
  {
    Serial.println(F("u-blox GNSS not detected at default I2C address. Please check wiring. Freezing."));
    digitalWrite(LED_BUILTIN, HIGH);
    while (1);
  }

  myGNSS.setI2COutput(COM_TYPE_UBX); //Set the I2C port to output UBX only (turn off NMEA noise)
  myGNSS.saveConfigSelective(VAL_CFG_SUBSEC_IOPORT); //Save (only) the communications port settings to flash and BBR
}
```

06/26/2023
Wesley Cooke

Sparkfun ZED-F9P Logging


```
void loop()
{
  //Query module only every second. Doing it more often will just cause I2C traffic.
  //The module only responds when a new position is available
  if (millis() - lastTime > 1000)
  {
    lastTime = millis(); //Update the timer

    long latitude = myGNSS.getLatitude();
    long longitude = myGNSS.getLongitude();
    byte SIV = myGNSS.getSIV();

    File outputfile = SD.open("data.txt", FILE_WRITE);
    if (outputfile)
    {
      outputfile.print(latitude);
      outputfile.print(", ");
      outputfile.print(longitude);
      outputfile.print(", ");
      outputfile.println(SIV);
    }
    outputfile.close();
  }
}
```

Figures 2 + 3

Data from the Sparkfun board plotted on google maps.

06/28/2023
Wesley Caopa

Setting up an RTK Base and Rover System

Overview

The ZED-F9P is able to operate in two modes. One as a base and one as a receiver. In the base mode, the ZED-F9P assumes that it is not moving. This allows it to watch satellites moving overhead and compute a correction value for the data it recieves. This correction value is in the RTCM format.
In the "rover" or receiver mode, the zed-f9p is calculating its own position based on the satellite signals it is receiving. It is able to take in RTCM correction data from various sources that are operating as a base.

Objective

Create a base station that transmits RTCM data over LoRa. Use the RTCM data on a rover device to track GPS locations.

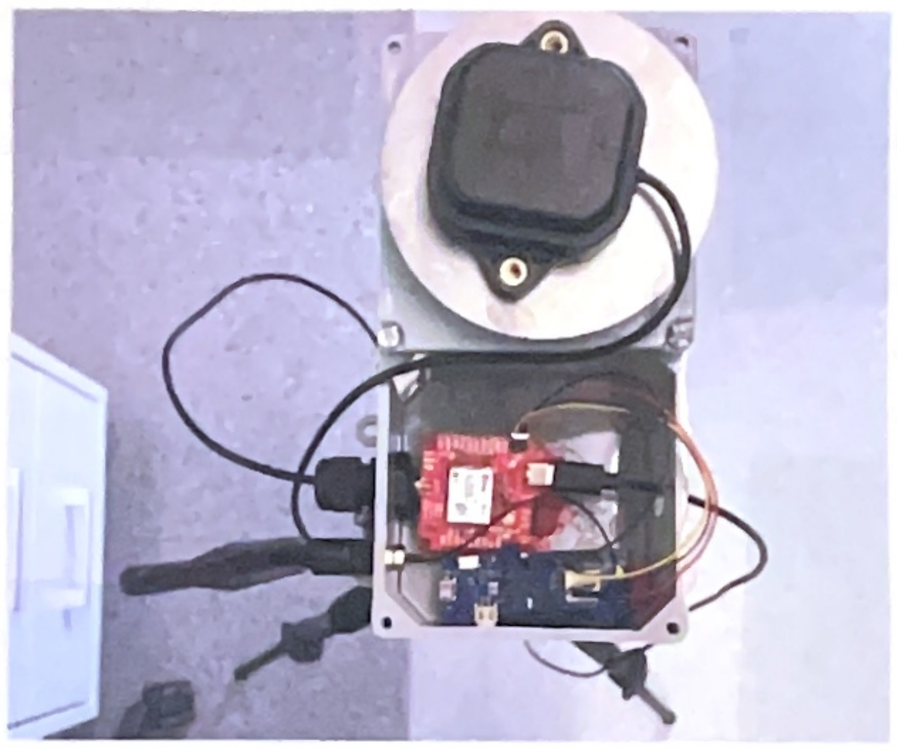


Figure 1
Our base station. This is a ZED-F9P connected to a 1310 via I2C. It sits in a junction box on top of a tripod. The USB connection is used to initiate a Survey-in with u-center on a laptop.

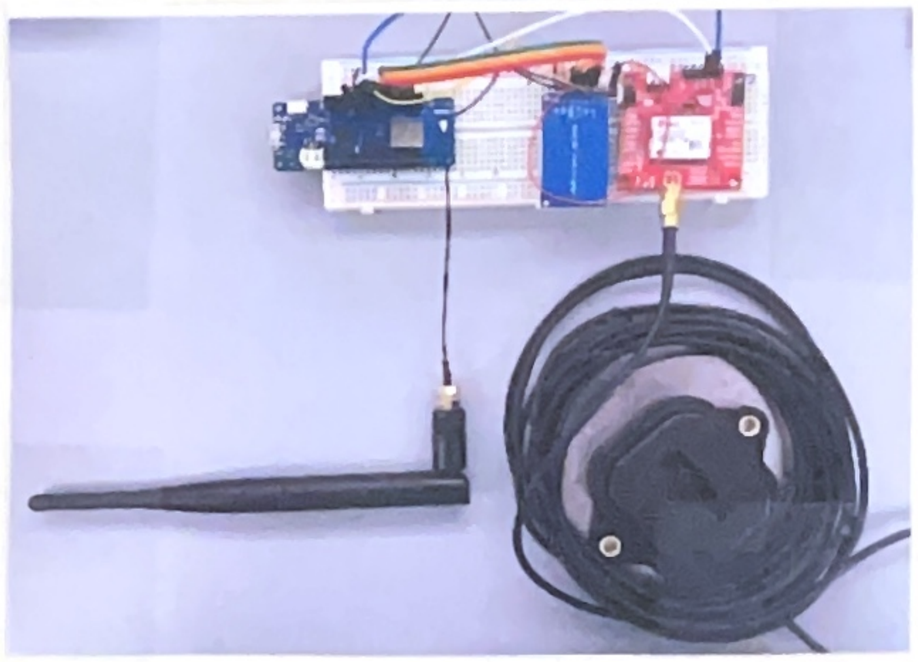


Figure 2
our "Rover" GPS tracker.

The hardware serial is connected to the UART2 of the zed-F9P. The SD card module is using SPI

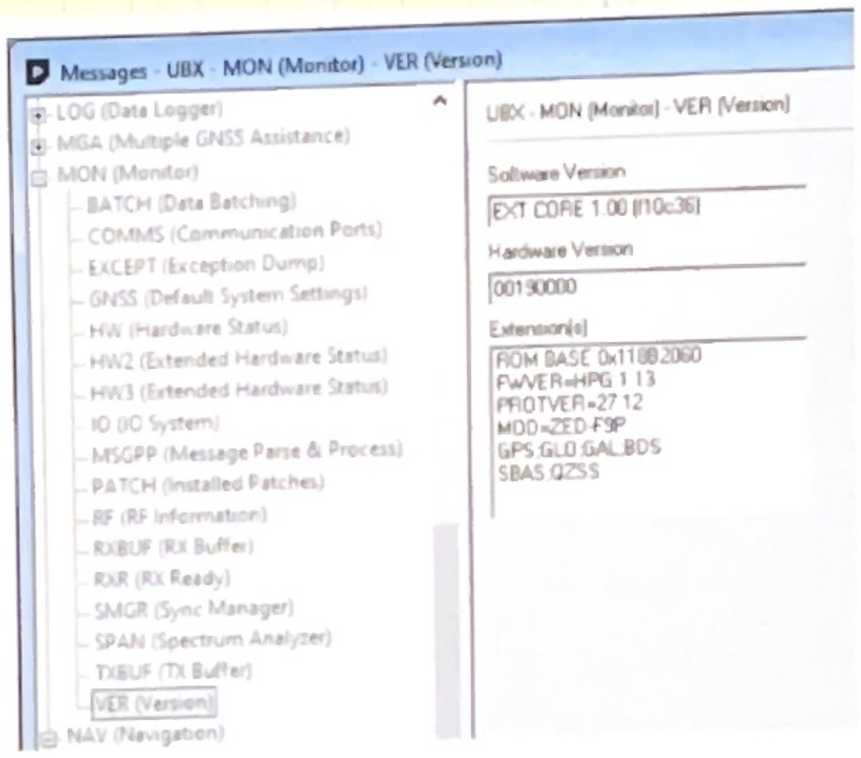
Setting up an RTK Base and Rover System

06/28/2023
Wesley Cooper

To configure the base station, I followed a guide from sparkfun's website. Ive included the peices that I found useful.

Quick note: You will see the terms *GPS* and *GNSS* used interchangeably throughout this tutorial. GNSS, or Global Navigation Satellite System, is the collective term for all GPS (USA), GLONASS (Russia), BeiDou (China), and Galileo (EU) satellites. GPS was the predominant constellation up to about 2017. After this time enough BeiDou satellites were functional and enough GLONASS satellites were transmitting additional bands (L2, etc) that advanced receivers are now designed to receive signals from a variety of constellations rather than *just* GPS. This is a very good thing. With more satellites and more frequencies we can improve accuracies greatly. Said differently, you don't own a *GPS* receiver, you own a *GNSS* receiver. Congrats!

Please make sure both base and rover receivers have the most up to date firmware from u-blox. As of writing, there has been a new v1.30 firmware release. u-blox provides a simple to use upgrade tool inside of u-center. You can find the latest firmware for all u-blox receivers on their website under the documentation and resources tab. u-blox has been making some good improvements for the ZED-F9P from v1.12, to v1.13, to v1.30. We can now receive from EGNOS!



Firmware version 1.13

To check your firmware, open the messages window. Shrink the NMEA branch by double clicking on 'NMEA'. Expand the UBX branch and look for 'MON' or monitor. Expand MON and look for VER (last on the list). FWVER will indicate your current firmware version.

Setting up a Temporary Base

A basic GNSS antenna on a tripod

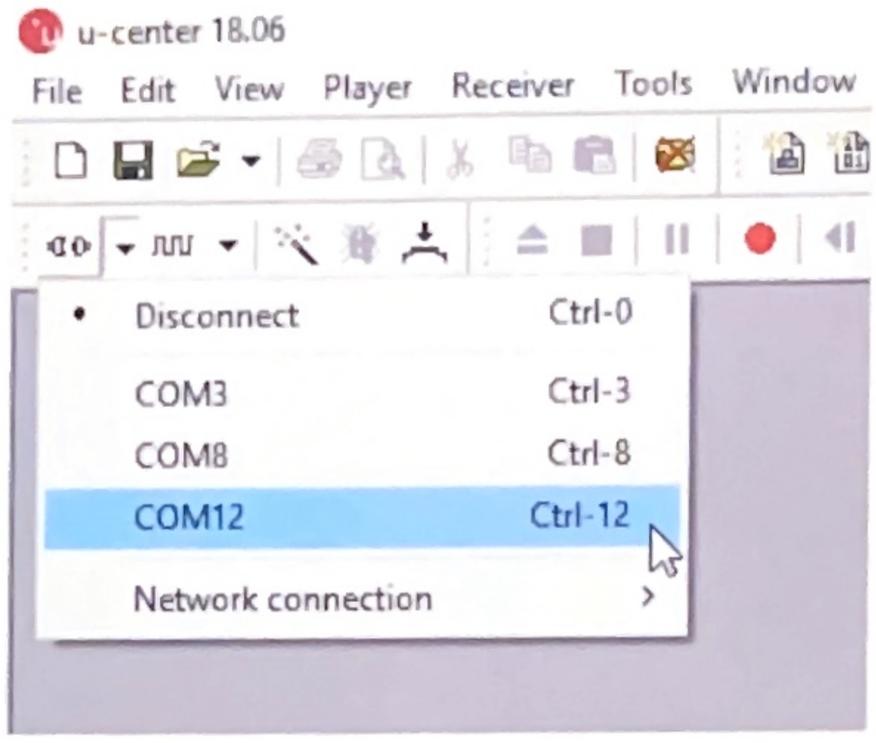
The first step in a base/rover setup is setting up a base station. There are a few ways to set up a base station: a temporary base station is faster but less precise. A static base station takes more time to set up and configure but provides the greatest precision. There is a third option and that is getting your correction data from a public feed. That option is covered in this tutorial and so we will not cover public correction data in this tutorial.

We've covered setting up a temporary base station in a previous tutorial but we will go a bit more in depth here. Additionally, this tutorial will cover the settings required for the ZED-F9P (rather than the previous generation NEO-M8P).

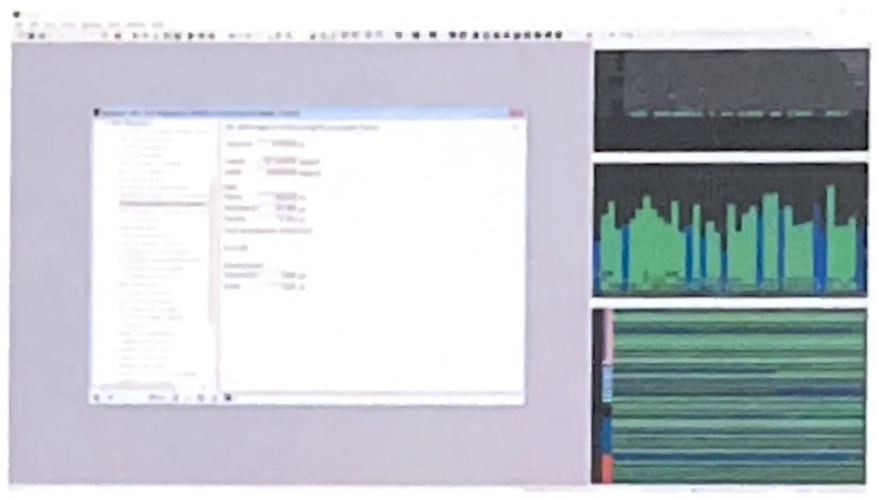
Above is the u-blox antenna attached to the ground plate, on top of a tripod, with SMA cable running indoors to the receiver. This is a great setup for experimenting and for short trips to the field. The purpose of a GNSS RTK base is to not move. Once you tell a receiver that it is a base station (ie, not moving) it will begin to look at the satellites zooming overhead and calculate its position. As the environment changes the signals from the GNSS (Global Navigation Satellite System - the collective term for GPS, GLONASS, Beidou, Galileo satellites) network change and morph. Because the base knows it is motionless, it can determine the disturbances in the ionosphere and troposphere (as well as other error sources) and begin to calculate the values needed to correct the location that the satellites are reporting to the actual location of the base station. These values are unsurprisingly called 'correction values' and are encoded into a format called RTCM. (RTCM stands for Radio Technical Commission for Maritime but is just a name for a standard akin to "802.11". Don't worry about it.). You will often see the term RTCM3 being used; this is simply correction data being transmitted using version 3 of the RTCM format.

Note: The accuracy of the base's location will be reflected in the accuracy of the rover's position. Said differently, if the base is incorrectly recorded 1m away from its actual location than all the rover readings will be exactly 1m wrong. It's really important to locate the antenna with a clear view of the sky and to obtain an accurate base station location.

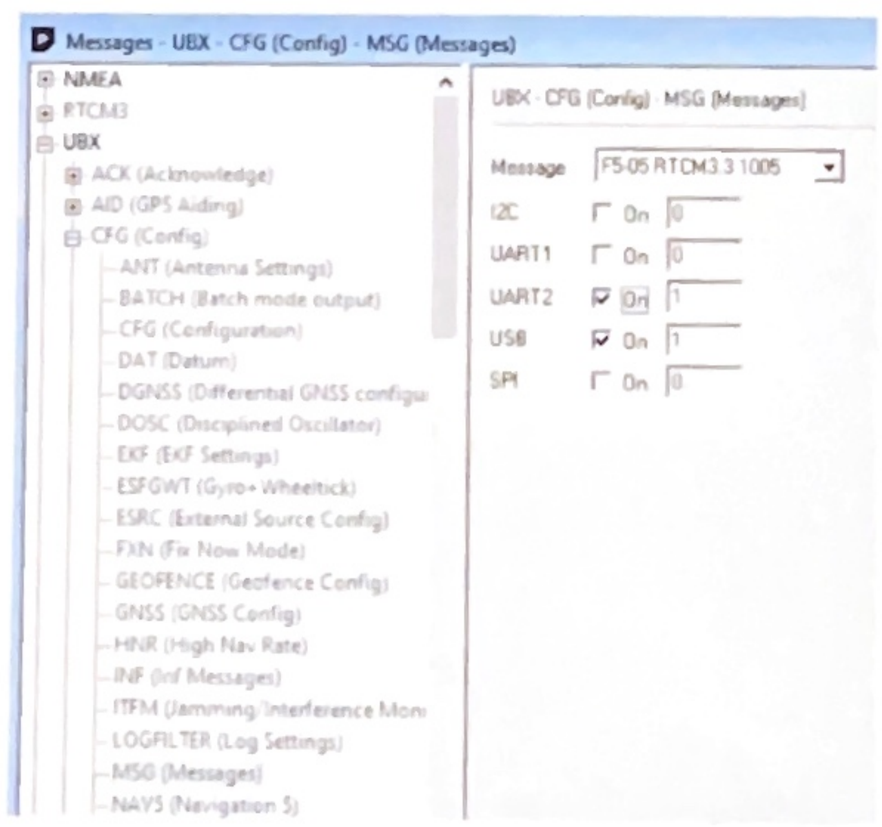
Open u-center and connect to the corresponding COM port that Windows created when the board was connected.



Having Problems With COM Ports? Please checkout our tutorial on u-center specifically. It will walk you through driver install and determining which COM port is the receiver.



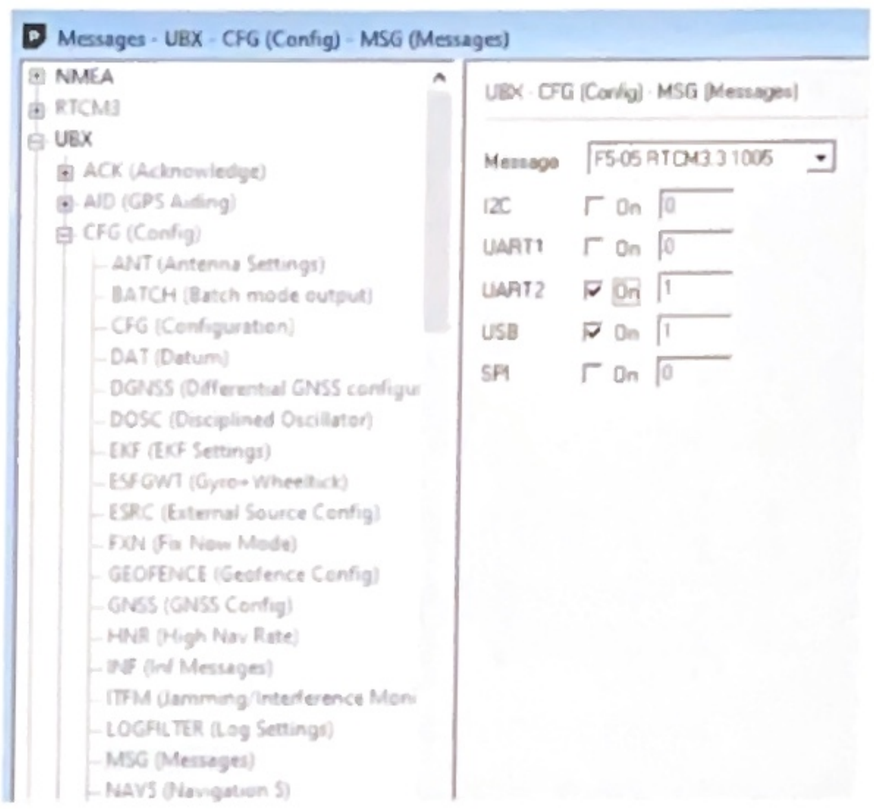
You should begin to see things change and data flowing through. Above shows the ZED-F9P running for a few minutes obtaining a heap of satellites and achieving 0.9m horizontal accuracy. Let's open the Messages window and begin configuring the module.



First, close the NMEA branch to get it out of the way. Open the UBX branch and then the CFG configuration branch. The number of configuration options can be overwhelming at first but over time they can become quite

helpful. We are looking for **MSG**.

Take this moment and ask yourself how you will be transmitting the correction data from the base to the rover. The correction data, or RTCM, varies in size but is approximately 1 to 2k bytes per second. Tiny compared to downloading a song or video, but just large enough that LoRa and smaller packet radios may not be able to handle it. We recommend using a Serial Telemetry Radio. Running at 915MHz, these radios can hit ~300m out of the box at 100mW with a 2km/500mW option available as well. The radios act as a transparent serial passthrough at 57600bps (out of the box). We've found them to be just about perfect for Base to Rover RTCM communication with the only limitation being distance. If you need miles of range, you will need to get your HAM radio license and crank the power. But if you're just getting started with RTK, these radios are great!



We are going to assume a telemetry radio will be attached to UART2 of the ZED-F9P. From u-center Messages window, enable the following messages for both UART2 and USB. *

- RTCM3.3 1005
- RTCM3.3 1074
- RTCM3.3 1084
- RTCM3.3 1094
- RTCM3.3 1124
- RTCM3.3 1230 x 5 (Enable message every 5 seconds)

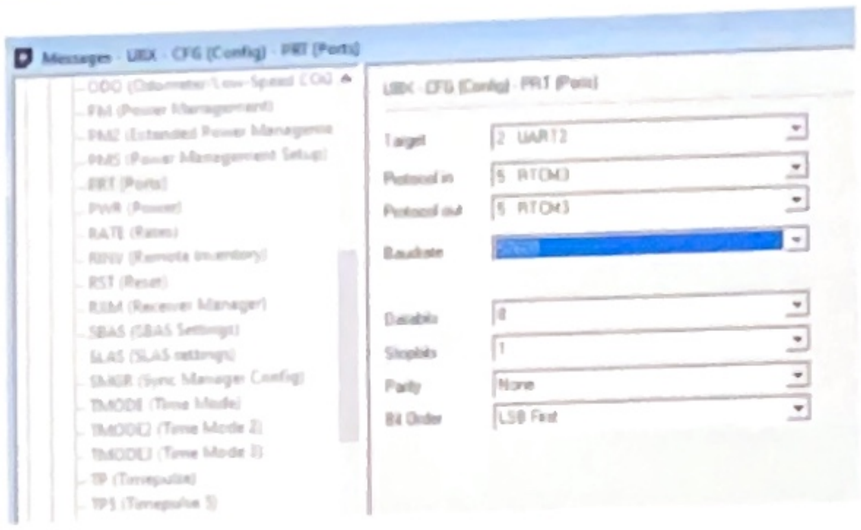
As you set each setting, you will need to press the 'Send' button. It doesn't hurt to press the 'Poll' button to verify the setting has stuck.

We recommend enabling these messages for both USB and UART2 as shown in the photo. This will tell the module to begin transmitting just these RTCM sentences (message types) once the module has completed its survey. UART2 is best used for sending serial RTCM in/out. USB is enabled so that we will be able to see and verify that RTCM messages are actually being output by the module.

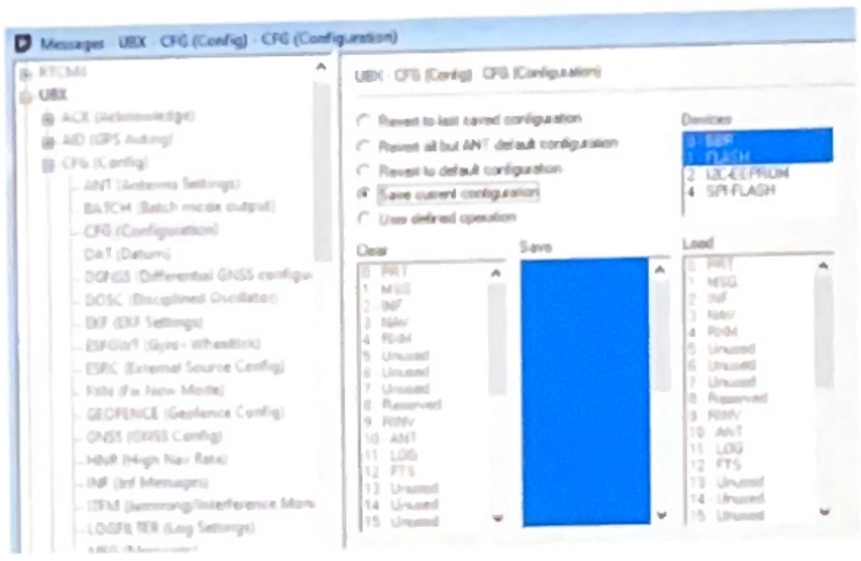
* Instead of using UART2, I enabled the messages on I2C. This means the messages will appear in the I2C traffic. It was more convenient to wire this way.

Setting up an RTK Base and Rover System

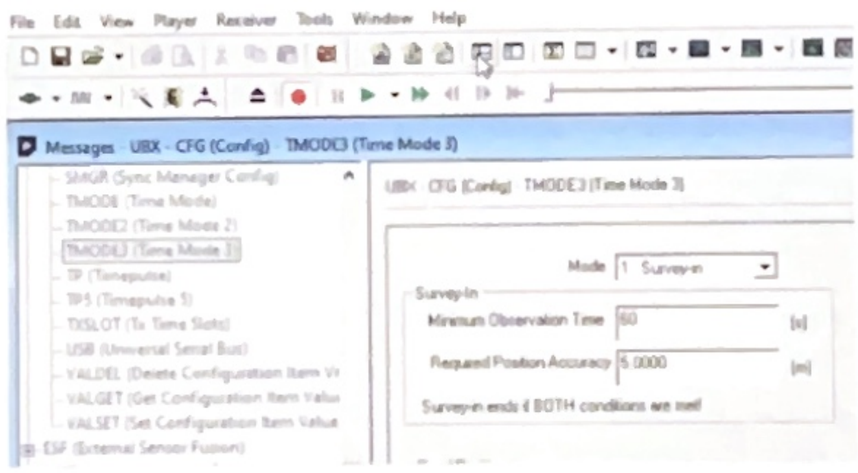
06/28/2023
Wesley Cooke



Next, set UART2 to the appropriate baud rate. Scroll down to the 'PRT' or port settings. Select UART2. By default it is 38400bps and the radios expect 57600bps. Increase the speed. Remember to press 'Send'.

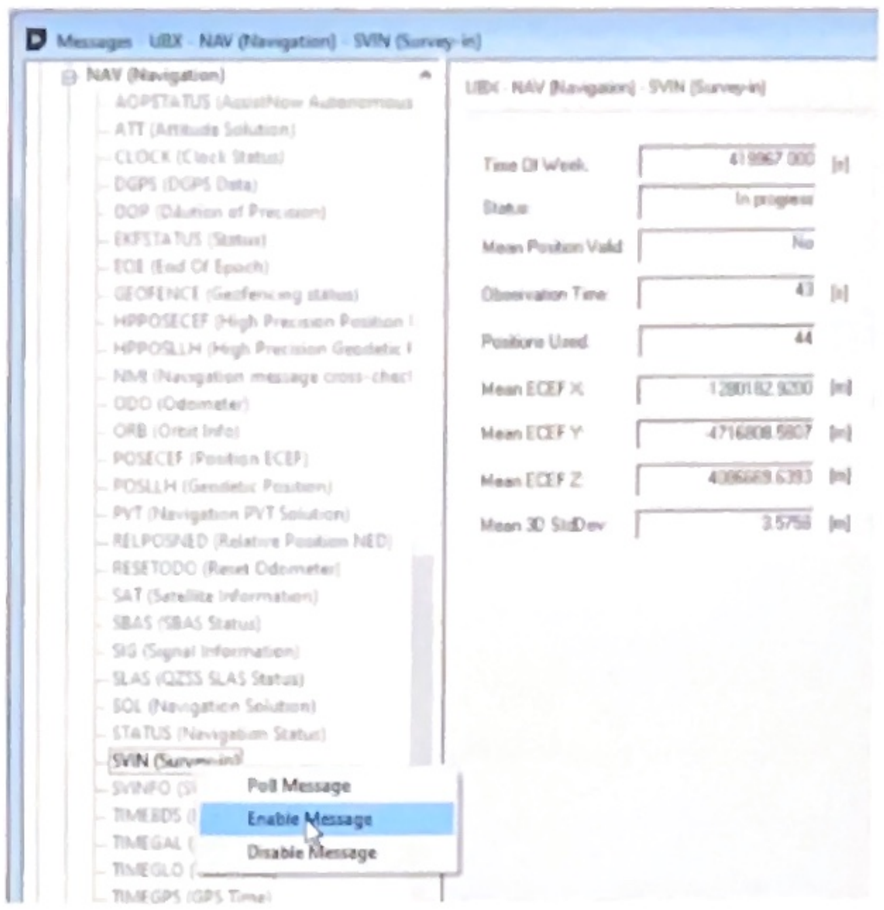


Now scroll up to the CFG or config option. This menu allows us to save the current settings. When you plug in your module again, you won't have to re-do all these settings. *BBR* is battery backed RAM. There is a small battery on the SparkFun GPS-RTK boards that should maintain the settings for over a week. The *Flash* setting records the settings to flash memory as well so if the battery loses power, flash will maintain the settings (also called non-volatile memory or NVM).

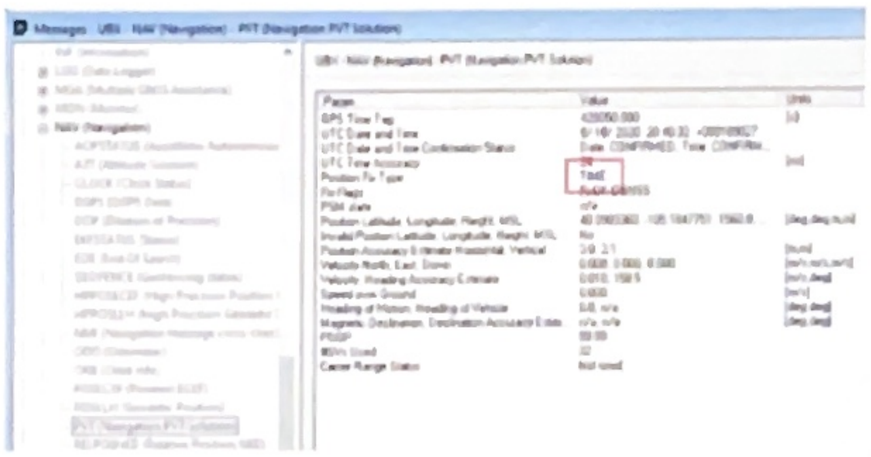


The last thing to do is to initiate a survey in. Navigate to the TMODE3 menu, set the mode to '1 - Survey-In', set the observation time to 60s, and required accuracy to 5m. These are the settings that u-blox recommends. Press 'Send' and sit back. It can take many minutes for the module to obtain enough fixes to have a standard deviation less than 5m.

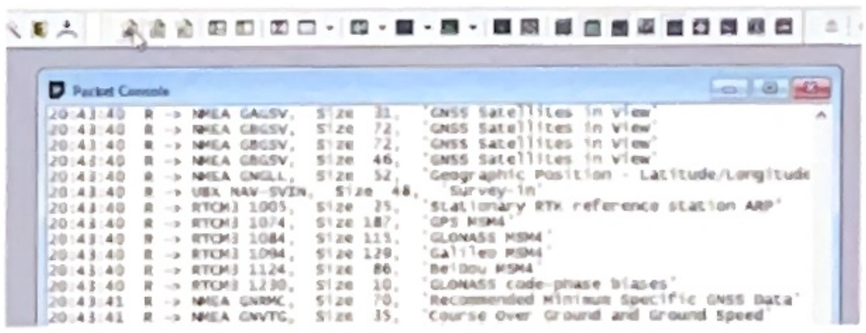
Note: You *can* enter the observation time and required accuracy values *then* save the current settings to BBR/Flash. This will effectively tell your module to survey-in at every power-on. This can be handy in many field deployments where u-center is not available but use it with caution. I have been very confused a few times when my device stopped updating its position because I had left it in survey-in mode and didn't realize it.



To monitor the status of the survey, scroll down, exiting the CFG section and enter the NAV section. Right click on 'SVIN' and click on *Enable Message*. This will tell the module to send the status of this register every second. Once the module has gotten 60s of data *and* the mean standard deviation is less than 5m then the survey will report 'Complete!'. This module's lat/long/height will no longer change because it has been fixed.



You can verify the survey in is complete by checking the PVT message. If the Fix Type is 'TIME' then you know you have successfully setup a base. Congrats!

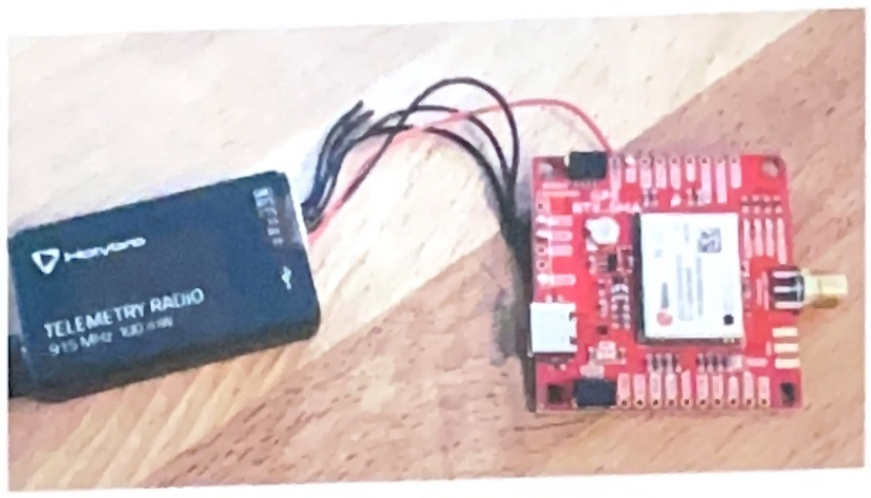


Furthermore, we can view the RTCM messages. Open the packet console. If the RTCM messages have been enabled *and* the survey-in is complete, we should see the RTCM messages scroll by. **Note:** The RTCM messages are not visible ASCII like NMEA sentences, they are encoded bytes of data. You can't 'see' them in the Text Console but they are shown in the Packet Console.

What about fast update rates? The ZED-F9P is capable of outputting up to 30Hz fix rates. This is an astounding amount of math. "Isn't moar, better?"™ Not really for RTK. The base is transmitting correction values once per second. We could increase the output of the module to 4 or 10Hz, but because the conditions between you and the satellites are not changing that quickly, it really only bogs down your radio link.

Setting up an RTK Base and Rover system

06/28/2023
Wesley Cooke



The last thing we need to do is attach our radio. Cut the 6 pin JST-GH cable that came with the Serial Radio Telemetry kit in half. Plug the cable into one radio and note where the TX, RX, 5V and GND pins fall. **Go slow**. I wired TX/RX backwards and caused myself an hour's worth of grief. I recommend stripping one wire at a time as this makes it clear which wire I'm working on. Solder the wires as follows:

- Radio 5V - RTK 5V
- Radio TX - RTK RX2
- Radio RX - RTK TX2
- Radio GND - RTK GND

For the advanced readers of this tutorial, go ahead and wire solder the second radio cable to your rover module in *exactly* the same way.

Shown above, I wired the radio to the 6-pin UART2 header. This space is normally used for the 6-pin connection to a Bluetooth module so if you'd like to leave those pins open (for future potential Bluetooth) you can also solder the radio to the TX2/RX2 pins on the side of the GPS-RTK board. We wired the radio to 5V instead of 3.3V on the 6-pin header. We have seen the radios work at 3.3V but a high voltage should get you the maximum transmit distance.

Now is a good time to mark your module with a sharpie or a 'BASE' label so you know which module is which.

Instead of doing this, we write a LoRa program for the 1310 to forward the RTCM packets.

Setting up an RTK Base and Rover system

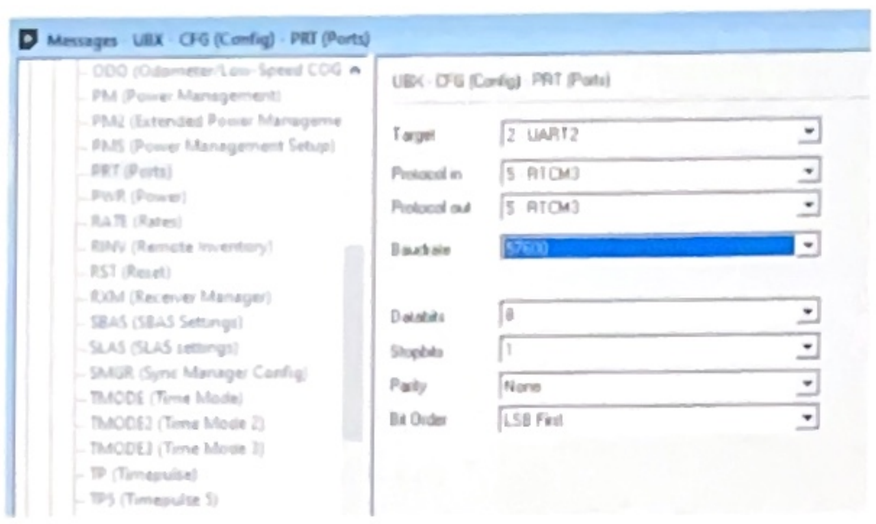
```
#include <Wire.h> //Needed for I2C to GNSS
#include <SparkFun_u-blox_GNSS_v3.h> //http://librarymanager/All#SparkFun_u-blox_GNSS_v3
#include <LoRa.h>;

// ZED-F9P object
SFE_UBLOX_GNSS myGNSS;

int byteIndex = 0;
uint8_t myBytes[500]; // Indexes 0 - 499

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  // Start the LoRa module
  if(!LoRa.begin(915E6))
  {
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  }

  // Start I2C
  Wire.begin();
  Wire.setClock(400000);

  // Make sure we are connected to the ZED-F9P
  if (myGNSS.begin() == false)
  {
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  }
}

void loop()
{
  myGNSS.checkUblox();
  delay(250);
}
void DevUBLOXGNSS::processRTCM(uint8_t incoming)
{
  // Store the incoming bytes until we have enough
  // to make some lora packets

  myBytes[byteIndex] = incoming ; // 499 is the last
  byteIndex++;                    // 499++ is 500

  // If we have filled indexes 0-499.
  // Then we need to make some LoRa packets.

  if (byteIndex == 500)
  {
    // LoRa packets are limited to 255 bytes.

    // First packet. Indexes 0-249
    LoRa.beginPacket();
    for(int i=0; i<250; i++)
    {
      LoRa.write(myBytes[i]);
    }
    LoRa.endPacket();

    // Second Packet. Indexes 250-499
    LoRa.beginPacket();
    for(int i=250; i<500; i++)
    {
      LoRa.write(myBytes[i]);
    }
    LoRa.endPacket();

    // Start storing new bytes at the beginning of the array
    byteIndex = 0;
  }
}
```

08/28/2025 writing code

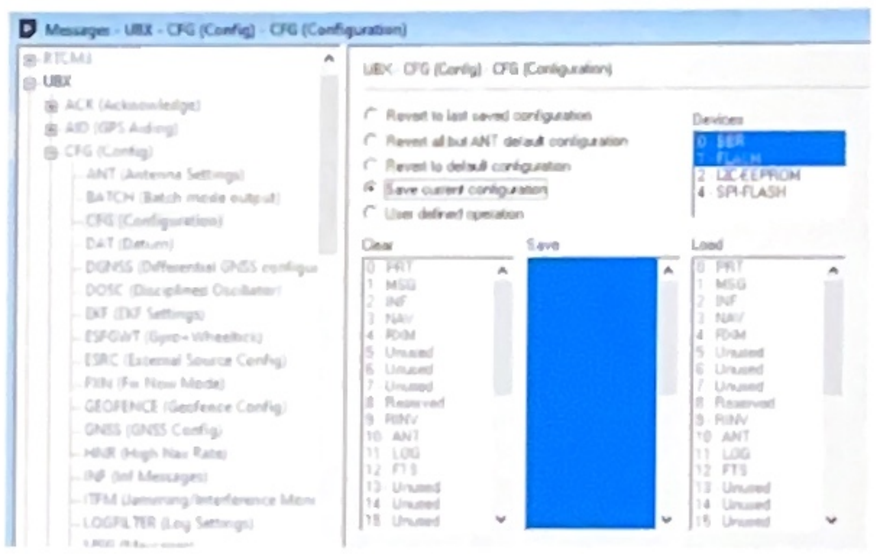
Rover Setup

We're going to assume you've got a base station setup, either as a temporary base or with ECEF coordinates (we'll go into that advanced topic later). u-blox has some good documents on how to set up the ZED-F9P in rover mode but it's surprisingly easy. The rover simply needs to be fed RTCM corrections and it will enter RTK Float, then RTK Fix.

Open u-center and connect to the rover module. Multiple instances of u-center can be started if you have multiple modules connected to one computer.

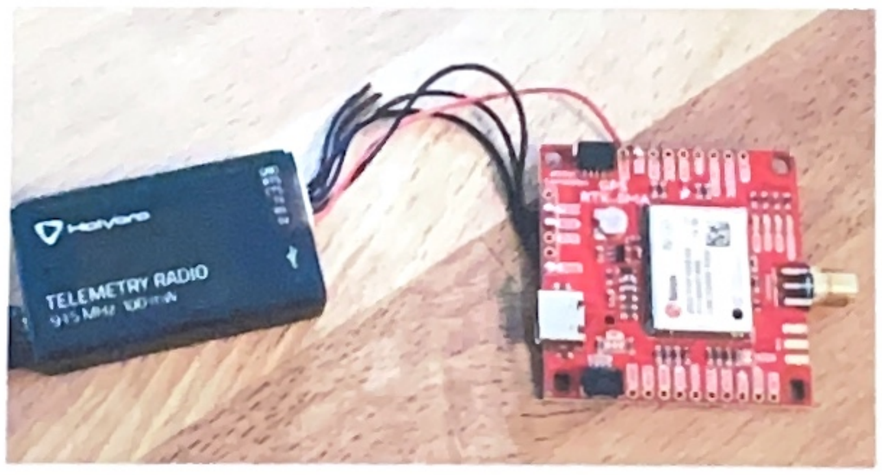


Navigate to the CFG / PRT message and configure UART2. We want to set UART2 to 57600bps to match the base and the telemetry radio kit we will be using.



Now save these port settings so they get loaded at each power-on.

Note: If you ever run into problems with the configuration of a module you can always select 'Revert to default configuration' and hit send. This should give your module a good brainwash returning it to the default factory settings. This command will also reset the BBR/Flash settings.



Solder the other half of the radio cable to the UART2 on the rover receiver. The setup is *exactly* the same as before with the base receiver:

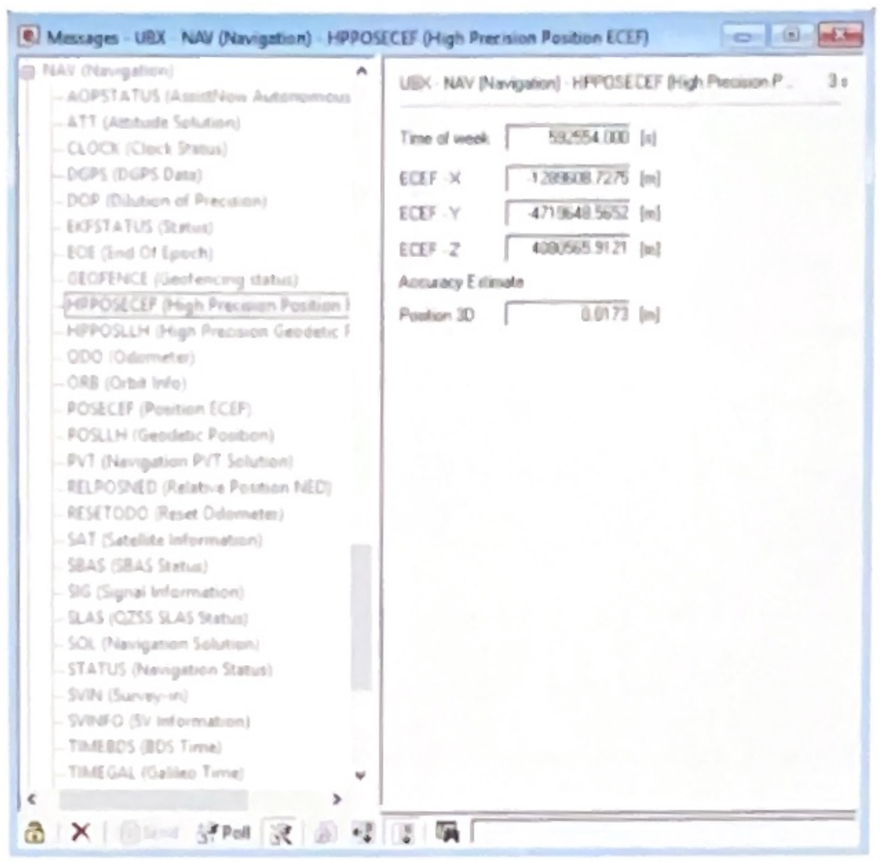
- Radio 5V - RTK 5V

* I set this to 230,400 to try to make sure LoRa would be fast enough.

- Radio TX - RTK RX2 *
- Radio RX - RTK TX2
- Radio GND - RTK GND

Time for a test! Power up the rover and base and you should see the green LED on the radios go from blinking to solid indicating the radios detect each other and are passing serial data between them. A blinking red LED on the radios indicates serial data being transmitted.

That's it! This rover module is now ready for the field. It is still recommended to have u-center handy to view the status of the rover as it receives RTCM corrections. As the rover receives RTCM it will quickly move from a 3D fix, to RTK Float, to RTK Fix.



There are a few messages to watch while the rover achieves RTK Fix. NAV/HPPOSECEF will show overall positional accuracy, NAV/HPPOSLLH will show the horizontal accuracy. NAV/PVT will show the fix type as it progresses from 3D, to RTK Float, to RTK Fixed.

PSST: We messed up. The RTK LED on the GPS-RTK boards is wired to ground instead of 3.3V. This means the RTK LED will be on in normal non-RTK mode. Once RTCM is being received and the module enters RTK Float the LED will blink. Once enough corrections are received, the RTK LED will turn off indicating RTK Fixed mode which is the best, 14mm horizontal accuracy situation.

- RTK LED On = Regular Satellite Mode
- RTK LED Blinking = RTK Float, ~<500mm accuracy
- RTK LED Off = RTK Fix, 14mm acurracy

* On our rover, we connected I2C from the Zed-F9P to I2C on the 1310. Then we connected UART2 to Serial 2.

The following code will push the RTCM messages recieved from LoRa to UART2 and track GPS locations.

Setting up an RTK Base and Rover System

06/28/2023
Wesley Cooke

```
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include <SD.h>
#include <SparkFun_u-blox_GNSS_v3.h>

// ZED-F9P Object
SFE_UBLOX_GNSS myGNSS;

long lastTime = 0;
uint8_t myBytes[249]; // One LoRa Packet
uint8_t myCount=0;
const int chipSelect = 7;

void setup()
{
  // Serial1 is connected to UART2 of the Sparkfun Board.
  Serial1.begin(230400);
  Wire.begin();

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  pinMode(chipSelect, OUTPUT);

  if (myGNSS.begin() == false)
  {
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  }

  myGNSS.setI2COutput(COM_TYPE_UBX);
  myGNSS.saveConfigSelective(VAL_CFG_SUBSEC_IOPORT);

  if (!LoRa.begin(915E6))
  {
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  }

  if (!SD.begin(chipSelect))
  {
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  }

  // If we made it this far, we are good.
  digitalWrite(LED_BUILTIN, LOW);

  File outputfile = SD.open("data.txt", FILE_WRITE);
  outputfile.println("Latitude, Longitutde, RTK, FIX, SIV");
  outputfile.close();
}
```

```
void loop()
{
  // Try to parse the packet
  int packetSize = LoRa.parsePacket();

  // If this packet is the size we expect
  if (packetSize == 250)
  {
    // Unpack it and send it out via Serial1
    for(int i=0; i<250; i++)
    {
      uint8_t myByte = LoRa.read();
      Serial1.write(myByte);
    }
  }

  if (millis() - lastTime > 1000)
  {
    lastTime = millis();
    long latitude = myGNSS.getLatitude();
    long longitude = myGNSS.getLongitude();
    byte fixType = myGNSS.getFixType();
    byte RTK = myGNSS.getCarrierSolutionType();
    byte SIV = myGNSS.getSIV();

    File outputfile = SD.open("data.txt", FILE_WRITE);
    if (outputfile)
    {
      outputfile.print(latitude);
      outputfile.print(", ");
      outputfile.print(longitude);
      outputfile.print(", ");
      outputfile.print(RTK);
      outputfile.print(", ");
      outputfile.print(fixType);
      outputfile.print(", ");
      outputfile.println(SIV);
    }
    outputfile.close();
  }
}
```

Setting up an RTK Base and Rover System

06/28/2023
Wesley Cooper

Figure 3
GPS data with RTK correction data.

* This portion of the data shows amazing results. A group of students were walking toward me. I moved to the side of the sidewalk. The gps was able to track that deviation.

07/30/2023
Wesley Cooke

A No OSS Relative Heading Observation

When testing the autonomous boat, we noticed an undesired change in direction and resultant spiral out of control.

Figure 1
GPS data of the waterlcraft in autonomous mode. Note that the path is good during the beginning phases, but begins acting more and more weird near the half way point.

BNO055 Relative Heading Observation

07/30/2023
Wesley Cocke

If we observe the GPS and the heading coming from the BNO055, we begin to see an issue.

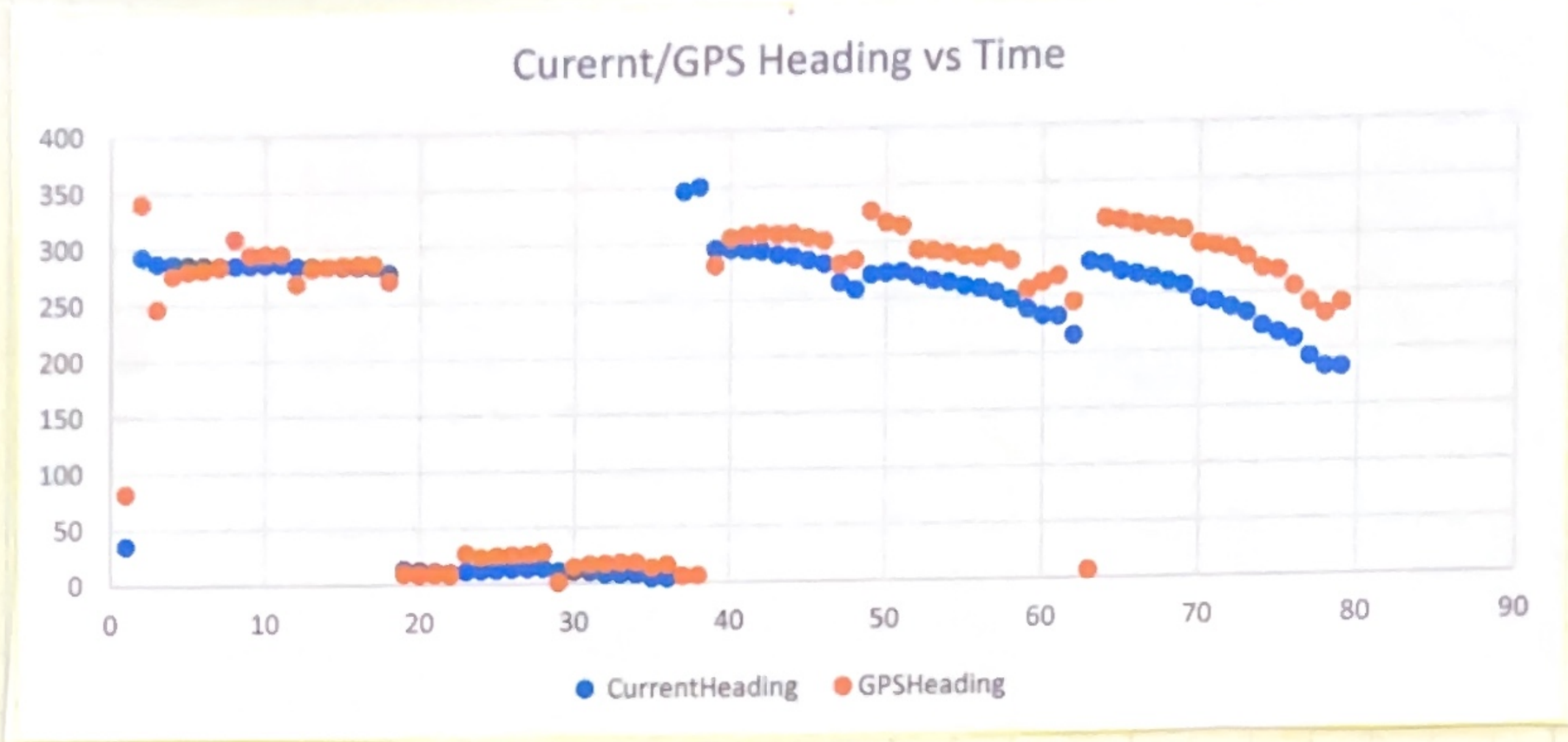


Figure 2
At the start of our program, the GPS and BNO tend to agree on what the heading is. It begins to drift apart though.

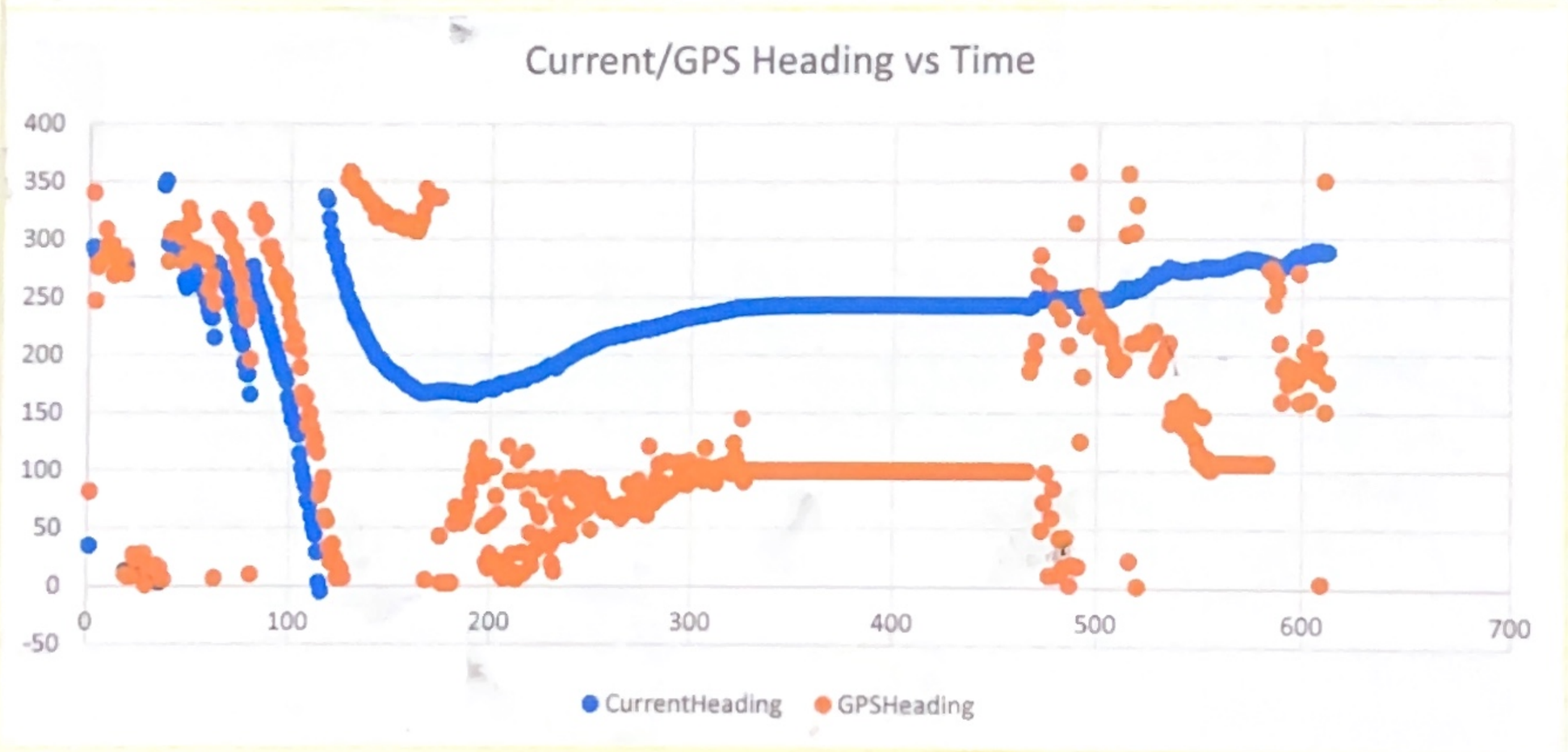


Figure 3
At sompoint the difference between the GPS and the BNO is very large.

What caused this difference?

Gyro drift may account for some of the error. However, gyro drift seems to be a small error compounded over time. Not a large jump.

Could this be gimbal Lock?...

Observations of the BNO055 by itself:

using the following program we can observe the heading reported by the BNO055 on the serial monitor.

bno_relative_test

```
#include <math.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

Adafruit_BNO055 myIMU = Adafruit_BNO055();
imu::Vector<3> euler;
imu::Vector<3> gyro;
imu::Quaternion quat;
float currentHeading;
uint8_t systemCal, gyroCal, accelCal, mgCal;

void setup()
{
  Serial.begin(115200);

  if (!myIMU.begin())
  {
    Serial.println("Failed to init BNO.");
    while (1);
  }
  //  Serial.println("BNO Connected.");
  Serial.print("Heading, GyroX, GyroY, GyroZ, gyroCal, accelCal, systemCal");

  delay(1000);
  myIMU.setExtCrystalUse(true);
  myIMU.setMode(adafruit_bno055_opmode_t::OPERATION_MODE_IMUPLUS);

}
```

BNO055 Relative Heading Observation

07/30/2023
Wayne

```
void loop()
{
  gyro = myIMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
  quat = myIMU.getQuat();
  euler = quat.toEuler();
  myIMU.getCalibration(&systemCal, &gyroCal, &accelCal, &mgCal);

  Serial.print(euler.x() * (180/3.14), 2);
  Serial.print(", ");
  Serial.print(gyro.x(), 2);
  Serial.print(", ");
  Serial.print(gyro.y(), 2);
  Serial.print(", ");
  Serial.print(gyro.z(), 2);
  Serial.print(", ");
  Serial.print(gyroCal);
  Serial.print(", ");
  Serial.print(accelCal);
  Serial.print(", ");
  Serial.println(systemCal);

  delay(100);
}
```

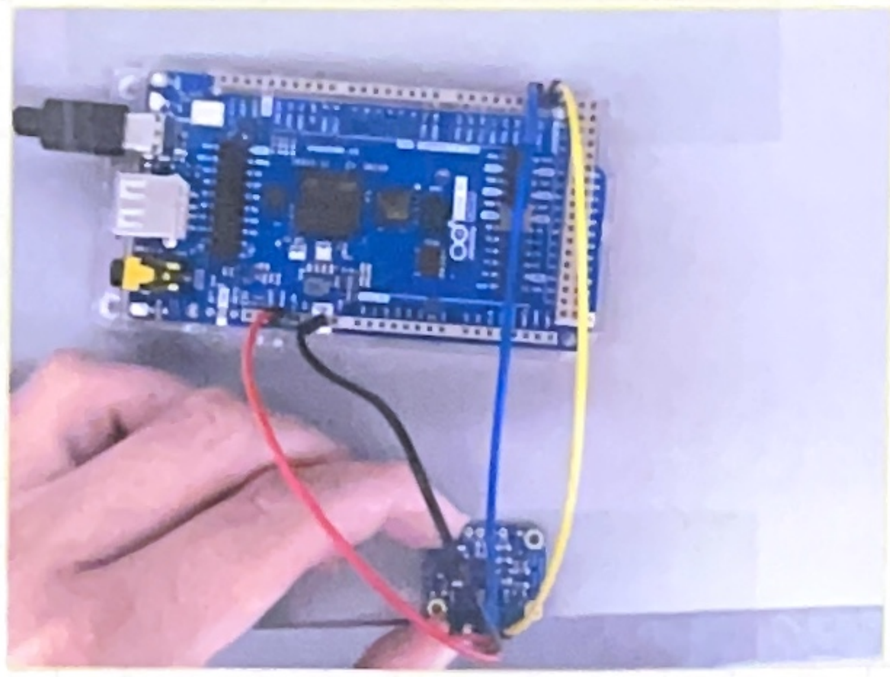


Figure 4
Orientation 1, When you power on your device, make sure the BNO055 is in this orientation. This ensures that 0° is aligned with the table.

Procedure

Rotate the chip about the Z axis. Bring it back to 0.
Rotate about the Y axis. Bring it back to 0.
Rotate about the X axis. Bring it back to 0.

Observe the output.

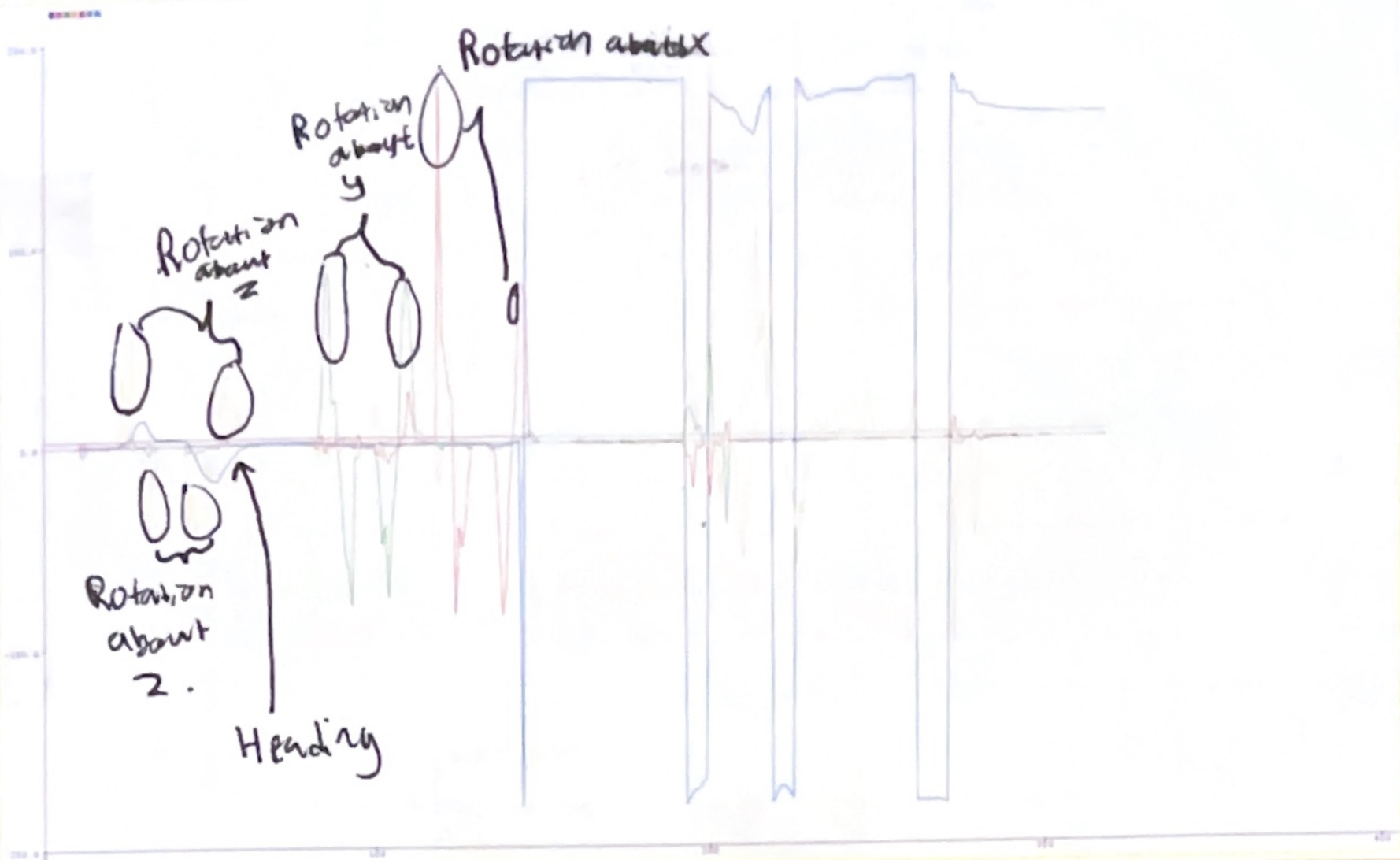


Figure 5

Starting in orientation 1 from figure 4 and following the procedure. This is the collected data.

Notice the blue line that represents heading starts around 0.

The yellow lines represent rotation about the Z-axis.
The green lines represent rotation about the y-axis.
The Red lines represent rotation about the x-axis.

Notice that at some ~~point~~ point after/during the rotation about the x-axis, our heading jumps from 0 to -180/180. This is even when we have returned it to Orientation 1.

BNO055 Relative Heading Observation

07/30/2023
evening coding

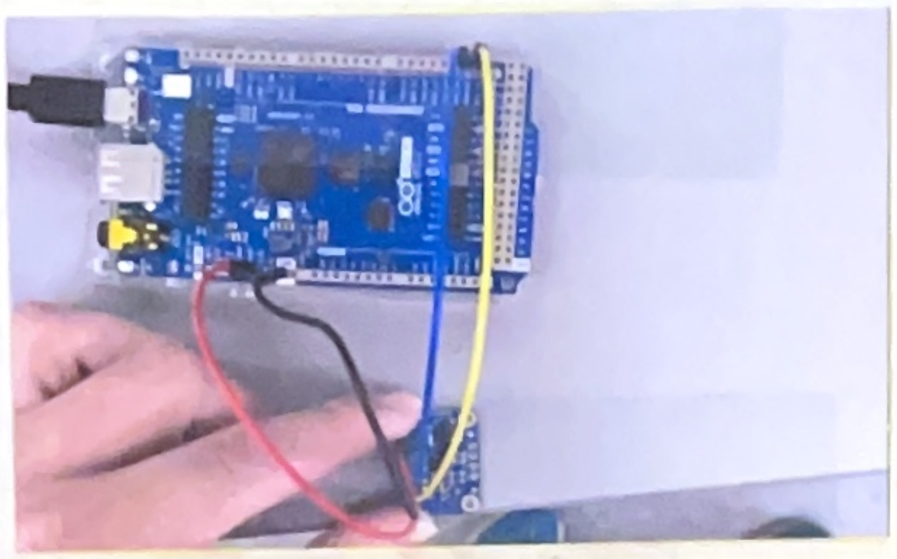


Figure 6
orientation 2.

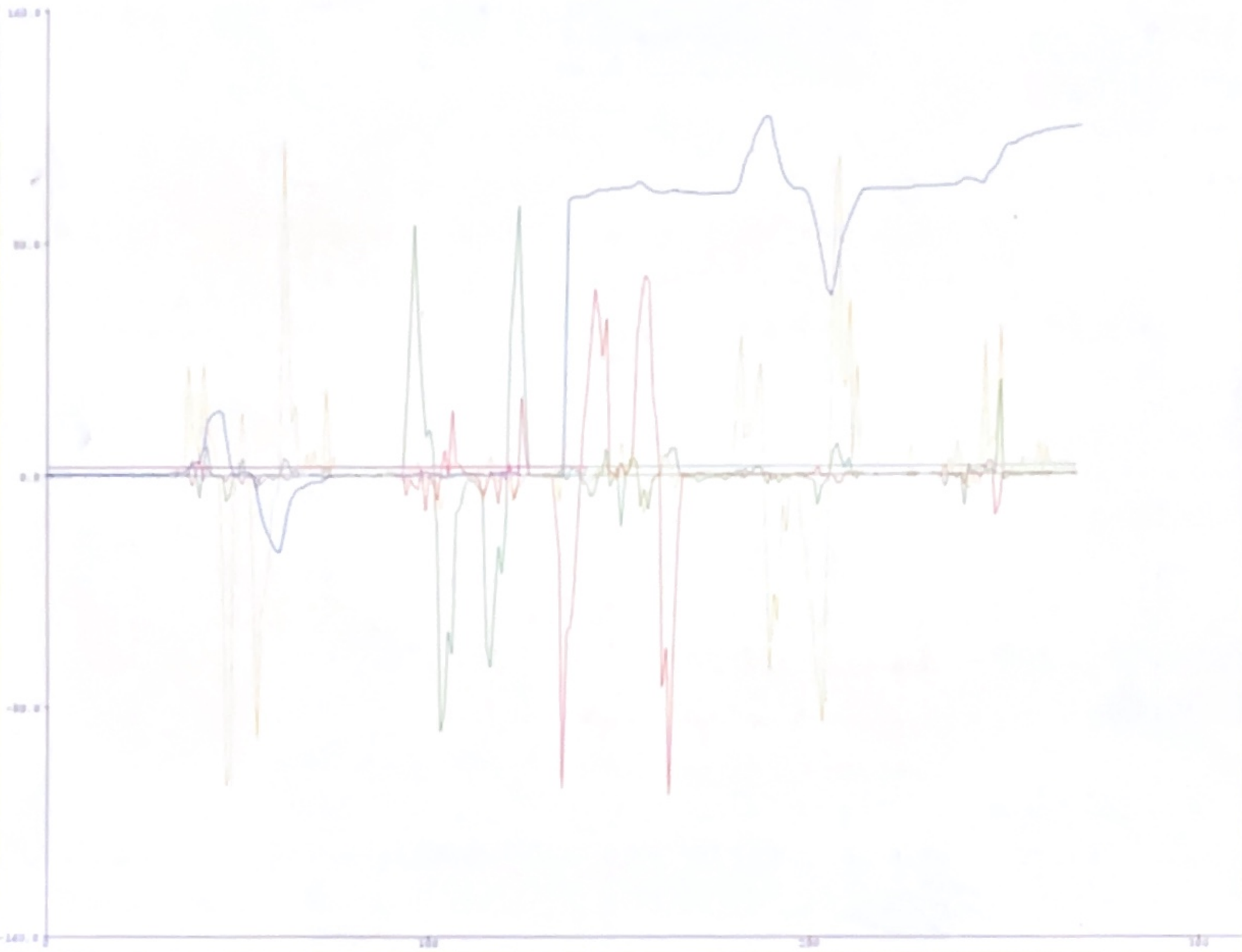


Figure 7
Data from orientation 2 in Figure 6.

Notice this time the 0° point jumps to 90°.

You can try this in all orientations.

07/30/2023
Wesley Cook

Blue Robotics T200 Underwater thruster

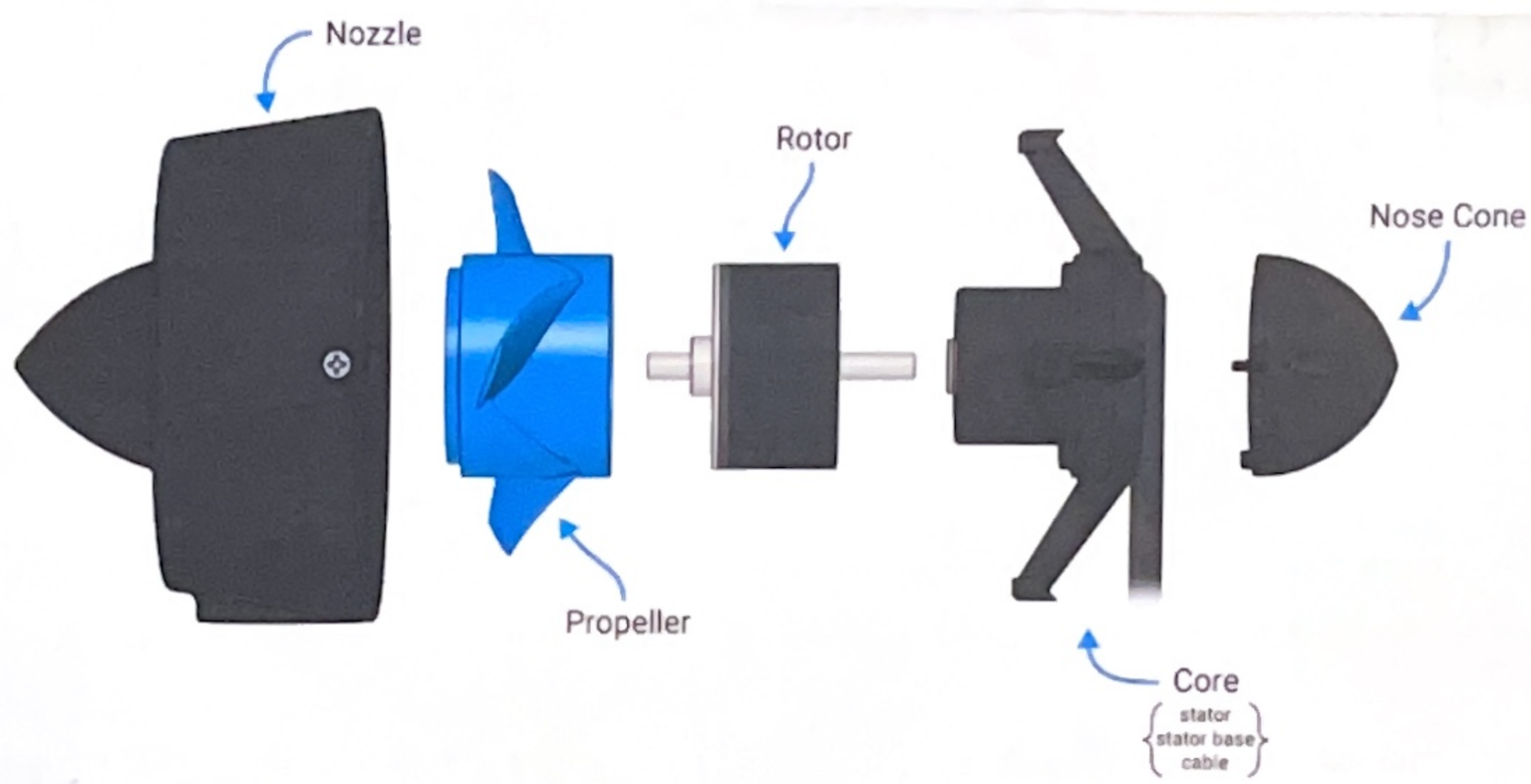
The T200 is a 3phase brushless dc motor. It operates in a similar manner to a stepper motor, but is constructed very differently. Based on the way current is applied to the three phases, the motor will spin. However, it ~~is much f~~ can achieve much greater speeds.

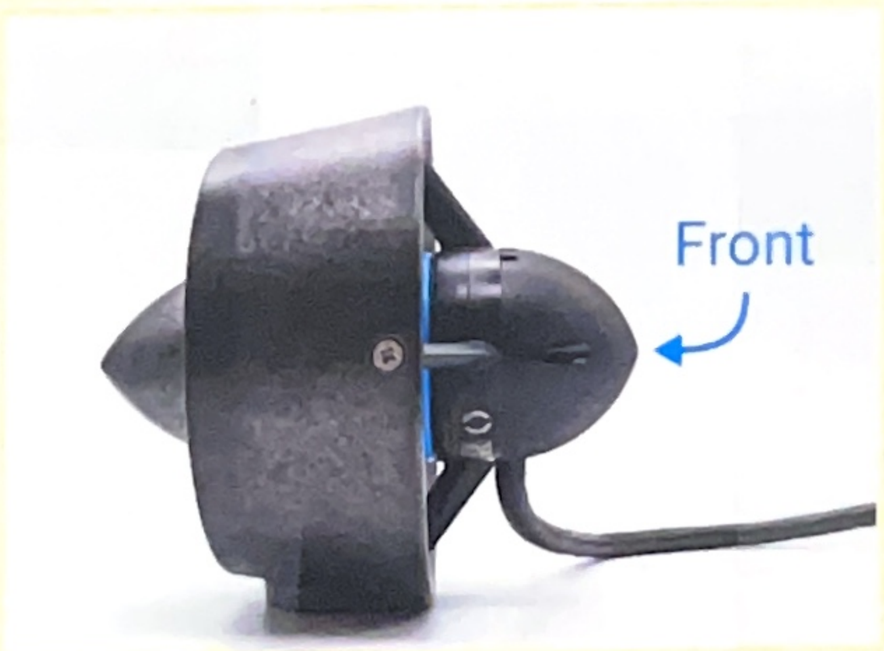
These motors require an electronic speed controller.* The speed controller will handle sending current to the correct phases at the appropriate times. The ESC requires a pulse of 20 ms with an on period of between 1000ms and 2000ms. *(esc)

Parts of a Thruster

This diagram shows the main components of a T200 or T500 thruster.

https://bluerobotics.com/learn/thruster-usage-guide/





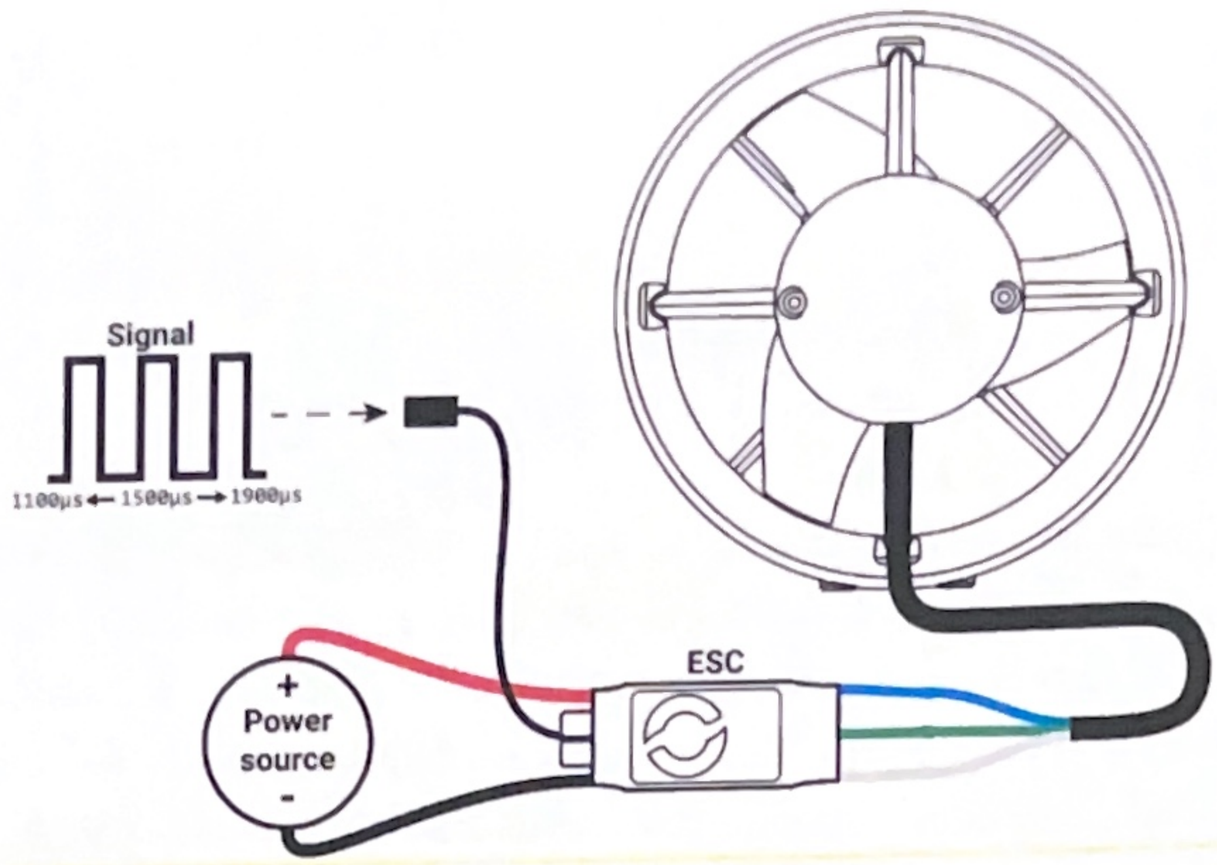
Figures 1 and 2

Basic components of the T200 thruster.

Blue Robotics T200 underwater Thruster

07/30/2013
Wesley Cooper

The following diagram shows how everything is connected:

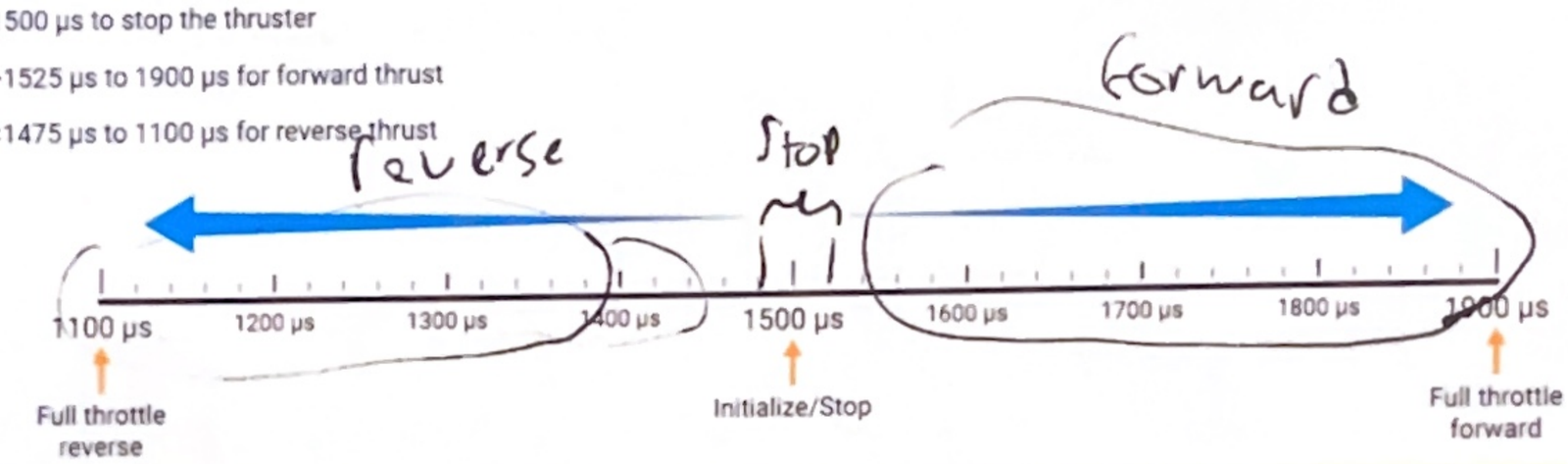


3. Sending the Signal

Connect the white ESC signal wire to the signal source output. Connect the black (ground) ESC wire to a ground pin on the signal source device. Send a PWM signal of 1500 microseconds (µs) to initialize the ESC. The ESC must be initialized before it can accept any other throttle signals. When the ESC is initialized you should hear two more beeps. The first beep indicates the ESC detects a throttle signal and the second beep indicates that the correct 1500 µs signal is detected and the ESC is now fully initialized.

After the ESC is initialized, sending a PWM signal from 1100 µs to 1900 µs will control the throttle of the thruster:

- 1500 µs to stop the thruster
- >1525 µs to 1900 µs for forward thrust
- <1475 µs to 1100 µs for reverse thrust



We used a 14.8V Li-Po battery as our power source.

To provide the signal to the ESC, we treat it as a servo object in code.

07/30/2023
Wesley Cook

Blue Robotics T200 Underwater Thruster


```
#include <Servo.h>

byte servoPin = 9;
Servo servo;

void setup() {

 Serial.begin(9600);
 servo.attach(servoPin);

 servo.writeMicroseconds(1500); // send "stop" signal to ESC.

 delay(7000); // delay to allow the ESC to recognize the stopped signal
}

void loop() {

  Serial.println("Enter PWM signal value 1100 to 1900, 1500 to stop");

  while (Serial.available() == 0);

  int val = Serial.parseInt();

  if(val < 1100 || val > 1900)
  {
    Serial.println("not valid");
  }
  else
  {
    servo.writeMicroseconds(val); // Send signal to ESC.
  }
}
```

07/30/2023
Wesley Cadra

Autonomous WIL

Objective

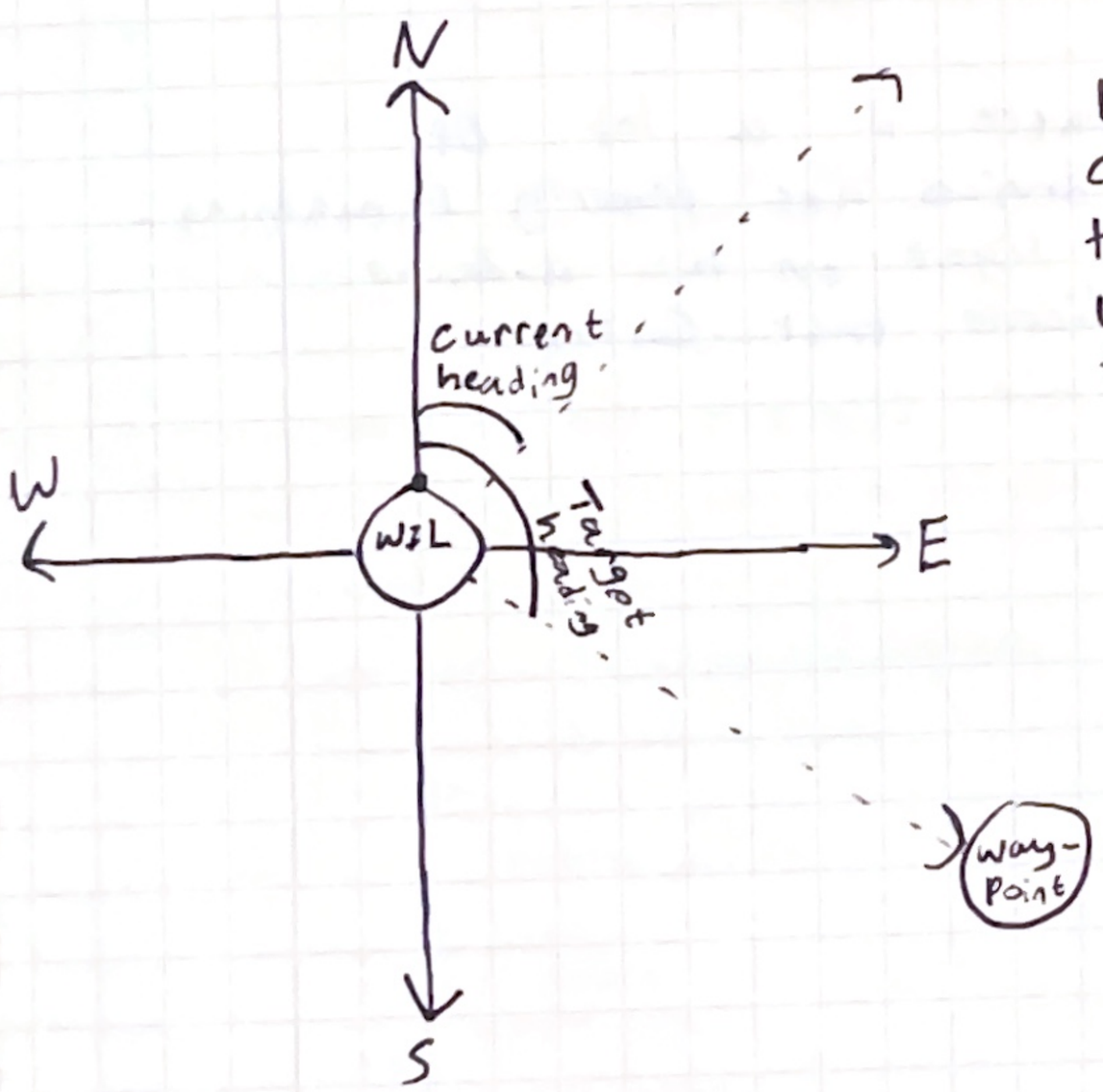
Combine the knowledge of GPS, BNO085, T200 thrusters, and LoRA to develop an autonomous boat affectionately call "WIL". Water is Life.

The gps will provide current location information. The BNO will provide heading information. Using google maps, we can select GPS destinations or "waypoints".

Using the current location and the waypoint information, we can calculate what heading WIL needs to be to at. to travel to the destination.

The BNO will ~~tells~~ tell us if we get too far off from the target heading.

We have added a MKR 1310 to allow for LoRa communications with a controller.



When we detect that the current heading is off from the target by some threshold, we control the motors in a way that rotates WIL towards the waypoint.

Code

```
#ifndef MainObjects_h_
# define MainObjects_h_

//////////////////
// GNSS Object //
#include <SparkFun_u-blox_GNSS_v3.h>
SFE_UBLOX_GNSS_SERIAL myGNSS;

////////////////
// BNO Object //
#include <Adafruit_BNO08x_RVC.h>
Adafruit_BNO08x_RVC myIMU = Adafruit_BNO08x_RVC();
const int bnoRST = 13;

////////////
// Servos // for the motor esc's
#include <Servo.h>
const int leftServoPin = 2;
const int rightServoPin = 7;
Servo leftServo;
Servo rightServo;

/////////////////
// Program Vars //
bool programState = 0; → 0 = Autonomous , 1 = Remote control.
int leftMotorSpeed = 1500;
int rightMotorSpeed = 1500; } 1500 = stopped
int leftMotorSpeedTarget = 1500;
int rightMotorSpeedTarget = 1500; } updated from remote control.

#endif
```

These variables and objects are used throughout the program.

On the following page, there is a lot of code that ensures different devices are properly functioning. If some component fails, a light on the arduino giga will light up to indicate that failure.

07/30/2023
Wesley Cooke

Autonomous WIL

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```
#include "MainObjects.h"
#include "AutoMode.h"
#include "RemoteMode.h"

#include <SPI.h>
#include <SD.h>
#include <Adafruit_BNO08x_RVC.h>
#include <SparkFun_u-blox_GNSS_v3.h>
#include <Servo.h>
```

} Files I made

} Libraries we are using

```
void setupSD()
{
    int chipSelect = 10;
    pinMode(chipSelect, OUTPUT);
    if(!SD.begin(chipSelect))
    {
        Serial.println("SD card failed to init... Freezing.");

        digitalWrite(LED_BUILTIN, HIGH);
        while(1);
    }
    Serial.println("Card Initialized!");

    File outputfile = SD.open("data.txt", FILE_WRITE);
    outputfile.println("Latitude, Longitude, Name, DistanceBetween, CurrentHeading, Targ
    outputfile.close();
}
```

// This method ensures that our SD card is properly functioning.

```
void setupGNSS()
{
    Serial1.begin(38400); // RX0 TX0

    if (!myGNSS.begin(Serial1))
    {
        Serial.println("GNSS failed to init... Freezing.");

        digitalWrite(LED_BUILTIN, HIGH);
        while(1);
    }

    myGNSS.setUART1Output(COM_TYPE_UBX);
    myGNSS.setI2COutput(COM_TYPE_UBX);
    myGNSS.saveConfiguration();

    Serial.println("GNSS initialized!");
}
```

// This method ensures that our GPS is properly functioning.

```
void setupBNO()
{
    Serial2.begin(115200); // Baud set by datasheet

    if (!myIMU.begin(&Serial2))
    {
        Serial.println("Could not find BNO08x!");

        digitalWrite(LED_BUILTIN, HIGH);
        while(1);
    }
    Serial.println("BNO08x found!");

    // Reset the heading
    pinMode(bnoRST, OUTPUT);
    digitalWrite(bnoRST, LOW);
    delay(10);
    digitalWrite(bnoRST, HIGH);

    Serial.println("BNO heading zeroed...");
}
```

// This method ensures our inertial measurement unit is properly functioning.

Autonomous WIL

07/30/2025
Vesely
Cadec

```
void setupServos()
{
    leftServo.attach(leftServoPin);
    rightServo.attach(rightServoPin);

    leftServo.writeMicroseconds(1500);
    rightServo.writeMicroseconds(1500);

    // Let the ESC's catch up
    // Listen for a high pitched tone after the first sequence of notes.
    delay(10000);
}
void setup()
{
    Serial.begin(115200);
    Serial.println("Initializing sensors!");

    pinMode(LED_BUILTIN, OUTPUT);

    // setupUSB();
    setupSD();

    setupGNSS();

    setupBNO();

    setupServos(); // Delays for 10 seconds for the ESC's

    Serial3.begin(115200); // For the LoRa UART.
}
void loop()
{
    readController(); // Make sure we read the controller

    if (programState == 0) // If the controller is in autonomous mode
    {
        autoMode(); // do the autonomous mode.
    }
    else if (programState == 1) // If the controller is in remote control mode
    {
        remoteMode();
    }
}
```

// do the remote mode.

Here, our code is fairly simple:
~~on the~~ On start, run all the setup methods.

In our loop, we are ~~ret~~ checking to see if the controller has sent some information. If it has changed which program mode we are in, the appropriate function will be called.

07/30/2023
Wrote code

Autonomous WIL

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h

```
#ifndef AutoMode_h_
# define AutoMode_h_

#include "GetHeading.h"
#include "TargetHeading.h"
#include "DistanceBetween.h"
#include "UpdateGPS.h"
#include "HeadingError.h"
#include "RotateInPlace.h"
#include "LogToSD.h"
#include "UpdateMotors.h"
#include "LogToSD.h"
// Index that we access the destination arrays by.
int waypointIndex = 0;

// Around the block but more waypoints
uint8_t numWaypoints = 27;

double currentHeading;

// Current Latitude and Longitude
long cLat;
long cLon;
long dLat; // destination Lat + Lon
long dLon;

long gpsHeading;
long gpsGroundSpeed;

double distanceBetween; // distance between current position and destination
double targetHeading;

double headingError;
double acceptableHeadingError = 0.02;

byte SIV;
// Destination Longitudes and Lattitudes
long dLons [] = { -820051916, ...}; // Cut for brevity
long dLats [] = { 334924613, ...};
void printVars()
{
  Serial.print("SIVs: ");
  Serial.println(SIV);

  Serial.print("Current Lat: ");
  Serial.println(cLat);

  Serial.print("Current Lon: ");
  Serial.println(cLon);

  Serial.print("Target Heading: ");
  Serial.println(targetHeading, 6);

  Serial.print("Current Heading: ");
  Serial.println(currentHeading);

  Serial.print("Heading Error: ");
  Serial.println(headingError, 6);

  Serial.print("Distance Between: ");
  Serial.println(distanceBetween, 6);
}
```

These are files that we are using to autonomously navigate.

Autonomous WIL

07/30/2023
Wesley Cooke

```
void autoMode()
{
  // Get the destination we want to go to based on the index.
  dLat = dLats[waypointIndex];
  dLon = dLons[waypointIndex];

  // Update our GPS variables and their distance
  updateGPS(&cLat, &cLon, &SIV, &gpsHeading, &gpsGroundSpeed);
  distanceBetween = getDistanceBetween(cLat, cLon, dLat, dLon);

  // Get the currentHeading, targetheading, and the error between them.
  currentHeading = getHeading();
  targetHeading = getTargetHeading(cLat, cLon, dLat, dLon);
  headingError = getHeadingError(currentHeading, targetHeading);

  // If at least 4 sats are in view
  if (SIV >= 4)
  {
    // Heading is too far off
    if (abs(headingError) > acceptableHeadingError)
    {
      rotateInPlace(&targetHeading, &currentHeading, &headingError, &acceptableHeadingEr
    }

    // We have entered remote control mode...
    if (programState == 1) *
    {
      return;
    }
    // We are close enough to our target waypoint
    if (distanceBetween <= 3.0) → 3.0 m radius
    {

      // Turn motors off and go to the next waypoint
      updateMotors(1500, 1500);
      waypointIndex++;

      // Next waypoint is out of bounds
      if (waypointIndex >= numWaypoints)
      {
        // Reset
        programState = 1;
        waypointIndex = 0;
        return;
      }

    }

    // Our heading is not too far off
    // and we are not close enough to the point
    else
    {
      // Go Forwards
      updateMotors(1700, 1700);
    }
  }
  // 4 Sats are not in view. Sit still...
  else
  {
    updateMotors(1500, 1500);
    Serial.println("No satellites yet...");
  }

  // Log data to the SD card for analysis
  logToSd(cLat, cLon, distanceBetween, currentHeading, targetHeading, gpsHeading, gpsGro
  // Log to serial monitor for debugging
  printVars();
}

#endif
```

This is the main autonomous navigation algorithm.

* Inside the rotate in place method, we are checking the controller for updates.

I have labeled some helper functions on the previous page ①-⑧. Here is the code for them.

①

This method gets important information from the GPS.

```
#ifndef UpdateGPS_h_
# define UpdateGPS_h_

#include "MainObjects.h"

void updateGPS(long* aLat, long* aLon, byte* aSIV, long* aHeading, long* aGroundSpeed)
{
    *aLat = myGNSS.getLatitude();
    *aLon = myGNSS.getLongitude();
    *aSIV = myGNSS.getSIV();
    *aHeading = myGNSS.getHeading();
    *aGroundSpeed = myGNSS.getGroundSpeed();
}

#endif
```

②

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

This method computes the distance between a pair of Lat & Lons.

```
#ifndef DistanceBetween_h_
# define DistanceBetween_h_

double getDistanceBetween(long lat1_l, long long1_l, long lat2_l, long long2_l)
{
  // returns distance in meters between two positions, both specified
  // as signed decimal-degrees latitude and longitude. Uses great-circle
  // distance computation for hypothetical sphere of radius 6372795 meters.
  // Because Earth is no exact sphere, rounding errors may be up to 0.5%.
  // Courtesy of Maarten Lamers

  double lat1 = (double)lat1_l / 10000000.0; // Convert lat and long to degrees
  double long1 = (double)long1_l / 10000000.0;
  double lat2 = (double)lat2_l / 10000000.0;
  double long2 = (double)long2_l / 10000000.0;
  double delta = radians(long1-long2);
  double sdlong = sin(delta);
  double cdlong = cos(delta);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  double slat1 = sin(lat1);
  double clat1 = cos(lat1);
  double slat2 = sin(lat2);
  double clat2 = cos(lat2);
  delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta = sq(delta);
  delta += sq(clat2 * sdlong);
  delta = sqrt(delta);
  double denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta = atan2(delta, denom);
  return delta * 6372795;
}

#endif
```

Autonomous WIL

07/30/2023
Wesley Crosby

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

③

```
#ifndef GetHeading_h_
# define GetHeading_h_

#include "MainObjects.h"

BNO08x_RVC_Data bnoData;
const float declination = 7;

float getHeading()
{
    // Clear the buffer before reading
    while (Serial2.available() > 0)
    {
        Serial2.read();
    }

    // Continue reading until we get a new heading.
    while(!myIMU.read(&bnoData)) {}

    float heading = bnoData.yaw - declination;

    if (heading < 0)
    {
        heading += 360;
    }

    return heading;
}

#endif
```

④

```
#ifndef TargetHeading_h_
# define TargetHeading_h_

double getTargetHeading(long lat1_l, long long1_l, long lat2_l, long long2_l)
{
  // returns course in degrees (North=0, West=270) from position 1 to position 2,
  // both specified as signed decimal-degrees latitude and longitude.
  // Because Earth is no exact sphere, calculated course may be off by a tiny fraction.
  // Courtesy of Maarten Lamers

  double lat1 = (double)lat1_l / 10000000.0; // Convert lat and long to degrees
  double long1 = (double)long1_l / 10000000.0;
  double lat2 = (double)lat2_l / 10000000.0;
  double long2 = (double)long2_l / 10000000.0;
  double dlon = radians(long2-long1);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  double a1 = sin(dlon) * cos(lat2);
  double a2 = sin(lat1) * cos(lat2) * cos(dlon);
  a2 = cos(lat1) * sin(lat2) - a2;
  a2 = atan2(a1, a2);
  if (a2 < 0.0)
  {
    a2 += TWO_PI;
  }
  return degrees(a2);
}

#endif
```

07/30/2023 Wesley Cecil

Autonomous WIL

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

5

```
#ifndef HeadingError_h_
# define HeadingError_h_

double getHeadingError(double aHeading, double aTargetHeading)
{
    // What is the Difference in our heading
    double error = aTargetHeading - aHeading;

    // Rescale to give shortest turn distance.
    // Will be between -180 and 180.
    if (error >= 180)
    {
        error -= 360;
    }
    else if (error <= -180)
    {
        error += 360;
    }
    return error/180.0;
}

#endif
```

6

```
#ifndef RotateInPlace_h_
# define RotateInPlace_h_

#include "HeadingError.h"
#include "GetHeading.h"
#include "MainObjects.h"
#include "ReadController.h"
#include "UpdateMotors.h"
void rotateInPlace(double* theTargetHeading, double* theHeading, double* theError, double* acceptableError)
{
    while(abs(*theError) > *acceptableError)
    {
        Serial.println("Fixing the heading...");
        // Pivot right
        if (*theError > 0)
        {
            updateMotors(1700, 1280);
        }
        // Pivot Left
        else
        {
            updateMotors(1280, 1700);
        }

        *theHeading = getHeading();

        *theError = getHeadingError(*theHeading, *theTargetHeading);

        Serial.print("Fixing Error: ");
        Serial.println(*theError);

        readController();

        if (programState == 1)
        {
            return;
        }
    }
}

#endif
```

Autonomous WIL

07/30/2023, Wesley Cuba

7

```
#ifndef UpdateMotors_h_
# define UpdateMotors_h_

#include "MainObjects.h"

void updateMotors(int targetLeft, int targetRight)
{
    while (leftMotorSpeed != targetLeft or rightMotorSpeed != targetRight)
    {
        if (leftMotorSpeed != targetLeft)
        {
            if(leftMotorSpeed < targetLeft)
            {
                leftMotorSpeed++;
            }
            else
            {
                leftMotorSpeed--;
            }
        }

        if (rightMotorSpeed != targetRight)
        {
            if(rightMotorSpeed < targetRight)
            {
                rightMotorSpeed++;
            }
            else
            {
                rightMotorSpeed--;
            }
        }

        leftServo.writeMicroseconds(leftMotorSpeed);
        rightServo.writeMicroseconds(rightMotorSpeed);
    }
}

#endif
```

only change the motor speed by 1 at a time. This helps eliminate abrupt jutters.

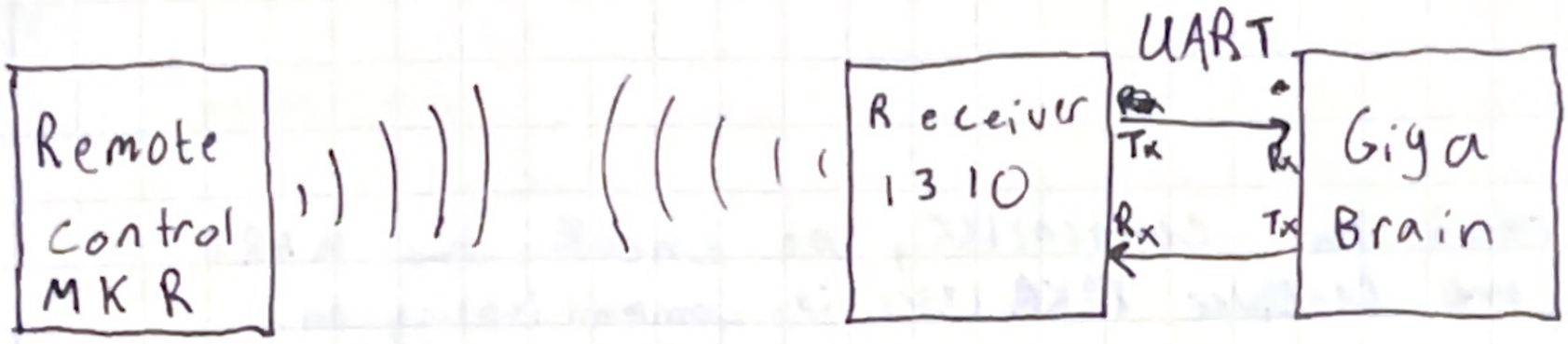
8

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```
#ifndef LogToSD_h_
# define LogToSD_h_

#include <SD.h>

int sampleNum = 1;

void logToSd(long lat, long lon, double distance, double cHead, double tHead, long gpsHead, long groundSpeed, byte SIVs)
{
    File outputfile = SD.open("data.txt", FILE_WRITE);

    if (outputfile)
    {
        outputfile.print(lat);          // Latitude
        outputfile.print(", ");
        outputfile.print(lon);          // Longitude
        outputfile.print(", ");
        outputfile.print(sampleNum);          // Name
        outputfile.print(", ");
        outputfile.print(distance);          // Distance
        outputfile.print(", ");
        outputfile.print(cHead);
        outputfile.print(", ");
        outputfile.print(tHead);
        outputfile.print(", ");
        outputfile.print(gpsHead);       // SIV
        outputfile.print(", ");
        outputfile.print(groundSpeed); // GroundSpeed from GPS
        outputfile.print(", ");
        outputfile.print(SIVs);     // Heading from GPS
    }

    outputfile.close();
    sampleNum++;
}

#endif
```

Next, we should explain how the remote control works. To achieve this, we are using point to point LoRa. However, the giga board does not have LoRa capabilities. To fix this, we are using a 1310 MKR board to recieve signals and then using a uart to send that data to the giga.

Remote control Block Diagram



Code on the Giga for Remote control

```
#ifndef RemoteMode_h_
# define RemoteMode_h_

#include "ReadController.h"
#include "UpdateMotors.h"
#include "MainObjects.h"

void remoteMode()
{
    readController();

    Serial.print(leftMotorSpeed);
    Serial.print(", ");
    Serial.print(rightMotorSpeed);
    Serial.print(", ");
    Serial.print(leftMotorSpeedTarget);
    Serial.print(", ");
    Serial.println(rightMotorSpeedTarget);

    updateMotors(leftMotorSpeedTarget, rightMotorSpeedTarget);
}

#endif
```

This is method is ran when the controller reports the Remote control mode. This mode will continue to check the controller for moter speeds and update them accordingly.

Autonomous WIL

07/30/2023
W Sylvy
Crandall

(9)

```
void readController()
{
  while (Serial3.available())
  {
      char start = Serial3.read();

      if (start == 'S')
      {
          _read_speed();
      }
      else if (start == 'M')
      {
          _read_state();
      }
  }
}

#endif
```

To read the controller, we check the UART that the receiver MKR 1310 is communicating on.

Here, we are looking for two types of LoRa packets. 'S' stands for "Speed" and 'M' stands for "Mode".

```
#ifndef ReadController_h_
# define ReadController_h_

#include "MainObjects.h"
#include "UpdateMotors.h"

char leftSpeedChars[5];
char rightSpeedChars[5];

void _read_state()
{
    char data = Serial3.read();

    if (Serial3.read() == 'E')
    {
        if (data == '0')
        {
            programState = 0;

            Serial.println("Switched to autonomous mode.");
        }
        else if (data == '1')
        {
            programState = 1;

            Serial.println("Switched to remote control mode.");
            updateMotors(1500, 1500);
        }
    }
}
```

If we see an 'M' we are looking for a 0 or 1.

```
void _read_speed()
{
    uint8_t numPackets = Serial3.available();

    while(numPackets < 8)
    {
        numPackets = Serial3.available();
        delay(10);
    }

    for (int i = 0; i < 4; i++)
    {
        leftSpeedChars[i] = Serial3.read();
    }
    leftSpeedChars[4] = '\0';

    for (int i = 0; i < 4; i++)
    {
        rightSpeedChars[i] = Serial3.read();
    }
    rightSpeedChars[4] = '\0';

    if (Serial3.read() == 'E')
    {
        leftMotorSpeedTarget = atoi(leftSpeedChars);
        rightMotorSpeedTarget = atoi(rightSpeedChars);
    }
}
```

If we see an 's', we know we are looking
4 bytes for the left speed
4 bytes for the right speed
an 'E' for the end of the packet.

We are storing each byte into an array and then using the "atoi" method to transform ASCII to an integer value.

Autonomous WIL

07/30/2023
2 weeks
code

Code on the Receiver MKR 1310.

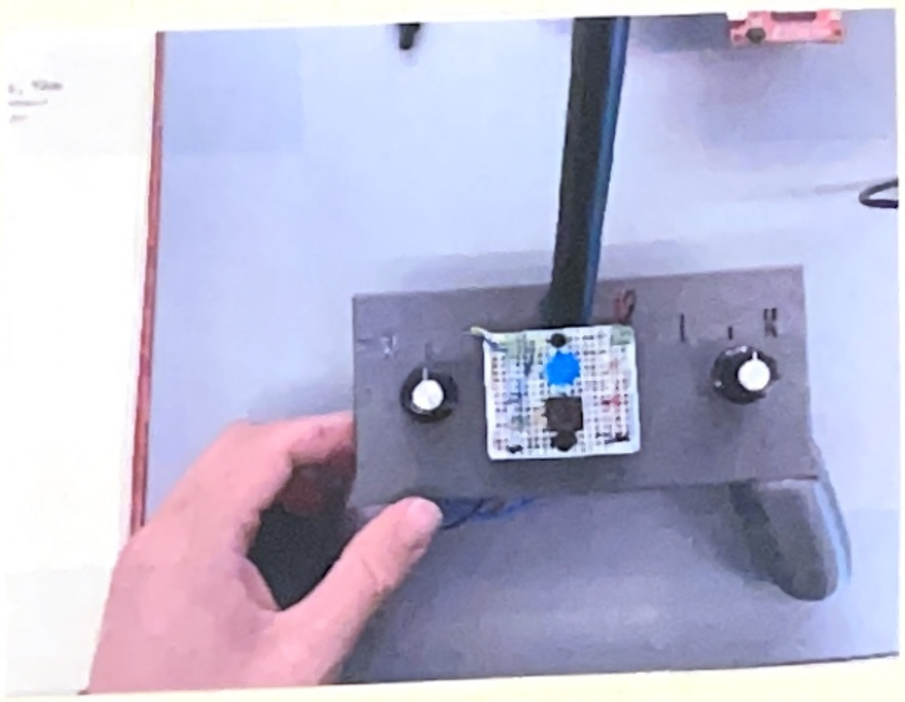
controller_receiver

```
#include <SPI.h>
#include <LoRa.h>

int leftSpeed;
int rightSpeed;
bool ledState;
char leftSpeedChar[5];
char rightSpeedChar[5];
bool validData;

void setup()
{
//  Serial.begin(9600);
  Serial1.begin(115200);// Hard ware Serial to Giga.

  if (!LoRa.begin(915E6))
  {
//    Serial.println("Starting LoRa failed!");
    digitalWrite(LED_BUILTIN, HIGH);
    while (1);
  }
  digitalWrite(LED_BUILTIN, LOW);
}
void loop()
{
  // Grab a packet
  int packetSize = LoRa.parsePacket();

  // Assuming there is a packet
  if (packetSize)
  {
    validData = 0;//Assume the data is invalid.

    // Read the first bit.
    char start = LoRa.read();

    if (start == 'S') // 'S' means we are receiving speed data
    {

      // Parse each char
      for (int i = 0; i < 4; i++)
      {
        leftSpeedChar[i] = LoRa.read();
      }
      leftSpeedChar[4] = '\0';
      for (int i = 0; i < 4; i++)
      {
        rightSpeedChar[i] = LoRa.read();
      }
      rightSpeedChar[4] = '\0';

      // Convert ASCII to INT
      leftSpeed = atoi(leftSpeedChar);
      rightSpeed = atoi(rightSpeedChar);
```

```
      // Assuming we get the proper ending bit
      if (LoRa.read() == 'E')
      {

        // Check to see if the data is valid
        validData = 1;
        if (leftSpeed > 1900)
          validData = 0;
        if (leftSpeed < 1100)
          validData = 0;

        if (rightSpeed > 1900)
          validData = 0;
        if (rightSpeed < 1100)
          validData = 0;
      }

      if (validData)
      {
        // Send it out over the hardware serial
        Serial1.print('S');
        Serial1.print(leftSpeed);
        Serial1.print(rightSpeed);
        Serial1.print('E');
      }
    }

    if (start == 'M') // 'M' means we are getting a program mode
    {
      char data = LoRa.read(); // Read the mode
      if (LoRa.read() == 'E')  // Check the ending bit
      {
        Serial1.print('M');     // Send out over hardware serial
        Serial1.write(data);
        Serial1.print('E');
      }
    }

  }

}
```

This code will receive data from the remote control and check if the values make sense.

If not, we will discard the packet.



Autonomous WIL

07/30/2023
Wesley Cooke

Code on the Remote Control.

lora_controller

```
#include <SPI.h>
#include <LoRa.h>

// leftSpeed and rightSpeed will be transmitted
int leftSpeed;
int rightSpeed;
int leftSpeedPre = 0;
int rightSpeedPre = 0;

// Readings from the two pots
float leftPot;
float rightPot;

int motorValue;
int turnBias;

// flag to update program from interrupt
bool updateProgram;

bool ledState = 1;
bool programState = 1;

// Timing Variables
long lastStateSent;
long currentTime;
```

```
void setup()
{
  Serial.begin(9600);

  analogReadResolution(10); // 10 bits on analogRead.. max of 1023
  pinMode(4,  INPUT); // Push button
  pinMode(A1, INPUT);
  pinMode(A2, INPUT); // Potientiometers
  pinMode(6, OUTPUT); // LCD

  // Interrupt for the button
  attachInterrupt(digitalPinToInterrupt(4), changeProgramMode, RISING);

  // LED.. start in "remote control mode"
  digitalWrite(6, HIGH);

  // Start LoRa comms...
  if (!LoRa.begin(915E6))
  {
    Serial.println("Starting LoRa failed!");
    digitalWrite(LED_BUILTIN, HIGH);
    while (1);
  }
}
```

```
void loop()
{
  // If it's been more than 1.5 seconds since the last state sent
  if (millis() - lastStateSent >= 1500)
  {
    // send the state again
    updateProgram = 1;
  }
```

07/30/2023
Wesley Cerda

Autonomous WIL


```
// If in remote control mode
if (ledState == 1)
{
  // Read the pots
  leftPot = analogRead(A1) / 1023.0;
  rightPot = analogRead(A2)/ 1023.0;

  // Rescale reading. Motor Range is 1100 - 1900.
  motorValue = (leftPot) * 800 + 1100;
  // Rescale reading. Bias is between -400 and 400.
  turnBias   = (rightPot) * 800 - 400;

  leftSpeed = motorValue - turnBias;
  rightSpeed = motorValue + turnBias;

  // Bounds checking...
  if (leftSpeed > 1900)
    leftSpeed = 1900;
  if (leftSpeed < 1100)
    leftSpeed = 1100;

  if (rightSpeed > 1900)
    rightSpeed = 1900;
  if (rightSpeed < 1100)
    rightSpeed = 1100;

  // Don't send if we already sent this value
  if (leftSpeed != leftSpeedPre or rightSpeed != rightSpeedPre)
  {
    LoRa.beginPacket();
    LoRa.print('S');
    LoRa.print(leftSpeed);
    LoRa.print(rightSpeed);
    LoRa.print('E');
    LoRa.endPacket();
  }

  // Keep track of the previous values
  leftSpeedPre = leftSpeed;
  rightSpeedPre = rightSpeed;
}

// We need to change the mode of the program
// and send it to the receiver.
if (updateProgram)
{
  // Update the LED with the current state.
  digitalWrite(6, ledState);

  // Send the state to the receiver
  LoRa.beginPacket();
  LoRa.print('M');
  LoRa.print(ledState);
  LoRa.print('E');
  LoRa.endPacket();

  // Update the last time sent and clear the flag.
  lastStateSent = millis();
  updateProgram = 0;
 }
}

void changeProgramMode()
{
  // When the button is pressed, this changes the state of the led
  // and sets the updateProgram flag.
  ledState = !ledState;
  updateProgram = 1;
}
```

Autonomous WIL

07/30/2023
Wesley Czeki

Results from Lake Olmstead

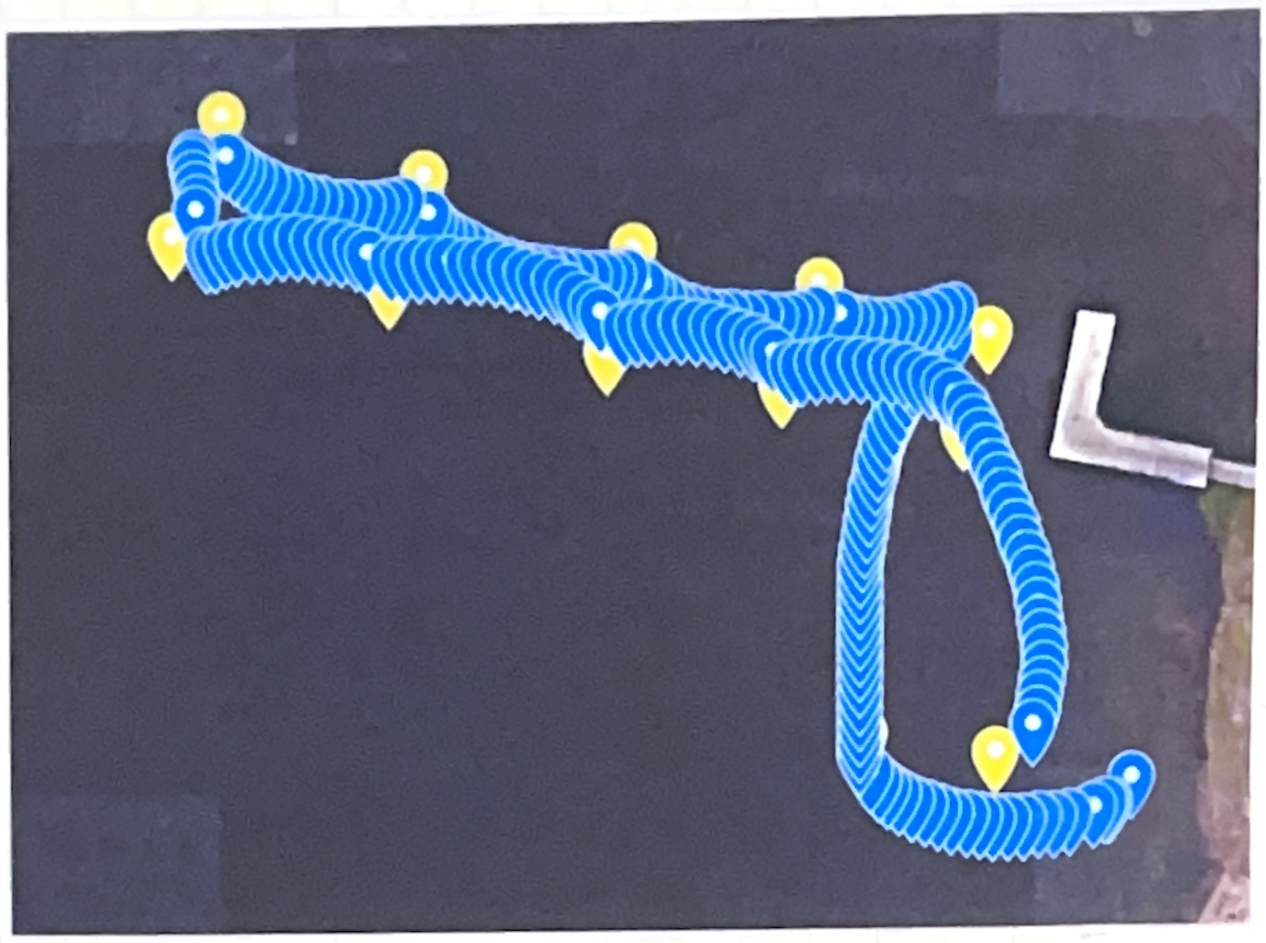


Figure 1

When the heading of WIL is not properly aligned with north, WIL will arc between waypoints.

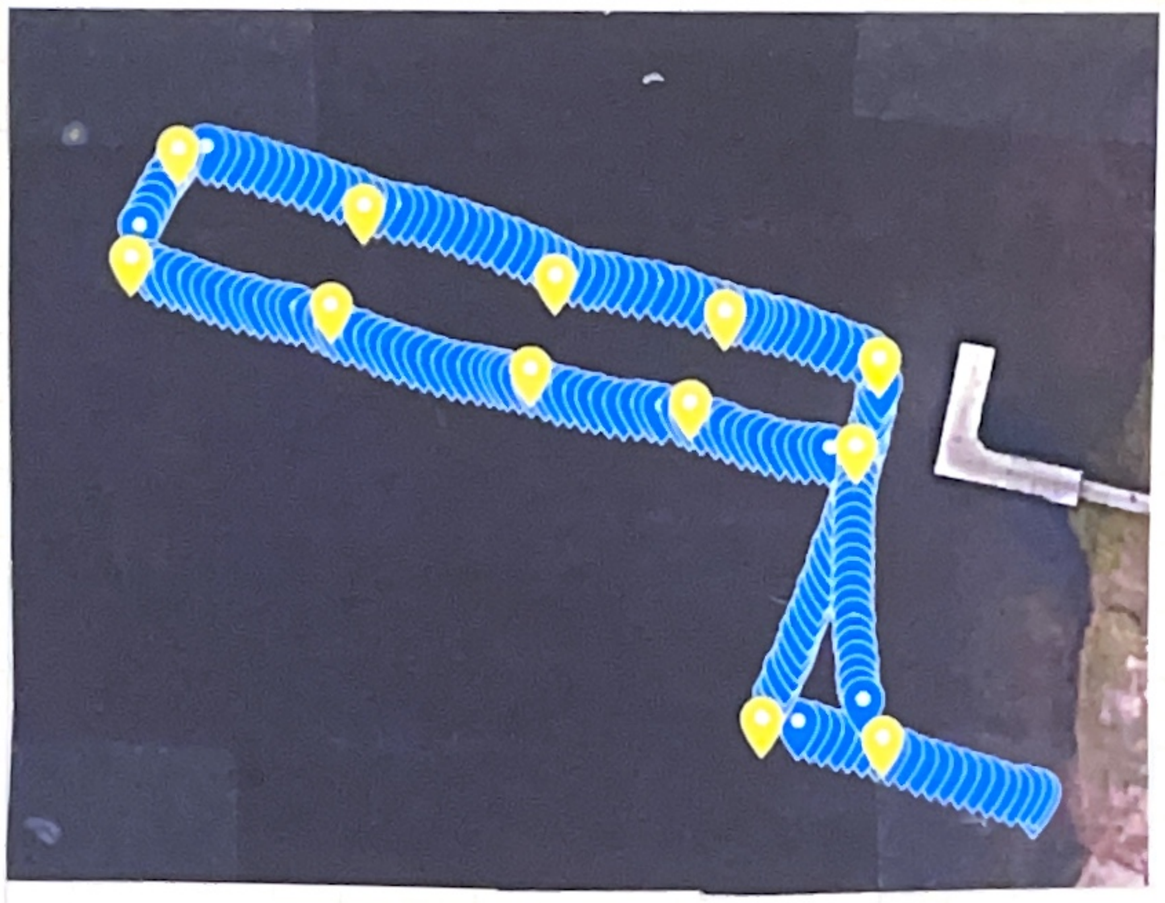


Figure 2

When the heading of WIL is better aligned with north, WIL will take a more straight path between waypoints.

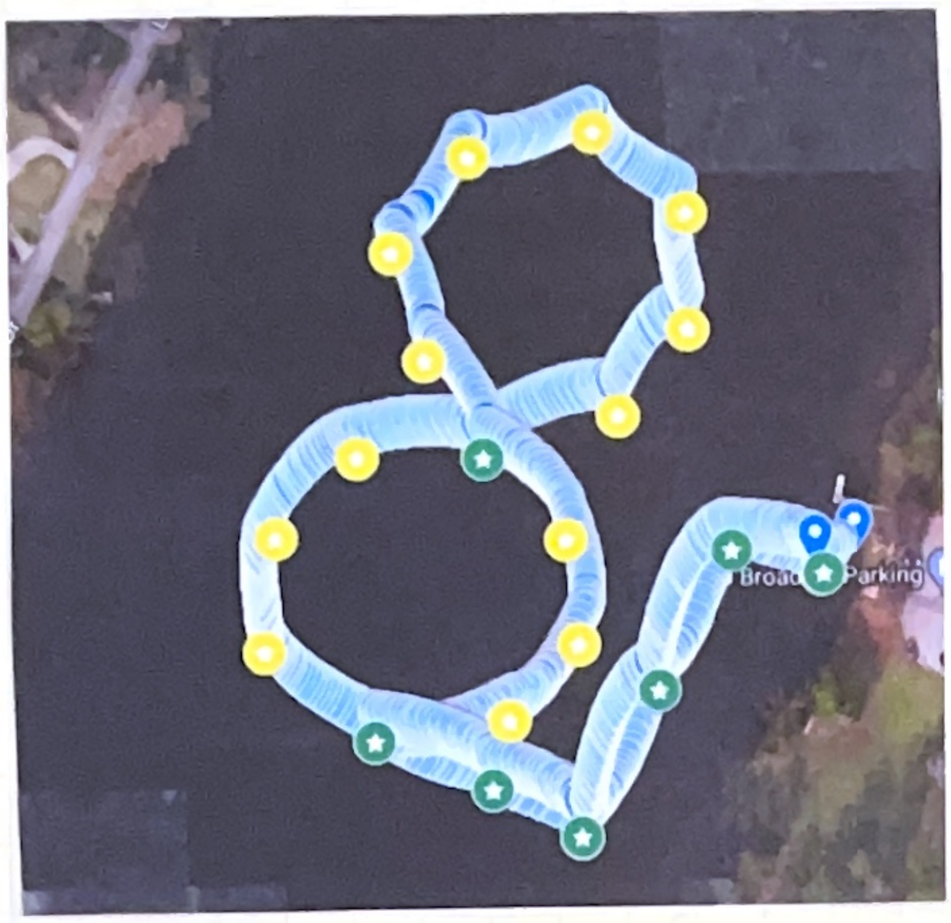


Figure 3

Testing WIL on a Larger figure 8 Path.

Figure 4

Painting the out riggers

Figure 5

Painting the hull

Autonomous WIL

07/30/2023
Wesley Cooke

Pictures by Erin Schmidt

Left to Right: Wesley Cooke, Brycen Havens, Dr. Andy Hauger

Objective

Make the code easier to understand and maintain/modify.

To explore ways of removing the "arcs" between waypoints, the land rover version of WIL will be used. I am atempting to write some libraries to ① learn how libraries work and ② make the code easier to read and modify in the future.

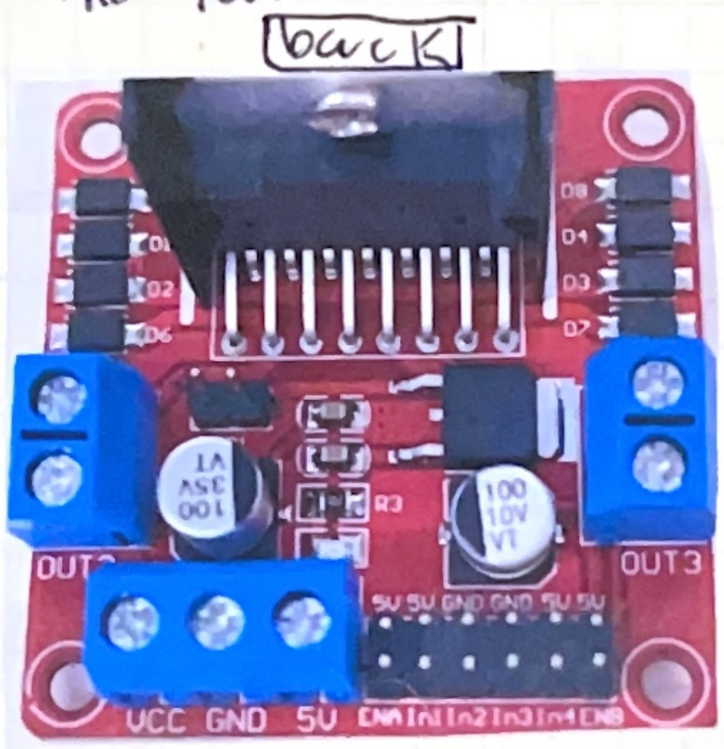


Figure 1

This is an L298N h-bridge. It can control two different motors. One on the left and right.

ENA requires a PWM signal and will adjust the speed of the motor connected to Out 3.

IN1 and IN2 will control which way the current flows, causing the motor to spin CW or CCW. These are digital pins.

For example:

IN1	IN2	Direction
LOW	LOW	None
LOW	HIGH	CW
HIGH	LOW	CCW
HIGH	HIGH	None

Depending on the order of IN1, IN2, and which way the motors are wired into OUT3 may affect the direction.

Writing an H-Bridge Library

Dec 2023
Wesley Loder

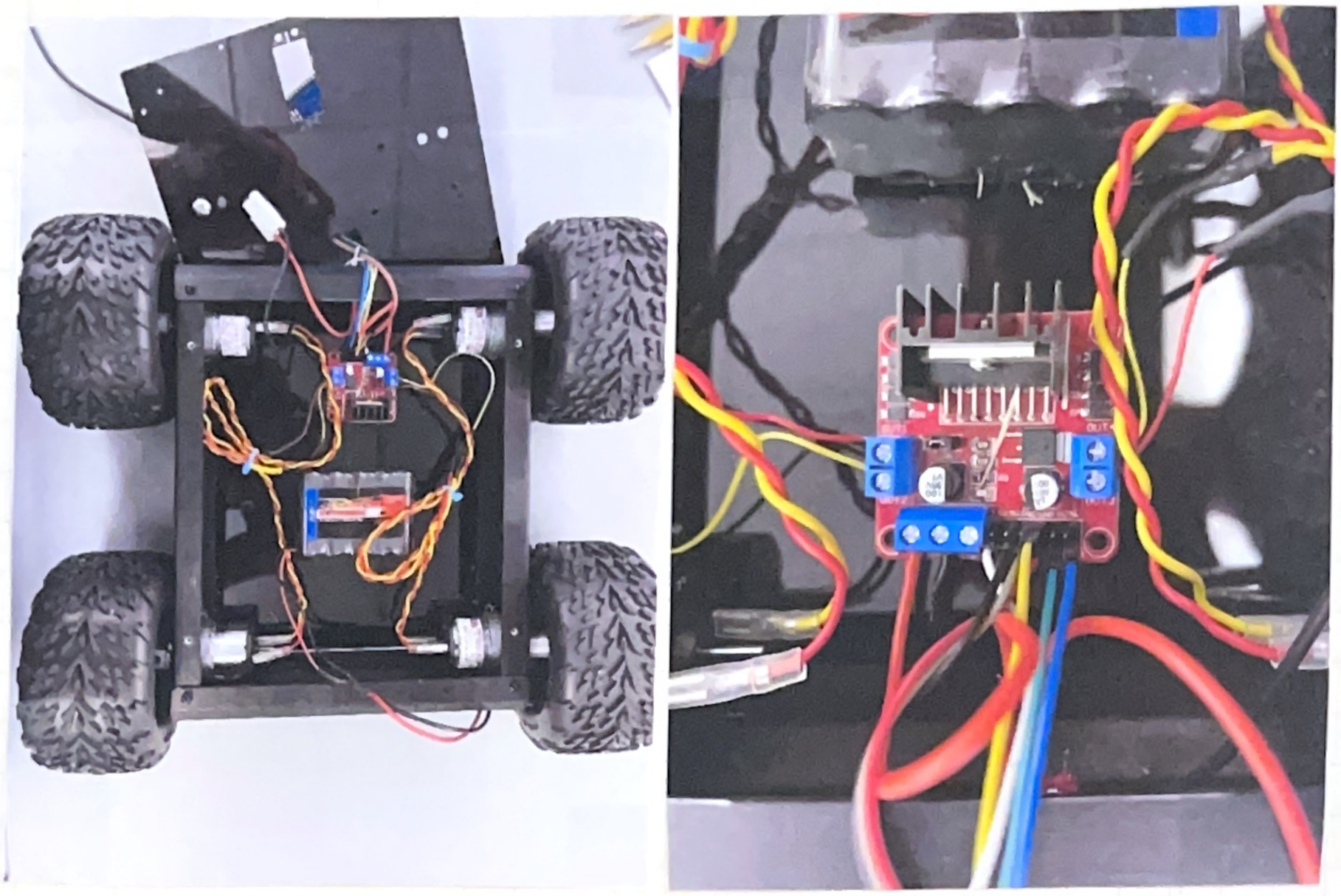
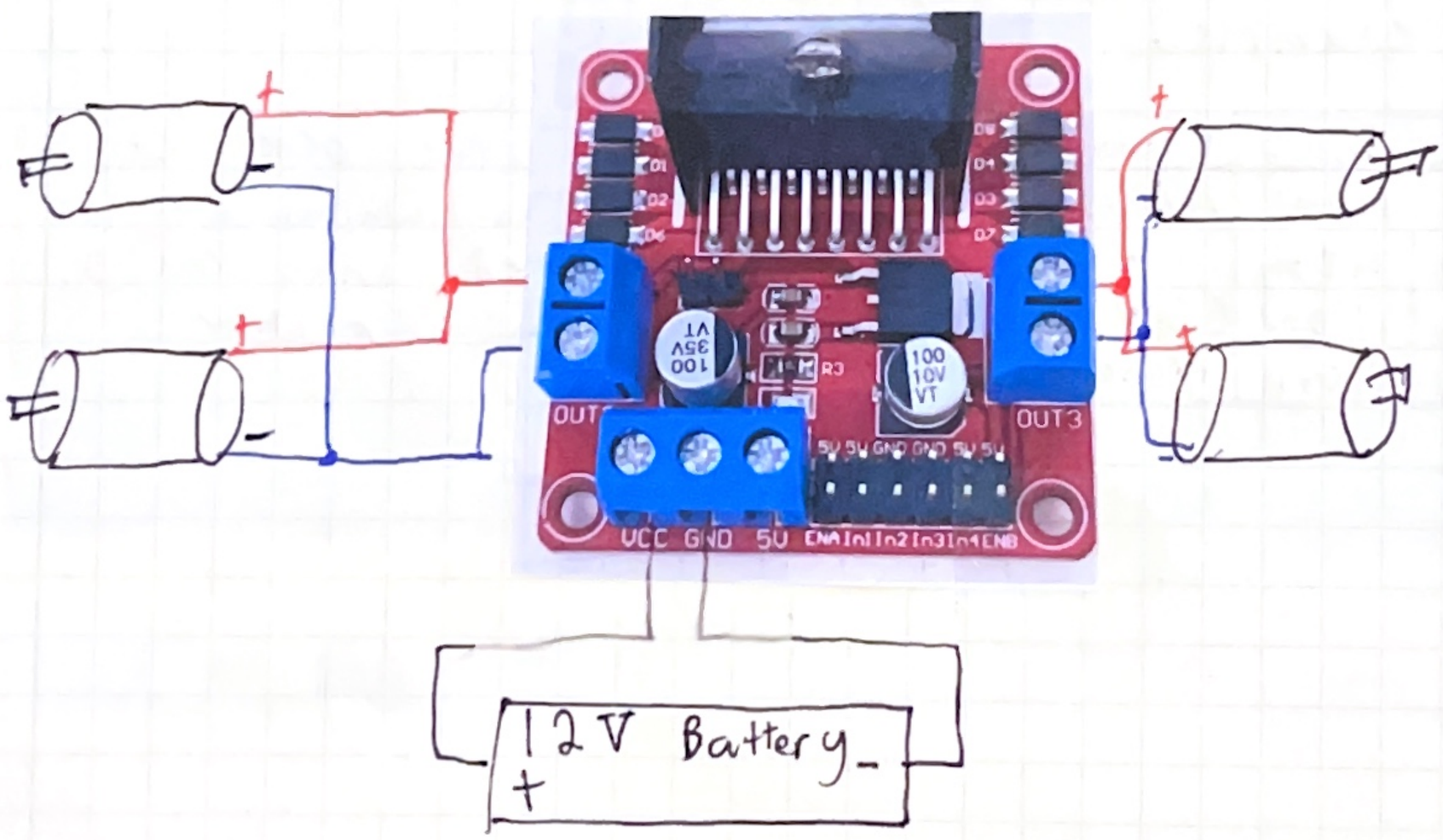


Figure 2

Upgraded WIL Land Chassis.
We got faster moters and bigger wheels.
Notice that we are driving two motors from one side of the h-bridge.



The following connections are made on the giga:

PIN	GIga
ENA	7
ENB	2
IN1	49
IN2	47
IN3	51
IN4	53
GND	GND

Wiring Table

Note that VCC is powered using a 12V battery.

The giga was powered using a rechargable battery.

Writing the Library

The first step is to write a header file. This will define what we plan to implement in the .cpp file. Place this in a folder labeled "src".

MotorControl.h:

```
#ifndef MOTORCONTROL_H
#define MOTORCONTROL_H

#include <Servo.h>
#include <Arduino.h>

const int FORWARD = 1;
const int REVERSE = 0;

class HBridgeControl
{
    public:
        HBridgeControl(int ENA, int ENB, int IN1, int IN2, int IN3, int IN4);
        void SetSpeed(int leftSpeed, int rightSpeed);
        void SetDirection(int leftDir, int rightDir);

        int leftENA;
        int rightENA;
        int leftDir1;
        int leftDir2;
        int rightDir1;
        int rightDir2;
};

#endif
```

Writing an H-Bridge Library

Dec 2023
Wesley Cooke

This file is a fancy way of telling arduino that we plan on implementing an object called "HBridgeControl". It will have a constructor that requires the pins we are using. We will implement functionality for the "SetSpeed" and "SetDirection" methods.

In a new file called "MotorControl.cpp" we can implement our constructor.

```
#include "MotorControl.h"

HBridgeControl::HBridgeControl(int ENA, int ENB, int IN1, int IN2, int IN3, int IN4
{
    rightENA = ENA;
    leftENA = ENB;
    rightDir1 = IN1;
    rightDir2 = IN2;
    leftDir1 = IN3;
    leftDir2 = IN4;


    pinMode(leftENA, OUTPUT);
    pinMode(rightENA, OUTPUT);
    pinMode(leftDir1, OUTPUT);
    pinMode(leftDir2, OUTPUT);
    pinMode(rightDir1, OUTPUT);
    pinMode(rightDir2, OUTPUT);
}
```

Notice that we include our header file we just created.

The constructor will always have the same name as the class. Notice that we have to place "HBridgeControl<u>::</u>" before the constructor name. This is to specify that we are implementing a method specifically for the HBridge control class.

We pass in all of our pins and set them as variables. Then we ensure that they are outputs.

Next, we can implement our functionality:

```
void HBridgeControl::SetSpeed(int leftSpeed, int rightSpeed)
{
    analogWrite(leftENA, leftSpeed);
    analogWrite(rightENA, rightSpeed);
}

void HBridgeControl::SetDirection(int leftDir, int rightDir)
{
    if (leftDir==FORWARD)
    {
        digitalWrite(leftDir1, HIGH);
        digitalWrite(leftDir2, LOW);
    }
    else if (leftDir == REVERSE)
    {
        digitalWrite(leftDir2, HIGH);
        digitalWrite(leftDir1, LOW);
    }

    if (rightDir == FORWARD)
    {
        digitalWrite(rightDir1, HIGH);
        digitalWrite(rightDir2, LOW);
    }
    else if (rightDir == REVERSE)
    {
        digitalWrite(rightDir2, HIGH);
        digitalWrite(rightDir1, LOW);
    }
}

#endif
```

Notice again the use of HBridgeControll::SetSpeed

Lets break it down:

return type

Void HBridgeControl::SetSpeed

Class we are Implementing the method of

Method name we want to implement

Notice how the prototype of the function is inside the header file we created earlier.

Writing an H-Bridge Library

Dec 2023
Wesley Code

Example Driver code:

```
#include "src/MotorControl/MotorControl.h"

// You may have to play around with the order of the IN3, IN4, and IN2, IN1
// to get the right directions, Or you can change the circuit
// I'm using ENA for the Right side motors, and ENB for the left side motors.
//                                          ENA, ENB, IN2, IN1, IN3, IN4
HBridgeControl myHBridge = HBridgeControl(  7,   2,  47,  49,  51,  53);
```

Class name — Variable name

This creates an "instance" of our class. "constructor"

```
void setup()
{
  // Turn the motors off.
  myHBridge.SetSpeed(0, 0);//Left, right
  // Set both the motors to forward
  myHBridge.SetDirection(FORWARD, FORWARD);
```

Left — Right

```
  //Serial for debugging
  Serial.begin(9600);
  while(!Serial);
  Serial.println("#GetObsessed");
  delay(2000);
}

void loop()
{
  Serial.println("Testing forwards:");
  myHBridge.SetDirection(FORWARD, FORWARD);
  myHBridge.SetSpeed(255, 255);
  delay(2000);

  Serial.println("Testing Reverse:");
  myHBridge.SetDirection(REVERSE, REVERSE);
  myHBridge.SetSpeed(255, 255);
  delay(2000);

  Serial.println("Testing Pivot Right:");
  myHBridge.SetDirection(FORWARD, REVERSE);
  myHBridge.SetSpeed(255, 255);
  delay(2000);

  Serial.println("Testing Pivot Left:");
  myHBridge.SetDirection(REVERSE, FORWARD);
  myHBridge.SetSpeed(255, 255);
  delay(2000);

  Serial.println("Done testing. Freezing. Press reset to test again!");
  myHBridge.SetSpeed(0, 0);
  while(1);
}
```

* Note that the orientation you choose may flip which direction Left/right are.

Writing an ESC Library

We are going to extend the motor control H-Bridge library to include an option for Electronic Speed controllers. See page 43 for more details on the ESC.

First, let's modify "MotorControl.h" to tell arduino what we are planning to implement.

```
class ESCControl
{
    private:
        int leftServoPin;
        int rightServoPin;
        Servo leftServo;
        Servo rightServo;

    public:
        ESCControl();
        void setup(int leftControlPin, int rightControlPin);
        void SetSpeed(int leftSpeed, int rightSpeed);
};

#endif
```

*

Here we are making a class called "ESC Control." The private variables are only accessable from inside the class definition. Public variables and methods can be called from an instance of the class.

Here, we are going to have an empty constructor, and two public methods: "Setup" and "set speed"

"Setup" and "SetSpeed" are examples of prototypes. They give an outline of a method without actually implementing it.

Next, let's modify "MotorControl.cpp"

```
ESCControl::ESCControl() {}
void ESCControl::setup(int leftControlPin, int rightControlPin)
{
    pinMode(leftControlPin, OUTPUT);
    pinMode(rightControlPin, OUTPUT);

    // Save the passed in variables as instance variables
    leftServoPin = leftControlPin;
    rightServoPin = rightControlPin;

    // Begin servo objects.
    leftServo.attach(leftServoPin);
    rightServo.attach(rightServoPin);

    // This should initalize the ESC.
    // Make sure you hear the startup sequence complete.
    // Otherwise you need to try again.
    leftServo.writeMicroseconds(1500);
    rightServo.writeMicroseconds(1500);
    delay(10000);
}
void ESCControl::SetSpeed(int CurrentLeftSpeed, int CurrentRightSpeed)
{
    // For our ESC, this has a range of 1100 - 1900.
    // 1500 is off, 1900 is fully forwards, and 1100 is fully reversed.
    rightServo.writeMicroseconds(CurrentRightSpeed);
    leftServo.writeMicroseconds(CurrentLeftSpeed);
}
```

Here, we have an empty constructor so that we can choose when our ESC's should begin.

This is because of their special startup sequence

Example Driver Code

```
#include "src/MotorControl/MotorControl.h"

ESCControl myESC = ESCControl();
int mySpeed;

void setup()
{
  // Use pins 2 and 7 for the control. Must be PWM pins.
  // Left Motor is using Pin 2, Right Motor is using Pin 7.
  myESC.setup(2, 7);
  // Serial Monitor Output
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Input an int speed: ");
  Serial.println("1500 will stop the motors.");
  Serial.println("1525 - 1900 will go forwards.");
  Serial.println("1100 - 1475 will go backwards.");

  while(!Serial.available())
  mySpeed = Serial.parseInt();

  if (mySpeed < 1100 or mySpeed > 1900)
  {
    Serial.println("Invalid speed detected!");
    mySpeed = 1500;
  }

  Serial.print("Speed will be set to: "); Serial.print(mySpeed); Serial.println();
  Serial.println("Testing the left motor for 5 seconds: ");
  myESC.SetSpeed(mySpeed, 1500);
  delay(5000);

  Serial.println("Testing the right motor for 5 seconds: ");
  myESC.SetSpeed(1500, mySpeed);
  delay(5000);

  Serial.print("Stopping all motors.");
  myESC.SetSpeed(1500, 1500);
}
```

Writing a GPS Library

The SparkFun GNSS library is very well written. However, it contains a lot of extra functionality and configuration. We are going to write a library that uses the SparkFun one, but only exposes what we need.

Let's start by making "GPS.h".

```
#ifndef GPS_H
#define GPS_H

#include <SparkFun_u-blox_GNSS_v3.h>

// The main data that we care to extract from the GPS
struct GPSData
{
    long lattitude;
    long longitude;
    byte SIV;      // satellites in view
    long heading; // Relative to magnetic north
    long speed;   // Ground Speed
};

class GPS
{
    private:
        SFE_UBLOX_GNSS_SERIAL myGNSS;
    public:
        void GetData(GPSData &data);
        bool setupGPS(Stream &theSerial);
        GPS();
};

#endif
```

The struct, you can think of it as Class that only has variables associated with it. No methods.

Another way is to think of it as a grour of variables ~~that can be acces~~ that are related and can be accessed with a common name.

Writing GPS Library

Dec 2023
Wesus Cooke

GPS.cpp:

```cpp
#include "gps.h"

GPS::GPS(){}

bool GPS::setupGPS(Stream &theSerial)
{
    if (!myGNSS.begin(theSerial))
    {
        return false;
    }

    // Set some settings to reduce noise on the communication lines
    myGNSS.setUART1Output(COM_TYPE_UBX);
    myGNSS.setI2COutput(COM_TYPE_UBX);
    myGNSS.saveConfiguration();
    return true;
}

void GPS::GetData(GPSData &data)
{
    // Here, we have passed the GPSData struct by reference.
    // So we fill it up with the new data and it is accessible
    // using that same object.
    data.lattitude = myGNSS.getLatitude();
    data.longitude = myGNSS.getLongitude();
    data.SIV       = myGNSS.getSIV();
    data.heading   = myGNSS.getHeading();
    data.speed     = myGNSS.getGroundSpeed();
}
```

On setup we pass in a reference to the serial port we are using. That is what the "&" is saying. When we get the data, we also pass in a reference to the memory where we want the variables to go.

Example Driver Code

```
#include "src/GPS/gps.h"

// Create a GPS Object and data holder
GPS myGPS = GPS();
GPSData myGPSData;

void printGPSData(GPSData &theGPSData)
{
    // Here we are passing the GPSData by reference
    Serial.print(theGPSData.lattitude);
    Serial.print(" ");
    Serial.print(theGPSData.longitude);
    Serial.print(" ");
    Serial.print(theGPSData.SIV);
    Serial.print(" ");
    Serial.print(theGPSData.heading);
    Serial.print(" ");
    Serial.print(theGPSData.speed);
    Serial.println();
}
void setup()
{
    // Serial monitor for output.
    Serial.begin(9600);
    while(!Serial); // Wait for serial monitor to open.
    Serial.println("#GetObsessed!");

    // Tell the micro controller that
    // the gps is connected to Serial1.
    Serial1.begin(38400);
    if (!myGPS.setupGPS(Serial1))
    {
        // If the setup fails, we freeze.
        digitalWrite(LED_BUILTIN, HIGH);
        Serial.println("GPS failed to begin.");
        while(1);
    }
}
void loop()
{
    // Tell the GPS to put GPSData inside
    // the myGPSData struct.
    myGPS.GetData(myGPSData);

    // Print out all the data
    printGPSData(myGPSData);
}
```

Example Output:

```
334679730 -819909447 6 0 74
334679640 -819909470 6 0 161
334679705 -819909473 6 0 88
334679716 -819909480 6 0 164
334679726 -819909555 6 0 15
334679730 -819909561 6 0 18
334679728 -819909553 6 0 40
334679729 -819909542 6 0 81
334679728 -819909410 11 0 36
```

Due to the Observation on page 92, we are trying the BNO 085 and using a different library until we can determine how to fix the BNO 085.

"BNO.h" Header File

```
#ifndef BNO_H
#define BNO_H

#include <Adafruit_BNO08x_RVC.h>

class BNO
{
    private:
        int bnoRST;
        Stream *streamPtr;
        Adafruit_BNO08x_RVC myIMU;
        BNO08x_RVC_Data bnoData;
    public:
        BNO();
        bool setupBNO(Stream *theSerial, int aResetPin);
        double GetHeading();
};

#endif
```

Writing a BNO Library

Dec 2023

BNO.cpp:

```cpp
#include "BNO.h"

BNO::BNO(){}

bool BNO::setupBNO(Stream *theStream, int aResetPin)
{
    myIMU = Adafruit_BNO08x_RVC();
    streamPtr = theStream;
    bnoRST = aResetPin;

    if (!myIMU.begin(theStream))
        return false;

    pinMode(bnoRST, OUTPUT);
    digitalWrite(bnoRST, LOW);
    delay(10);
    digitalWrite(bnoRST, HIGH);
    return true;
}
```

Here, we implement an empty constructor and instead let the user call the "setupBNO" method.

Notice that we are really just "wraping" Adafruits library with our own extra functionality.

```
double BNO::GetHeading()
{

    while (streamPtr->available() > 0)
    {
        streamPtr->read();
    }

    // Continue reading until we get a new heading.
    while(!myIMU.read(&bnoData)) {}

    // Adjust magnetic reading to get geographic north.
    // float heading = bnoData.yaw - declination;

    // Adjust using GPS offset
    double heading = bnoData.yaw; //+ headingOffset;

    // Make sure we get a value between 0-360
    while (heading < 0)
    {
        heading += 360;
    }

    while (heading > 360)
    {
        heading -= 360;
    }

    return heading;
}
```

Here, we implemet how to get the heading.

Writing a BNO Library

Dec 2023 [illegible] code

Sample Driver Code and Output:

```
#include "src/BNO/BNO.h"

BNO myBNO = BNO();
int bnoResetPin = 6;
double currentHeading;

void setup()
{
  // Local Debugging Serial monitor
  Serial.begin(9600);
  while(!Serial);
  Serial.println("#GetObsessed!");

  // Start the BNO on Serial2 with the reset pin
  Serial2.begin(115200);
  if (!myBNO.setupBNO(&Serial2, bnoResetPin))
  {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("BNO Failed to start");
    while(1);
  }
}

void loop()
{
  currentHeading = myBNO.GetHeading();
  Serial.println(currentHeading);
  delay(500);
}
```

```
13:27:27.888 -> #GetObsessed!
13:27:27.888 -> 359.98
13:27:28.408 -> 359.98
13:27:28.921 -> 359.98
13:27:29.442 -> 359.98
13:27:29.915 -> 359.97
13:27:30.437 -> 359.97
13:27:30.952 -> 353.39
13:27:31.472 -> 348.17
13:27:31.967 -> 341.65
13:27:32.479 -> 350.21
13:27:32.976 -> 7.96
13:27:33.498 -> 9.86
13:27:34.020 -> 9.60
13:27:34.542 -> 9.23
13:27:35.050 -> 9.23
```

Writing a Heading control Library

With a source of GPS coordinates and a way to tell which direction we are facing, we can now combine that information to figure out how to get from one location to the next.

Heading Control.H:

```
#ifndef HEADINGCONTROL_H
#define HEADINGCONTROL_H

struct GPSPair
{
    // Lat1 and Lon1 can be viewed as the "current position".
    // Lat2 and Lon2 can be veiwed as the "target position".
    long lat1_L; // Lattitude and longigutes
    long lon1_L; // The _L indicates they are
    long lat2_L; // long data types. Not
    long lon2_L; // in decimal form yet.
};

class HeadingControl
{
    public:
        double GetDistanceBetween(GPSPair &someGPSPoints);
        double GetTargetHeading(GPSPair &someGPSPoints);
        double GetHeadingError(double CurrentHeading, double TargetHeading);
        HeadingControl();
};

#endif
```

Notice that we care about 3 functionalities:

What is the distance between a pair of GPS coords?
What heading do we need to go from pair 1 to pair 2?
What is the difference in our current and target heading?

Writing a Heading Control Library

Dec 2023
Working Code

HeadingControl.cpp:

```cpp
#include "HeadingControl.h"
#include <math.h>
#include <Arduino.h>

HeadingControl::HeadingControl(){}

double HeadingControl::GetDistanceBetween(GPSPair &someGPSPoints)
{
  // returns distance in meters between two positions, both specified
  // as signed decimal-degrees latitude and longitude. Uses great-circle
  // distance computation for hypothetical sphere of radius 6372795 meters.
  // Because Earth is no exact sphere, rounding errors may be up to 0.5%.
  // Courtesy of Maarten Lamers

  // Convert lat and long to degrees
  double lat1   =  (double)someGPSPoints.lat1_L  / 10000000.0;
  double long1  =  (double)someGPSPoints.lon1_L  / 10000000.0;
  double lat2   =  (double)someGPSPoints.lat2_L  / 10000000.0;
  double long2  =  (double)someGPSPoints.lon2_L  / 10000000.0;
  double delta  = radians(long1-long2);
  double sdlong = sin(delta);
  double cdlong = cos(delta);
  lat1          = radians(lat1);
  lat2          = radians(lat2);
  double slat1  = sin(lat1);
  double clat1  = cos(lat1);
  double slat2  = sin(lat2);
  double clat2  = cos(lat2);
  delta         = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta         = sq(delta);
  delta        += sq(clat2 * sdlong);
  delta         =  sqrt(delta);
  double denom  = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta         = atan2(delta, denom);
  return delta * 6372795;
}
```

```cpp
double HeadingControl::GetTargetHeading(GPSPair &someGPSPoints)
{
  // returns course in degrees (North=0, West=270) from position 1 to position 2,
  // both specified as signed decimal-degrees latitude and longitude.
  // Because Earth is no exact sphere, calculated course may be off by a tiny fraction.
  // Courtesy of Maarten Lamers

  // Convert lat and long to degrees
  double lat1   =  (double)someGPSPoints.lat1_L  / 10000000.0;
  double long1  =  (double)someGPSPoints.lon1_L  / 10000000.0;
  double lat2   =  (double)someGPSPoints.lat2_L  / 10000000.0;
  double long2  =  (double)someGPSPoints.lon2_L  / 10000000.0;
  double dlon   = radians(long2-long1);
  lat1          = radians(lat1);
  lat2          = radians(lat2);
  double a1     = sin(dlon) * cos(lat2);
  double a2     = sin(lat1) * cos(lat2) * cos(dlon);
  a2            = cos(lat1) * sin(lat2) - a2;
  a2            = atan2(a1, a2);

  if (a2 < 0.0)
  {
    a2 += TWO_PI;
  }
  return degrees(a2);
}
```

```
double HeadingControl::GetHeadingError(double aHeading, double aTargetHeading)
{
    // Computes the difference between aHeading and aTargetHeading.
    // Returns a value between -1 and 1 representing how away the two headings are.

    double error = aTargetHeading - aHeading;

    // Give the shortest turn distance from -180 to 180.
    if (error >= 180)
    {
        error -= 360;
    }
    else if (error <= -180)
    {
        error += 360;
    }

    // Return error number between -1 and 1.
    return error/180.0;
}
```

Sample Driver Code:

```
#include "src/HeadingControl/HeadingControl.h"
#include "src/GPS/gps.h"
#include "src/BNO/BNO.h"

// Objects
HeadingControl myHeadingControl = HeadingControl();
GPS            myGPS            = GPS();
BNO            myBNO            = BNO();

// Data Structs
GPSData        myGPSData;
GPSPair        myGPSPair;

// Set a destination here using a LONG variable.
// This is inbetween the garages at the Pi Lab.
long destinationLat  =  334672860;
long destinationLon  =  -819910550;

// Variables we want to get from heading control
double distanceBetween;
double targetHeading;
double headingError;

int bnoResetPin = 6;
double currentHeading;
```

Writing a Heading Control Library

Dec 2023
Working code

```
11:36:49.119 -> Distance: 61.79
11:36:49.119 -> TargetHeading: 210.26
11:36:49.119 -> CurrentHeading: 213.59
11:36:49.119 -> Error: -0.02
11:36:49.119 ->
11:36:50.129 -> Distance: 61.54
11:36:50.129 -> TargetHeading: 210.25
11:36:50.129 -> CurrentHeading: 213.59
11:36:50.129 -> Error: -0.02
11:36:50.129 ->
11:36:51.118 -> Distance: 61.35
11:36:51.118 -> TargetHeading: 210.22
11:36:51.118 -> CurrentHeading: 213.59
11:36:51.118 -> Error: -0.02
11:36:51.118 ->
```

```
void setup()
{
  // Serial monitor for output.
  Serial.begin(9600);
  Serial.println("GetObsessed!");

  // Connect the GPS to Serial1.
  // This is RX0 and TX0 on the giga board.
  Serial1.begin(38400);
  if(!myGPS.setupGPS(Serial1))
  {
    // Freeze if setup fails
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("GPS Failed to begin.");
    while(1);
  }

  // Connect the BNO to Serial2
  Serial2.begin(115200);
  if(!myBNO.setupBNO(&Serial2, bnoResetPin))
  {
    // Freeze if BNO failed
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("BNO Failed to start");
    while(1);
  }

  // Set the "destination" to point 2 in the data structure.
  myGPSPair.lat2_L = destinationLat;
  myGPSPair.lon2_L = destinationLon;

}
```

```
void loop()
{
  // Get an update from the GPS.
  myGPS.GetData(myGPSData);

  // Set the current position to point 1 in the data structure.
  myGPSPair.lat1_L = myGPSData.lattitude;
  myGPSPair.lon1_L = myGPSData.longitude;

  // Get the pieces of information that we care about.
  distanceBetween = myHeadingControl.GetDistanceBetween(myGPSPair);
  targetHeading   = myHeadingControl.GetTargetHeading(myGPSPair);
  currentHeading  = myBNO.GetHeading();
  headingError    = myHeadingControl.GetHeadingError(currentHeading, targetHeading);

  // Print the information
  Serial.print("Distance: ");        Serial.println(distanceBetween);
  Serial.print("TargetHeading: ");   Serial.println(targetHeading);
  Serial.print("CurrentHeading: ");  Serial.println(currentHeading);
  Serial.print("Error: ");           Serial.println(headingError);     Serial.println();
}
```

Fixing the Arc

When we align WIL with North, there is a sight error that causes arcs between GPS points as seen on page 114 and 115.

To eliminate these arcs, we need to eliminate this error. How do we do this?

~~Whe~~ We have a heading from our BNO Chip. This heading is good to read at any time and doesn't depend on the speed of WIL.

We also get a heading from the GPS. However, the heading from the GPS is not accurate unless WIL is moving fast enough.

What we might be able to do is compute the average error in the heading that the BNO reports and the heading the GPS reports over the course of some amount of time.

This average difference is hopefully our "error" term that is introduced from not perfectly aligning WIL with North.

Note that this will ensure we are aligned with True North and not Magnetic north.

Let's write a library that can keep track of this error. Note that the average of 0° and 360° when dealing with headings should be 0°. The average of 0° and 180° should be 90.

* We are also going to use a Circular Array. This means, if we try to add more values than the array can hold, we will wrap around to the start of the array and begin over writing old values.

Fixing the Arc

Dec 2023
Wisely coda

CircularArray.H

```cpp
// CircularArray.h

#ifndef CIRCULARARRAY_H
#define CIRCULARARRAY_H

#include <Arduino.h>

class CircularArray {
public:
    CircularArray();
    ~CircularArray();

    void Update(double value);
    void ResetAverage();
    double GetAverage() const;

private:
    double* sinValues;     // Circular buffer to keep track of the last N double values
    double* cosValues;
    uint16_t limit;       // Limit for the number of values in the array
    uint16_t count;       // Count of the values
    uint16_t frontIndex;  // Index of the front element in the circular buffer
    double sinSum;         // Sum of the values for calculating the average
    double cosSum;
};

#endif // CIRCULARARRAY_H
```

Here, the ~CircularArray(); is a deconstructor. It tells arduino what to do if this object is deleted.

CircularArray.cpp

```cpp
// CircularArray.cpp

#include "CircularArray.h"

CircularArray::CircularArray() : limit(100), count(0), frontIndex(0), sinSum(0), cosSum(0)
{
    sinValues = new double[limit]();
    cosValues = new double[limit]();
}

CircularArray::~CircularArray()
{
    delete[] sinValues;
    delete[] cosValues;
}

void CircularArray::Update(double value)
{
    if (count >= limit) {
        sinSum -= sinValues[frontIndex];
        cosSum -= cosValues[frontIndex];
    } else {
        ++count;
    }
    double sinValue = sin(radians(value));
    double cosValue = cos(radians(value));

    sinSum += sinValue;
    cosSum += cosValue;

    sinValues[frontIndex] = sinValue;
    cosValues[frontIndex] = cosValue;

    frontIndex = (frontIndex + 1) % limit;
}

double CircularArray::GetAverage() const
{
    if (count == 0) {
        return 0.0;
    }

    double returnValue = degrees(atan2(sinSum, cosSum));

    return returnValue;
}

void CircularArray::ResetAverage()
{
    count = 0;
    frontIndex = 0;
    sinSum = 0;
    cosSum = 0;
}
```

Fixing the Arc

Dec 2023
Wiley Coder

Example Driver Code:

```
#include "src/CircularArray/CircularArray.h"

// Create an instance of CircularArray
CircularArray myArray = CircularArray();
double average;

void setup()
{
  Serial.begin(9600);
  while(!Serial);
}

void loop()
{
  Serial.println("Averaging 1 and 359");
  myArray.Update(1);
  myArray.Update(359);
  average = myArray.GetAverage();
  Serial.print("Average: ");      Serial.println(average, 100);

  //Reset the Array
  myArray.ResetAverage();

  Serial.println("Averaging 0 and 180");
  myArray.Update(0);
  myArray.Update(180);
  average = myArray.GetAverage();

  Serial.print("Average: "); Serial.println(average, 100);

  while(1);
}
```

```
Averaging 1 and 359
Average: -0.0000000000000(
Averaging 0 and 180
Average: 90.0000000000000(
```

Writing a Controller Library

Controller.H

```cpp
// Controller.h

#ifndef CONTROLLER_H
#define CONTROLLER_H

#include <Arduino.h>

class Controller {
public:
    Controller();

    void ReadController();
    void _read_state();
    void _read_speed();
    void begin();

    int leftMotorSpeedTarget;
    int rightMotorSpeedTarget;
    bool ProgramMode;

private:
    char leftSpeedChars[5];
    char rightSpeedChars[5];

};

#endif // CONTROLLER_H
```

Controller.CPP

```cpp
void Controller::_read_speed()
{
    uint8_t numPackets = Serial3.available();

    while(numPackets < 8)
    {
        numPackets = Serial3.available();
        delay(10);
    }

    for (int i = 0; i < 4; i++)
    {
        leftSpeedChars[i] = Serial3.read();
    }
    leftSpeedChars[4] = '\0';

    for (int i = 0; i < 4; i++)
    {
        rightSpeedChars[i] = Serial3.read();
    }
    rightSpeedChars[4] = '\0';

    if (Serial3.read() == 'E')
    {
        leftMotorSpeedTarget = atoi(leftSpeedChars);
        rightMotorSpeedTarget = atoi(rightSpeedChars);

        if (leftMotorSpeedTarget < 1100)
            leftMotorSpeedTarget = 1100;
        if (leftMotorSpeedTarget > 1900)
            leftMotorSpeedTarget = 1900;

        if (rightMotorSpeedTarget < 1100)
            rightMotorSpeedTarget = 1100;
        if (rightMotorSpeedTarget > 1900)
            rightMotorSpeedTarget = 1900;

    }
}

void Controller::ReadController()
{
  while (Serial3.available())
  {
      char start = Serial3.read();
      if (start == 'S')
      {
          _read_speed();
      }
      else if (start == 'M')
      {
          _read_state();
      }
  }
}
```

```cpp
#include "Controller.h"

Controller::Controller(){}

void Controller::begin()
{
    Serial3.begin(115200);
    ProgramMode = 0;
    leftMotorSpeedTarget = 1500;
    rightMotorSpeedTarget = 1500;
}

void Controller::_read_state()
{
    char data = Serial3.read();

    if (Serial3.read() == 'E')
    {
        if (data == '0')
        {
            ProgramMode = 0;

            //Serial.println("Switched to autonomous mode.");
        }
        else if (data == '1')
        {
            ProgramMode = 1;

            //Serial.println("Switched to remote control mode.");
        }
    }
}
```

Code from page 108 + 109, but formatted as a library.

Putting it all Together Land


```
#include "src/BNO/BNO.h"
#include "src/GPS/gps.h"
#include "src/HeadingControl/HeadingControl.h"
#include "src/MotorControl/MotorControl.h"
#include "src/CircularArray/CircularArray.h"
#include <SD.h>

// Objects we will use                                ENA, ENB, IN2, IN1, IN3, IN4,
HBridgeControl myHBridge        = HBridgeControl(7,   2,   47,   49,   51,   53); // Motor Control
HeadingControl myHeadingControl = HeadingControl(); // Heading Controller
CircularArray  myArray          = CircularArray();  // Array to keep track of drift.
GPS            myGPS            = GPS();            // Our GPS Sensor
BNO            myBNO            = BNO();            // Our Acc., Gyro., Mag., sensor.


// Data Structures we will use
GPSData        myGPSData;
GPSPair        myGPSPair;

// Important Variables
double distanceBetween;        // Distance between current lat/lon and destination lat/lon
double targetHeading;          // The heading we need to travel at to reach destination
double currentHeading;         // The heading we are traveling at
double headingError;           // Difference in target and current heading
double gpsError;               // difference in myGPSData.Heading and currentHeading
double averageGPSError = 0;    // Average difference in myGPSData.Heading and currentHeading
int    bnoResetPin     = 6;    // Pin to reset the BNO chip
int    SDChipSelect    = 10;

// Acceptable Heading Error - 2%
const double ACC_HEAD_ERR = 0.02;

//// Path
long dLons[] =
{
  -819913429,
  -819916915,
  -819910571,
  -819909347,
  -819909936
};

long dLats[] =
{
  334676789,
  334674976,
  334672878,
  334675469,
  334677542
};

int waypointIndex   = 0;
int numWayPoints    = 5;
bool newWayPoint    = 1;
bool updatedAverage = 0;
double totalDistance;
long startTime;
bool GotFirstStartTime = 0;
```

These are all the variables and objects the Program will work with.

Putting it all Together


```
void setup()
{
  Serial.begin(9600);
  Serial.println("#GetObsessed!");
  pinMode(LED_BUILTIN, OUTPUT);

  // setupGPS()
  // Connect the GPS to Serial1.
  // This is RX0 and TX0 on the giga board.
  Serial1.begin(38400);
  if(!myGPS.setupGPS(Serial1))
  {
    // Freeze if setup fails
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("GPS Failed to begin.");
    while(1);
  }

  // setupBNO()
  // Connect the BNO to Serial2
  Serial2.begin(115200);
  if(!myBNO.setupBNO(&Serial2, bnoResetPin))
  {
    // Freeze if BNO failed
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("BNO Failed to start");
    while(1);
  }

  //setupHBridge()
  // Turn the motors off.
  myHBridge.SetSpeed(0, 0);
  // Set both the motors to forward
  myHBridge.SetDirection(FORWARD, FORWARD);

  // SetupSD()
  // Set the SPI ChipSelect pin as an output
  pinMode(SDChipSelect, OUTPUT);
  if(!SD.begin(SDChipSelect))
  {
   // If it fails to begin, freeze.
   // Check that the card is properly inserted.
   digitalWrite(LED_BUILTIN, HIGH);
   Serial.println("SD Card failed to begin.");
   while(1);
  }
  startTime = millis();
}
```

These are all the setup commands we want to do to ensure everything is functioning properly.

Putting it all Together Land


```
void loop()
{
  // Set the "destination" to point 2 in the GPS pair.
  myGPSPair.lat2_L = dLats[waypointIndex];
  myGPSPair.lon2_L = dLons[waypointIndex];

  // Update Our GPS data
  myGPS.GetData(myGPSData);

  // Set the "current" position to point 1 in the GPS pair.
  myGPSPair.lat1_L = myGPSData.lattitude;
  myGPSPair.lon1_L = myGPSData.longitude;

  // Get the information we need to control the robot
  distanceBetween = myHeadingControl.GetDistanceBetween(myGPSPair);
  targetHeading   = myHeadingControl.GetTargetHeading(myGPSPair);
  currentHeading  = myBNO.GetHeading() - averageGPSError;   // would need to flip here too.
  headingError    = myHeadingControl.GetHeadingError(currentHeading, targetHeading);
  gpsError        = currentHeading - (myGPSData.heading / 100000.0);  // could flip

  // Log the data we are about to use to make decisions
  logToSd();
  printVars();
  // If there are more than 4 sats. in view.
  if (myGPSData.SIV > 4)
  {

    // Start the Timer for the first time
    if(GotFirstStartTime == 0)
    {
      startTime          = millis();
      GotFirstStartTime = 1;
    }

    // Keep track of the error
    myArray.Update(gpsError);

    // If we haven't been to this waypoint before.
//    if (newWayPoint)
//    {
//      // Get the distance we are away from it
//      totalDistance = distanceBetween;
//      // Tell the program we have seen this point before.
//      newWayPoint = 0;
//    }

    // If our heading is too far off from the acceptable error
    if (abs(headingError) > ACC_HEAD_ERR)
    {
      myHBridge.SetSpeed(0, 0); // Stop to think
      RotateInPlaceLand();      // Fix Heading
    }
    // If we are within 3.0 m of our destination point.
    if(distanceBetween <= 3.0)
    {
      // Turn Motors Off and go to the next waypoint
      myHBridge.SetSpeed(0, 0);
      waypointIndex++;

      // Tell the program we are going to a new way point
      // So we need to get a new total distance
      newWayPoint = 1;

      // Get the average for this leg and use it for the next one. Reset the average after.
      averageGPSError = myArray.GetAverage() + averageGPSError;
      myArray.ResetAverage();
      startTime = millis();
```

page 34 book 2 for update

The heart of the algorithm...

Putting it all Together Land

Dec 2023 Wendy Carlin

```
      // Tell the program we haven't updated the average for this leg yet.
      // updatedAverage = 0;

      if (waypointIndex >= numWayPoints)
      {
        // Done with path. Freeze.
        while(1);
      }
    }
//    else if ((distanceBetween <= totalDistance/2 and updatedAverage==0) and newWayPoint == 0)
//    { // If we are halfway to a new point and we haven't updated our average yet
//      // Then update the average
//      averageGPSError = myArray.GetAverage() + averageGPSError;
//      // Reset the average for the error
//      myArray.ResetAverage();
//      // Tell the program we have updated the average for this leg
//      updatedAverage  = 1;
//    }
    else if ((millis() - startTime) > 10000) // If we haven't updated GPS average for 10 seconds.
    {
      averageGPSError = myArray.GetAverage() + averageGPSError;
      myArray.ResetAverage();
      startTime = millis();
    }
    else
    {
      // Our heading is not too far off and we are not close enough to the point
      // Go Forwards
      myHBridge.SetSpeed(255, 255);
    }
  }
  else
  {
    // 4 Sats are not in view. Standby...
    myHBridge.SetSpeed(0, 0);
  }
}
```

```
void RotateInPlaceLand()
{
  // While our heading is too far off from the target
  while(abs(headingError) > ACC_HEAD_ERR)
  {
    if (headingError > 0)
    {
      // Rotate Right
      // Set the left motor Forward and the right motor to reverse.
      myHBridge.SetDirection(FORWARD, REVERSE);
      myHBridge.SetSpeed(255, 255);
    }
    else
    {
      // Rotate Left
      // Set the left motor Reverse and the right motor Forward.
      myHBridge.SetDirection(REVERSE, FORWARD);
      myHBridge.SetSpeed(255, 255);
    }

    currentHeading  = myBNO.GetHeading() - averageGPSError;
    headingError    = myHeadingControl.GetHeadingError(currentHeading, targetHeading);
  }
  // Set the motors back to straight
  myHBridge.SetDirection(FORWARD, FORWARD);
}
```

* → page 39 book 2.

Putting it All Together water


```
#include "src/BNO/BNO.h"
#include "src/GPS/gps.h"
#include "src/HeadingControl/HeadingControl.h"
#include "src/MotorControl/MotorControl.h"
#include "src/CircularArray/CircularArray.h"
#include "src/Controller/Controller.h"
#include <SD.h>

// Objects we will use
ESCControl      myMotors          = ESCControl();      // Motor Control
HeadingControl  myHeadingControl  = HeadingControl();  // Heading Controller
CircularArray   myArray           = CircularArray();   // Array to keep track of drift.
GPS             myGPS             = GPS();             // Our GPS Sensor
BNO             myBNO             = BNO();             // Our Acc., Gyro., Mag., sensor.
Controller      myController      = Controller();


// Data Structures we will use
GPSData         myGPSData;
GPSPair         myGPSPair;

// Important Variables
double distanceBetween;        // Distance between current lat/lon and destination lat/lon
double targetHeading;          // The heading we need to travel at to reach destination
double currentHeading;         // The heading we are traveling at
double headingError;           // Difference in target and current heading
double gpsError;               // difference in myGPSData.Heading and currentHeading
double averageGPSError = 0;    // Average difference in myGPSData.Heading and currentHeading
int    bnoResetPin     = 6;    // Pin to reset the BNO chip
int    SDChipSelect    = 10;
int    leftServoPin    = 2;
int    rightServoPin   = 7;
int    currentLeftSpeed  = 1500;
int    currentRightSpeed = 1500;

// Acceptable Heading Error - 2%
const double ACC_HEAD_ERR = 0.02;
//

// Omitting Path (dLats[], dLons[])
int waypointIndex    = 0;
int numWayPoints     = 99;
bool newWayPoint     = 1;
bool updatedAverage  = 0;
double totalDistance;
long startTime;
bool GotFirstStartTime = 0;
```

new

All the objects and variables the program will work with.

Putting it All Together

Dec 2023
Vishnu Verma

```
void setup()
{
  //Serial.begin(9600);
//  Serial.println("$GetObsessed!");
  pinMode(LED_BUILTIN, OUTPUT);

  // setupGPS()
  // Connect the GPS to Serial1.
  // This is RX0 and TX0 on the giga board.
  Serial1.begin(38400);
  if(!myGPS.setupGPS(Serial1))
  {
    // Freeze if setup fails
    digitalWrite(LED_BUILTIN, HIGH);
    //Serial.println("GPS Failed to begin.");
    while(1);
  }

  // setupBNO()
  // Connect the BNO to Serial2
  Serial2.begin(115200);
  if(!myBNO.setupBNO(&Serial2, bnoResetPin))
  {
    // Freeze if BNO failed
    digitalWrite(LED_BUILTIN, HIGH);
    //Serial.println("BNO Failed to start");
    while(1);
  }

  // setUPServos()
  // Set up the motors and turn them off
  myMotors.setup(leftServoPin, rightServoPin);

  // SetupSD()
  // Set the SPI ChipSelect pin as an output
  pinMode(SDChipSelect, OUTPUT);
  if(!SD.begin(SDChipSelect))
  {
   // If it fails to begin, freeze.
   // Check that the card is properly inserted.
   digitalWrite(LED_BUILTIN, HIGH);
   //Serial.println("SD Card failed to begin.");
   while(1);
  }

  File outputfile = SD.open("GPSData.txt", FILE_WRITE);
  outputfile.println("==================");
  outputfile.close();

  myController.begin();
  startTime = millis();
}
```

```
280 void loop()
281 {
282   myController.ReadController();
283   //Serial.println(myController.ProgramMode);
284
285    File outputfile = SD.open("GPSData.txt", FILE_WRITE);
286   outputfile.println("Inside main loop");
287   outputfile.close();
288   if (myController.ProgramMode == 0)
289   {
290     // ==========================
291      outputfile = SD.open("GPSData.txt", FILE_WRITE);
292     outputfile.println("Inside auto Loop");
293     outputfile.close();
294     // ==========================
295
296     // Set the "destination" to point 2 in the GPS pair.
297     myGPSPair.lat2_L = dLats[waypointIndex];
298     myGPSPair.lon2_L = dLons[waypointIndex];
299
300     // Update Our GPS data
301     myGPS.GetData(myGPSData);
302
303     // Set the "current" position to point 1 in the GPS pair.
304     myGPSPair.lat1_L = myGPSData.lattitude;
305     myGPSPair.lon1_L = myGPSData.longitude;
307     // Get the information we need to control the robot
308     distanceBetween = myHeadingControl.GetDistanceBetween(myGPSPair);
309     targetHeading   = myHeadingControl.GetTargetHeading(myGPSPair);
310     currentHeading  = myBNO.GetHeading() - averageGPSError;             // would need to flip here too.
311     headingError    = myHeadingControl.GetHeadingError(currentHeading, targetHeading);
312     gpsError        = currentHeading - (myGPSData.heading / 100000.0);  // could flip
313
314     // ==========================
315      outputfile = SD.open("GPSData.txt", FILE_WRITE);
316     outputfile.println("Before Log");
317     outputfile.close();
318     // ==========================
319
320     // Log the data we are about to use to make decisions
321     logToSd();
322     //printVars();
323
324     // If there are more than 4 sats. in view.
325     if (myGPSData.SIV > 4)
326     {
327
328     // ==========================
329      outputfile = SD.open("GPSData.txt", FILE_WRITE);
330     outputfile.println("Inside SIV");
331     outputfile.close();
332     // ==========================
333
334       // Start the Timer for the first time
335       if(GotFirstStartTime == 0)
336       {
337         startTime          = millis();
338         GotFirstStartTime = 1;
339       }
340
341       // ==========================
342       outputfile = SD.open("GPSData.txt", FILE_WRITE);
343       outputfile.println("Before Update the Array");
344       outputfile.close();
345       // ==========================
346
347       // Keep track of the error
348       myArray.Update(gpsError);
349
350       outputfile = SD.open("GPSData.txt", FILE_WRITE);
351       outputfile.println("After Update the Array");
352       outputfile.close();
```

Putting it All Together

Dec 2023
Wesley Castle

```
      // If we haven't been to this waypoint before.
      //     if (newWayPoint)
      //     {
      //       // Get the distance we are away from it
      //       totalDistance = distanceBetween;
      //       // Tell the program we have seen this point before.
      //       newWayPoint = 0;
      //     }

      // If our heading is too far off from the acceptable error
      if (abs(headingError) > ACC_HEAD_ERR)
      {
        updateMotors(1500, 1500); // Stop to think
        RotateInPlaceBoat();      // Fix Heading
      }

      // If we are within 3.0 m of our destination point.
      if(distanceBetween <= 2.0)
      {
        // Turn Motors Off and go to the next waypoint
        updateMotors(1500, 1500);
        waypointIndex++;

        // Tell the program we are going to a new way point
        // So we need to get a new total distance
        newWayPoint = 1;

        // Get the average for this leg and use it for the next one. Reset the average after.
        averageGPSError = myArray.GetAverage() + averageGPSError;
        myArray.ResetAverage();
        startTime = millis();

        // Tell the program we haven't updated the average for this leg yet.
        //       updatedAverage = 0;

        if (waypointIndex >= numWayPoints)
        {
          // Done with path.
          newWayPoint = 1;
          waypointIndex = 0;
          myController.ProgramMode = 1;
          myArray.ResetAverage();
          GotFirstStartTime = 0;
          averageGPSError = 0;
        }
      }
        //     else if ((distanceBetween <= totalDistance/2 and updatedAverage==0) and newWayPoint == 0)
        //     { // If we are halfway to a new point and we haven't updated our average yet
        //       // Then update the average
        //       averageGPSError = myArray.GetAverage() + averageGPSError;
        //       // Reset the average for the error
        //       myArray.ResetAverage();
        //       // Tell the program we have updated the average for this leg
        //       updatedAverage  = 1;
        //     }
      else if ((millis() - startTime) > 10000) // If we haven't updated GPS average for 10 seconds.
      {
        averageGPSError = myArray.GetAverage() + averageGPSError;
        myArray.ResetAverage();
        startTime = millis();
      }
      else
      {
        // Our heading is not too far off and we are not close enough to the point
        // Go Forwards
      // ==========================
       outputfile = SD.open("GPSData.txt", FILE_WRITE);
      outputfile.println("Set Motors To Forward");
      outputfile.close();
      // ============================
        updateMotors(1650, 1650);
      }
    }
```

Putting it all Together

Water

Dec 2023
Wagny
Czeku

```
    else
    {
      // 4 Sats are not in view. Standby...
      // ==========================
       outputfile = SD.open("GPSData.txt", FILE_WRITE);
      outputfile.println("4 Not In View");
      outputfile.close();
    // ==========================
      updateMotors(1500, 1500);
    }
  }
  else // Remote Control Mode
  {
    //Serial.print("LeftSpeed: ");
    //Serial.print(myController.leftMotorSpeedTarget);
    //Serial.print(" RightSpeed: ");
    //Serial.println(myController.rightMotorSpeedTarget);

    updateMotors(
      myController.leftMotorSpeedTarget,
      myController.rightMotorSpeedTarget);
  }
}
```

```
void RotateInPlaceBoat()
{
  // While our heading is too far off from the target
  while(abs(headingError) > ACC_HEAD_ERR)
  {
    // check the controller for updates
    myController.ReadController();
    if (myController.ProgramMode == 1)
    {
      // if we should be in remote control mode, leave this method.
      return;
    }

    if (headingError > 0)
    {
      // Rotate Right
      // Set the left motor Forward and the right motor to reverse.
      updateMotors(1700, 1280);
    }
    else
    {
      // Rotate Left
      // Set the left motor Reverse and the right motor Forward.
      updateMotors(1280, 1700);
    }

    currentHeading  = myBNO.GetHeading() - averageGPSError;
    headingError    = myHeadingControl.GetHeadingError(currentHeading, targetHeading);
  }
  // Set the motors back to straight
  updateMotors(1500, 1500);
}
```

Putting it all together


```
void updateMotors(int targetLeft, int targetRight)
{
    while (currentLeftSpeed != targetLeft || currentRightSpeed != targetRight)
    {
        if (currentLeftSpeed != targetLeft)
        {
            if(currentLeftSpeed < targetLeft)
            {
                currentLeftSpeed++;
            }
            else
            {
                currentLeftSpeed--;
            }
        }

        if (currentRightSpeed != targetRight)
        {
            if(currentRightSpeed < targetRight)
            {
                currentRightSpeed++;
            }
            else
            {
                currentRightSpeed--;
            }
        }
        myMotors.SetSpeed(currentLeftSpeed, currentRightSpeed);
    }
}
```

This method ensures that the motor speed doesn't change by more than 1 at a time. This helps reduce jutters.

RSS | Putting it All Together Water

Dec 2023
W Issy
code

These methods are used in both programs but only listed once for space reasons.

```
void printVars()
{
  Serial.print(myGPSData.lattitude);
  Serial.print(", ");
  Serial.print(myGPSData.longitude);
  Serial.print(", ");
  Serial.print(myGPSData.SIV);
  Serial.print(", ");
  Serial.print(myGPSData.heading);
  Serial.print(", ");
  Serial.print(myGPSData.speed);
  Serial.print(", ");
  Serial.print(distanceBetween);
  Serial.print(", ");
  Serial.print(targetHeading);
  Serial.print(", ");
  Serial.print(currentHeading);
  Serial.print(", ");
  Serial.print(headingError);
  Serial.print(", ");
  Serial.print(gpsError);
  Serial.print(", ");
  Serial.print(averageGPSError);
  Serial.println();
}
```

```
void logToSd()
{
  File outputfile = SD.open("GPSData.txt", FILE_WRITE);

  outputfile.print(myGPSData.lattitude);
  outputfile.print(", ");
  outputfile.print(myGPSData.longitude);
  outputfile.print(", ");
  outputfile.print(myGPSData.SIV);
  outputfile.print(", ");
  outputfile.print(myGPSData.heading);
  outputfile.print(", ");
  outputfile.print(myGPSData.speed);
  outputfile.print(", ");
  outputfile.print(distanceBetween);
  outputfile.print(", ");
  outputfile.print(targetHeading);
  outputfile.print(", ");
  outputfile.print(currentHeading);
  outputfile.print(", ");
  outputfile.print(headingError);
  outputfile.print(", ");
  outputfile.print(gpsError);
  outputfile.print(", ");
  outputfile.print(averageGPSError);
  outputfile.println();

  // Close the file
  outputfile.close();
}
```