Overview

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

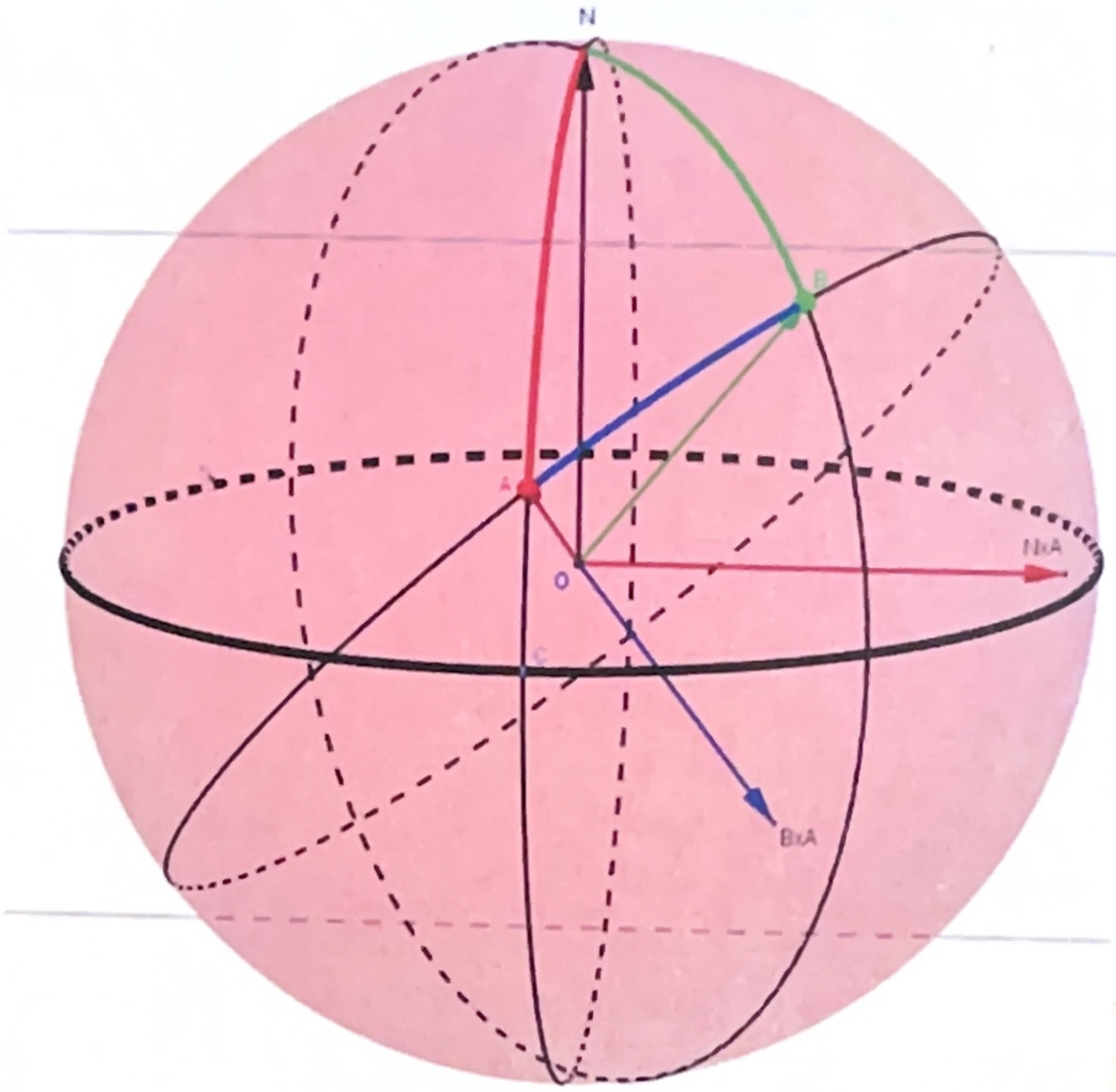
Thus

```
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/

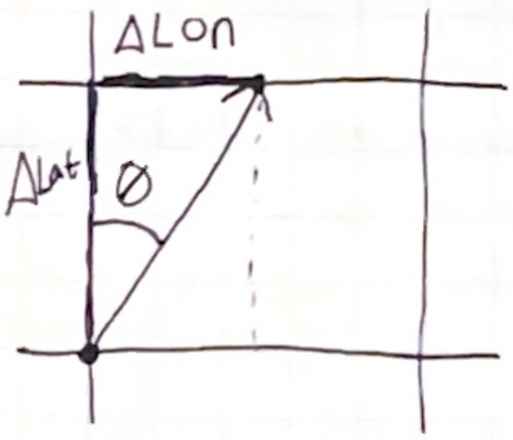


This shows a unit sphere. The heavy black circle is the equator; vector N is perpendicular to that (pointing to the north pole). Vectors A and B point toward the points A and B on the earth's surface. A blue arc is drawn between A and B along a great circle. The bearing from A to B is the angle at A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

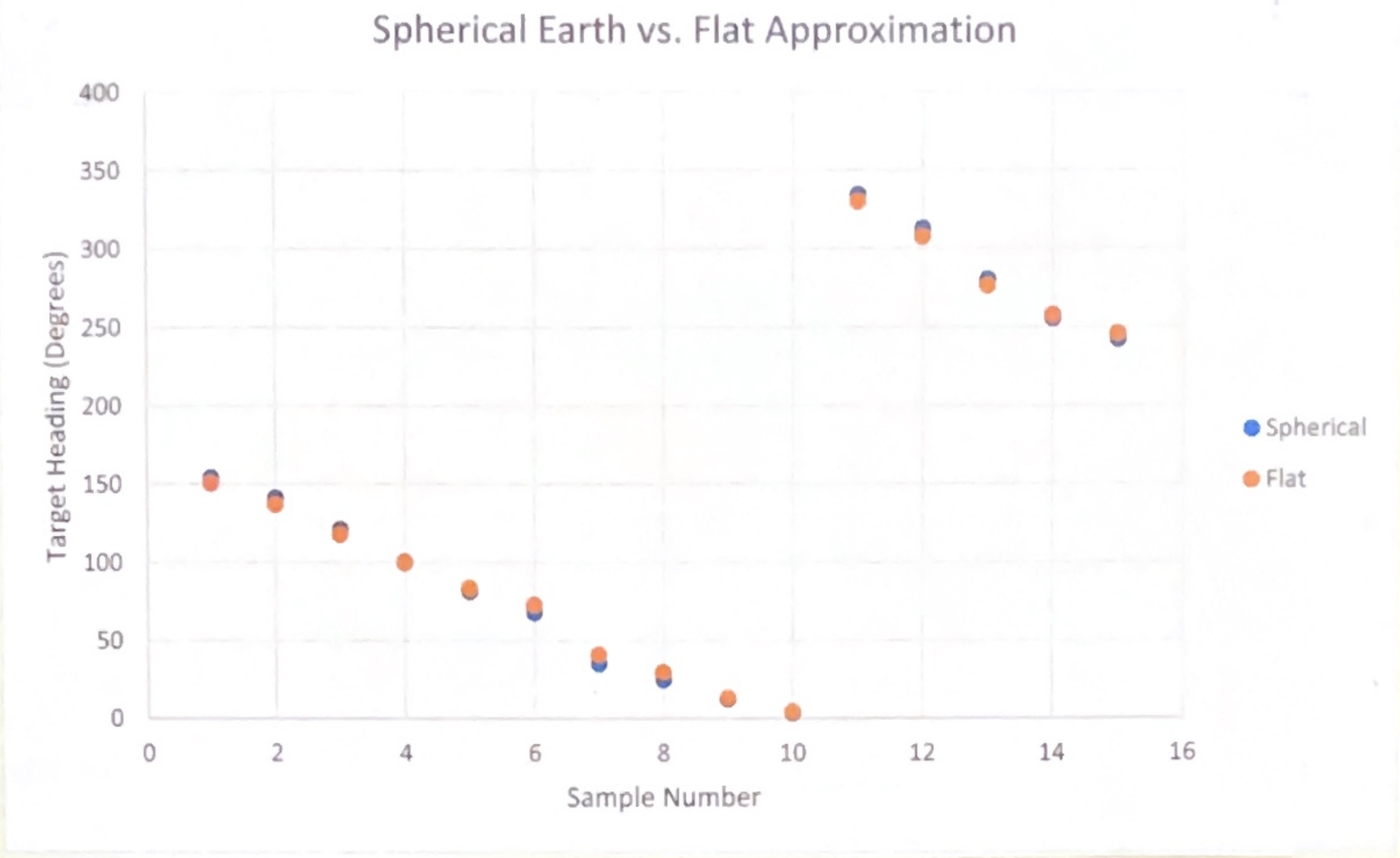
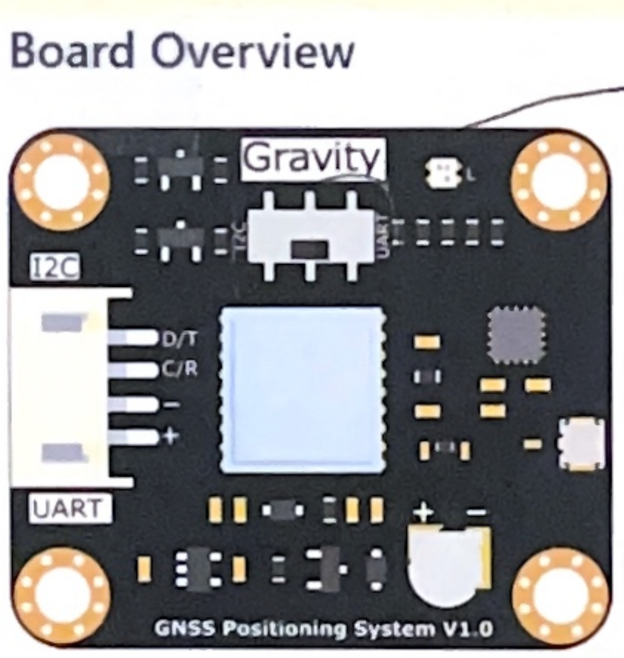


Figure 2
Comparing the spherical heading to the flat heading, Notice that since we were at small distances, the approximation isn't far off. However, if the distance is much bigger, the difference will also be bigger.



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

28 | DF Robot GNSS | 05/26/23 Wesley Creath

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

```
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

```
73 void loop()
74 {
75   // Get our current GPS coords
76   currentLat = gnss.getLat();
77   currentLon = gnss.getLon();
78   satsUsed = gnss.getNumSatUsed();
79
80   // Get the decimal degree's representation of the Lat and Lon.
81   cLat = currentLat.latitudeDegree;
82   cLon = currentLon.lonitudeDegree;
83
84   // Get the directions
85   // These directions are flipped in the library???
86   cLatDir = currentLon.lonDirection;
87   cLonDir = currentLat.latDirection;
88
89   // Correct the lattitude and longitude based on directions
90   if (cLatDir == 'S')
91   {
92     cLat = -cLat;
93   }
94   if (cLonDir == 'W')
95   {
96     cLon = -cLon;
97   }
98
99   // Get Target heading in degrees using the fancy equation
100  targetHeading = getTargetHeading(cLat, cLon, dLat, dLon) * RAD_TO_DEG;
101
102  // Get the heading using an approximation in degrees
103  flatEarthHeading = atan2((dLon-cLon)*DEG_TO_RAD, (dLat-cLat)*DEG_TO_RAD) * RAD_TO_DEG;
104
105  // Make sure headings are positive
106  if (targetHeading < 0) {targetHeading += 360;}
107  if (flatEarthHeading < 0) {flatEarthHeading += 360;}
108
109  printStuff();
110  delay(1000);
111 }
```