MBED Tracking Antenna System – A Critical Component to a UAS

User Manual and Technical Documentation

UCSD ECE Dept - Daniel Bedenko - dbedenko@ucsd.edu

Table of Contents

1. Introduction

Recently amateur grade Unmanned Aerial Systems(UAS) have been increasing in popularity. In general users want to fly their radio controlled airplanes as if they are in the cockpit by using a camera and a video transmitter, often times users wear video goggles, or a monitor on the ground to fly their aircraft. This particular Tracking Antenna was developed for such a purpose, but in particular, it was developed to help the UCSD AUVSI team for their annual sUAS (Student Unmanned Aerial System) Competition which takes place in Maryland. The UCSD team also uses an camera and video transmitter but their objective is to find and identify targets. More about the competition can be found on this website. http://65.210.16.57/studentcomp2010. Both of the above applications require an analog or digital video signal broadcasting from the aircraft. With a stationary application, this is a task easily accomplished with the use of a flat panel antenna, since they are light, cheap, and directional, and with line of sight work very well. For a highly dynamic mobile application, a directional antenna such as a flat panel antenna is problematic. The UCSD team uses a 14 dBi flat panel antenna with a 30 degree vertical and horizontal beam width, that means to get their video signal they would have to fly within 30 degrees of the antenna, which not practical. In the past the team has dedicated a freshman to point the antenna at the aircraft at all times, the developed Tracking Antenna System removes the need for a human gimbal and replaces it with a smart gimbal that receives position information of the airplane as a binary signal through the ground station of the autopilot, and then commands an azimuth and elevation servo to point the flat panel antenna at the aircraft. This system can be used with any autopilot system, or GPS only system as long as the communication protocol is met. This system was developed with the sUAS competition and UCSD team in mind, but the system is not specific to any ground station. Because magnetic declination changes with geo-position, this particular system can accurately be used only in San Diego, or Lexington Park Maryland, which is where the competition is held. Magnetic declination is the angle between magnetic north and true north, and it not only varies with location, but also with time. This system can be easily adapted for any location if the user so chooses by changing the code when the magnetic declination is known for a particular location and time.

2. Quick Operational Guide

The use of this system needs to be included at the end of the pre-flight system checklist, before taking off. For the purpose of this guide let's assume the user is at this stage, and is ready to take off. The user needs to complete the following steps for easy operation.

- Connect a Serial Cable to the system DB9 connector. Communications should already be established at the ground station based on the communication protocol.
- Connect the power cable from the battery to the system. LED1 should be blinking, LED2 should be blinking. LED1 is a measure of communication and protocol, and is set to blink for each correct communication string. LED3 or LED4 should be ON depending on where the user is located. LED3 for San Diego, and LED4 for Lexington Park, Maryland. The elevation servo should be pointing the system to 30 degrees.
- LED2 should be blinking, and this means that the system is not correctly aligned to True North. Loosen the yaw screw on the tripod and slowly rotate the system around the yaw axis around to where you think North should be, as soon as the user is within a degree of True North, LED2 will stop blinking, and the elevation servo will return to zero elevation. Tighten the yaw screw at the tripod. At this point, if the system is moved, the above steps need to be repeated.
- Connect the Coax cable to the Antenna from your video receiver, and zip-tie a meter or so of cable length to the leg of the tripod so that the rotational components are provided with adequate strain relief on the Antenna.
- Check reception of signal, complete the rest of the pre flight check list and you are ready to take off, as soon as there is some distance between the aircraft and the ground station the Tracking system will track the aircraft.

CAUTION: It is not recommended resetting the system while the aircraft is in the air. If the system is moved, you can power cycle it, and reset it to True North reference using the above instructions, but if this happens during flight, the antenna will go to the tracking position as soon as the reference north is set, at all times keep external objects away from the rotational parts of the system, this may cause harm to the user and the gear mechanism.

3. Theory of Operation

The very first process the system completes after establishing the commutation is it sets the magnetic declination, by retrieving the location of the ground station from the communication message. The detailed communication protocol is discussed on page 5. For example, in San Diego the magnetic declination is approximately +12, this is a known number that is stored in software. At this time a North reference point must be set and the system must be aligned to the True North axis by using a magnetometer. An indication to the user is made by blinking LED2 that the system is not set, a secondary visual indication is given by the elevation servo which is pointed 30 degrees up. As soon as the user points the gimbal to True north which is calculated from the magnetometer and the known declination, the user is prompted to stop moving when LED2 is turned off, and the elevation servo will point to zero elevation. Now the system enters it main operational loop. A new communication string is read in, and the latitude, longitude and altitude of the aircraft and the ground station is obtained. The latitude, longitude, and altitude (LLA) is converted to Earth Centered Earth Fixed Coordinates(ECEF) using [*Formula1*]. The two ECEF coordinate vectrors are subtracted, to obtain a single ECEF vector that points from the ground to the aircraft. This new ECEF vector is then converted to North East Down (NED) [*Formula2*] Reference Frame so that the azimuth and elevation angles can be obtained from the NED vector. Azimuth angle is obtained by taking the inverse tangent of North and East component of the NED vector [*Formula3*], and elevation angle is obtained using all three components using [*Formula4*].

Servos had to be calibrated to obtain a relationship between the pulse width and angle that pulse width produces. This was done by using a tilt meter, and sending servo pulses by a separate function which is included in the Code Section. A relationship was found between the pulse width and angle using the polyfit function in Matlab, this relationship is available for reference [*Formula5*]. This relationship depends on mechanical set up of the system, and will be different for any new system designed from this document. Once the mapping from angle to pulse width is known, it is implemented in software and the pulse width is sent to the elevation and azimuth servos.

Unfortunately, a true continuous rotation in azimuth was not possible due financial constrains. Instead a 360 degree servo rotation is implemented. If at any time the aircraft will be flown between 255 and 285 degrees from true north, the Azimuth rotation is flipped to by subtracting 180 degrees from that particular

position. The elevation flips and now operates in the 90 to 180 degree region to accommodate the Azimuth rotation. With this set up the aircraft can freely make circles around the ground station and the tracking system will follow. Generally it's not recommended to take off south of your ground station due to this rotational flip, but it's possible. More testing is necessary, to see the effect on signal strength during takeoff.

4. Electrical Specification

	Min	Nom	Max	Units
Input Voltage	4.5	5	6	V
Input Current	200	800	1200	mA
NiCd or NiMH Battery (1.2V/cell)	4	4	5	cells

5. LED Status Indications

LED1	Should blink, which indicates receipt of a correct communication string per change in state
LED2	Will blink while the system is not aligned to true north, is off during operation
LED3	Based on received message, magnetic declination is +12, San Diego, CA, otherwise off
LED4	Based on received message, magnetic declination is -10, Lexington Park, MD, otherwise off

6. Communication Protocol

The communication string is an array of bytes of size 26. There is a start byte [0xFF] and an end byte, [0xFE]. The data part of the communication string is the Latitude Longitude and Altitude in Decimal Degrees and Meters respectfully, of the aircraft and the ground station.

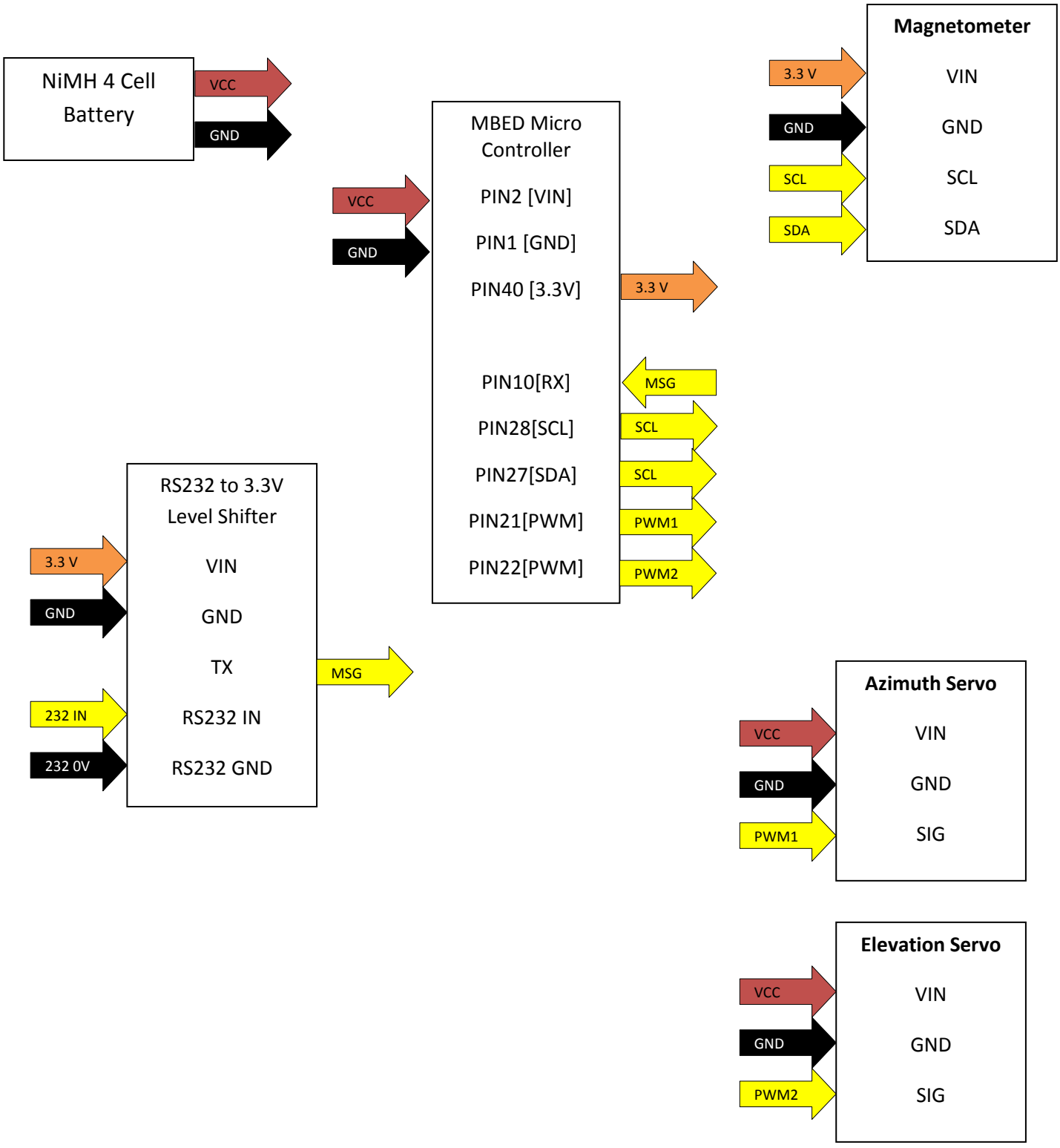
0xFF	Plane Lat	Plane Lon	Plane Alt	Ground Lat	Ground Lon	Ground Alt	0xFE

Each individual data field, is a 4 byte in a single precision floating format. As shown in the example below.

Plane Latitude in Decimal Degrees	Plane Latitude as a single precision floating point format			
39.995972	0xE0	0xFB	0x1F	0x42

By reversing the order, and combining the 4 bytes to a single 32 byte float, you can obtain the Plane Latitude in Decimal Degrees.

7. Wiring Diagram



8. Component List, Cost Overview

Component Name	Part Number	Supplier	Specifications	Cost
mbed microcontroller	NXP LPC 1768	mbed.org	Cortex-M3 Core running at 96MHz, with 512KB FLASH, 64KB RAM	*free*
RS232 Level Shifter	PRT-08780	Sparkfun.com	RS232 to (input)V out	9.95
Compass Module	SEN-07915	Sparkfun.com	1 degree repeatability, 1 to 20Hz selectable update rate 0.5 degree heading resolution	34.95
HS635HB Servo (azimuth)	HS-635HB	www.hitecrcd.com	Dual Ball Bearing, Karbonite Gears, Pulse Width Control 1500usec Neutral, 4.8-6.0 Volts	29.58
HS255BB Servo (elevation)	HS-255BB	www.hitecrcd.com	Pulse Width Control 1500usec Neutral, 4.8-6.0 Volts	22.83
Bottom Mount and Custom Potentiometer	SPG425A-360	servocity.com	Max rotation 400 degrees, .32sec/60 Deg	76.98
Breadboard	n/a	frys.com	Standard bread board	7.60
Metal Components Machining Fasteners	n/a n/a n/a	Ind. Meta Supply SIO Machine Shop Marshals Industrial	5051 Alum	14.76 23.25 5.44 7.60

Total Cost with Shipping **237.22**

9. Running C/C++ Code

File: tracking.c

```
#include <HMC6352.h>
#include <tracking.h>


//Magnetometer
HMC6352 compass(p28, p27);
//serial ports
Serial usb 232(USBTX, USBRX);      // tx, rx Debugging Port not used in actual Op
Serial groundStation(p9, p10);     // tx, rx
//leds
DigitalOut led1(LED1); DigitalOut led2(LED2); DigitalOut led3(LED3); DigitalOut led4(LED4);


int main() {
// wgs84 g for ground p for plane lat[deg] lon[deg] alt[m]
float compas, p_lat, p_lon, p_alt, g_lat, g_lon, g_alt;
// everything in meters ecef  NED is north east down ecef_xyz is the vector pointing
// from the ground to the plane in ecef. NED is the North East Down vector from
// ground to plane
float p_x, p_y, p_z, g_x, g_y, g_z, ecef_x, ecef_y, ecef_z, N, E, D;
// groundStation pos in Radians
float g_latR, g_lonR;
// Azimuth and Elevation Angles [Radians]
float Az, Ev;
// Servo commands
int Az_uS, Ev_uS;
bool  sanDiego;

//Set up serial settings Bits/S 8N1 is default
   usb_232.baud(57600);
   groundStation.baud(57600);
//HMC6352 Magnetometor Settings //Continuous mode, periodic set/reset, 20Hz rate.
   compass.setOpMode(HMC6352 CONTINUOUS, 1, 20);
// Set up Leds OFF
   led1 = led2 = led3 = led4 = OFF;

// Set up Servos for Azimuth and Elevation
 PwmOut Az servo(p21);
 Az_servo.period(0.020);
 PwmOut Ev_servo(p22);
 Ev_servo.period(0.020);

 // Set up Initial Servo Position
  Az = torads(90);
  Ev = torads(30);
  Az_uS = (int)(0.9195*Az*Az + 121.18*Az + 1308.43);
  Ev_uS = (int)(-1.223*Ev*Ev - 589.16*Ev + 2376.78);


  while (1) //set up while
    {


      //set up servo for turning on

      Az servo.pulsewidth us(Az uS);
      Ev_servo.pulsewidth_us(Ev_uS);


      wait(0.05);
      compas = (compass.sample() / 10.0);
      usb 232.printf("[%f] Heading \n", compas);
      led2 = !led2;                              //blink untill heading is set
```

```
      getlla(p_lat, p_lon, p_alt, g_lat, g_lon, g_alt);

      if (g_lon < -100){sanDiego = 1; led3 = ON;}//-100 long is about center of US
      else{sanDiego = 0; led4 = ON;}

      if (sanDiego == 1 && compas > 346.88 && compas < 348.88)
        {usb 232.printf("[%f] TRUE NORTH = 347.8 Magnetic North -- SD, CA\n", compas);
         led2 = OFF;  //reference found enter main loop
         Ev = torads(1);
         Ev_uS = (int)(-1.223*Ev*Ev - 589.16*Ev + 2376.78);
         Ev_servo.pulsewidth_us(Ev_uS);
         break;
        }
      if (sanDiego == 0 && compas < 11.98 && compas > 9.98)
        {usb 232.printf("[%f] TRUE NORTH = 10.98 Magnetic North -- Lexington Park, MD\n", compas);
         led2 = OFF;  //reference found enter main loop
         Ev = torads(1);
         Ev_uS = (int)(-1.223*Ev*Ev - 589.16*Ev + 2376.78);
         Ev servo.pulsewidth us(Ev uS);
         break;
        }
     }//end of set up while loop you have now pointed the antenna tracker at TRUE NORTH


//Start Main loop
 while(1){

  /* This main loop gets plane coordinates from the serial port
   * converts it to a NED vector pointing from the ground statio
   * to the airplane and then sends servos to point at it
   */

  // get ground and plane coordinates
  getlla(p_lat, p_lon, p_alt, g_lat, g_lon, g_alt);

  //convert airplane to ecef
  lla2ecef(p_lat, p_lon, p_alt, p_x, p_y, p_z);

  //convert ground to ecef
  lla2ecef(g_lat, g_lon, g_alt, g_x, g_y, g_z);

  //subtract plane from ground to get the difference vector
  ecef_x = p_x - g_x;
  ecef y = p y - g y;
  ecef_z = p_z - g_z;

  g_latR= torads(g_lat);
  g_lonR= torads(g_lon);

  /* translation ecef xyz to north east down
   *
   *  TE2L = [ -sin(glat)*cos(glon)  -sin(glat)*sin(glon)  cos(glat)
   *           -sin(glon)             cos(glon)               0
   *           -cos(glat)*cos(glon)  -cos(glat)*sin(glon) -sin(glat)  ]
   *  NED = TE2L*ecef
   */

  N = -sin(g_latR)*cos(g_lonR)*ecef_x + -sin(g_latR)*sin(g_lonR)*ecef_y + cos(g_latR)*ecef_z;
  E = -sin(g_lonR)*ecef_x + cos(g_lonR)*ecef_y;
  D = -cos(g_latR)*cos(g_lonR)*ecef_x  -cos(g_latR)*sin(g_lonR)*ecef_y + -sin(g_latR)*ecef_z;
  D = (-1)*D; //we are on the ground looking up! :)

  Az = atan2(N,E);
  Ev = abs(atan2(D, sqrt(N*N + E*E))); // this will always be positive

 // if plane is on the ground next to ground station point north
 if(g lat == p lat && g lon == p lon)
  {Ev = 0; Az = PI/2;}

  //Aling Az is from 0 to 2*pi - no negative numbers
```

```
  if (Az < 0)
     Az = Az + 2*PI;

  //from here on out it depends on the how the servo pulse width relates to
  //angle her my azimuth servo is defined from 270 -> -90 deg 1900 - 1120 uS
  //and the elevation Servo is defines from 0 - 180 deg  0500 - 2400 uS


  //blackout region where the azimuth will swing back to Az-PI
  //And Ev = pi - Ev
    if (Az > 17*PI/12 && Az < 19*PI/12)
        {Az = Az - PI;
         Ev = PI - Ev;}

  // 270 -> -90
     if (Az > 3*PI/2 && Az < 2*PI)
     Az = Az - 2*PI;

  //now get the uS .. the values below are from polyfit matlab
  //from your servo calibrations to the second degree

  Az_uS = (int)(0.9195*Az*Az + 121.18*Az + 1308.43);
  Ev_uS = (int)(-1.223*Ev*Ev - 589.16*Ev + 2376.78);

  // Send Servo Commands
   Az servo.pulsewidth us(Az uS);
   Ev_servo.pulsewidth_us(Ev_uS);


 // usb 232.printf("[%2.2f Az]  [%i Az uS]  [%2.2f Ev] [%i Az uS] \n", todegs(Az), Az uS, todegs(Ev), Ev uS );

 //usb 232.printf("%2.2f Azimuth %2.2f  North %2.2f East\n", todegs(Az), N, E );
 // usb_232.printf("[%f   %f    %f ] Plane \n", p_lat, p_lon, p_alt );
 // usb_232.printf("[%f   %f    %f ] Plane ecef \n ", p_x, p_y, p_z );
 // usb_232.printf("[%f   %f    %f ] Ground \n", g_lat, g_lon, g_alt);
 // usb 232.printf("[%f   %f    %f ] Ground ecef \n ", g x, g y, g z );
}

} // end main
void  lla2ecef(float lat, float lon, float alt, float& x, float& y, float& z)
{
  // WGS84 ellipsoid constants:
   float a = 6378137; //earth semi-major axis
   float esqrd = 0.006694379990141; //Eccentricity
   float N;
   //calculation
    lat = torads(lat);
    lon = torads(lon);
     //The length of normal (N) to the ellipsoid
    N = a / sqrt(1 - esqrd * sin( lat ) * sin( lat ) ) ;
    x = (N+alt) * cos(lat) * cos(lon);
    y = (N+alt) * cos(lat) * sin(lon);
    z = ((1-esqrd) * N + alt) * sin(lat);

}
float torads(float deg)
{
    return (deg*PI) / 180 ; //in rads
}
float todegs(float rads)
{
    return (rads*180) / PI ; //in degs
}
void servo_calib(void)
{
 PwmOut servo(p22);
 servo.period(0.020);
 char pwm_uS[5];
 int pwm uS i;
    while(1)
```

```
            {
            usb_232.printf(" \n Enter PWM in uS [0600-2400] : ");
            usb_232.scanf("%s", pwm_uS);
            pwm_uS_i = atof ( pwm_uS );
            servo.pulsewidth_us(pwm_uS_i);
            usb_232.printf("\n You Entered: %i ", pwm_uS_i);
            }
}
void  getlla(float& p_lat, float& p_lon, float& p_alt, float& g_lat, float& g_lon, float& g_alt)
{
     uchar msg[26];
     do{
        while( (msg[0] = groundStation.getc()) != 0xff); // wait for the start of communication string
            for(int i=1; i<26; i++)  {
            msg[i] = groundStation.getc(); //fill message string
            }
       }while(msg[25] != 0xfe); // make sure the last character is 0xFE otherwise re read
             led1 = !led1; //blink :)
    //airplane
    p_lat = tofloat(msg[4], msg[3], msg[2], msg[1]);
    p_lon = tofloat(msg[8], msg[7], msg[6], msg[5]);
    p_alt = tofloat(msg[12], msg[11], msg[10], msg[9]);
    //ground
    g_lat = tofloat(msg[16], msg[15], msg[14], msg[13]);
    g_lon = tofloat(msg[20], msg[19], msg[18], msg[17]);
    g_alt = tofloat(msg[24], msg[23], msg[22], msg[21]);

}
float tofloat(uchar b0, uchar b1, uchar b2, uchar b3)
{
    uchar b[] = {b3, b2, b1, b0};
    float *fp = (float *)b;
    return *fp;
}
```

File: tracking.h

```
#define PI 3.14159274
#define ON  1
#define OFF 0
typedef unsigned char uchar;


float torads(float deg);
float todegs(float rad);

//Gets string from groundStation and converts to Lat Lon Alt
void  getlla(float& p_lat, float& p_lon, float& p_alt, float& g_lat, float& g_lon, float& g_alot);

//converts lat lon alt to ECEF earth centered earth fixed coordinates
void  lla2ecef(float lat, float lon, float alt, float& x, float& y, float& z);

//converts 4 bytes to float
float tofloat(uchar b0, uchar b1, uchar b2, uchar b3);

//this is a testing function it allows me to input a puse width in uS and output it to the servo(s)
// Allows me to get y = Mx + B.. it's not used in the main program but is usefull
void  servo calib();
```

10. Formula and Relation Reference

1. Latitude Longitude Altitude to Earth Centered Earth Fixed Coordinate System
   a) Define the flatness (f) and eccentricity (e) of the ellipsoid as follows [MAE142 notes]

$$f=\frac{a}{a-b} \qquad e^2=\frac{1}{f}\left(2-\frac{1}{f}\right)$$

$a=semimajor\ axis$
$b=semiminor\ axis$
$f=flatness$
$e=eccentricity$

Semi-major axis = 6378137 m
Semi-minor axis = 6356752.3142 m
Flattening = 298.257223563
Eccentricity = 0.0818191908426

   b) The length of normal (N) to the ellipsoid, extending from the surface of the ellipsoid to its intersection with the z-axis of the ECEF is:

$$N(\varphi)=\frac{a}{\sqrt{1-e^2\sin^2\varphi}} \qquad \varphi=latitude$$

   c) Covert

$$X=(N+h)\cos(\varphi)\cos(\lambda) \qquad Z=[N(1-e^2)+h]\sin(\varphi)$$
$$Y=(N+h)\cos(\varphi)\sin(\lambda)$$

$\varphi=latitute$
$\lambda=longitude$
$h=altitude\ above\ ellipsoid$

2. Conversion to North East Down Coordinate System from ECEF [MAE142 notes]

$$\begin{bmatrix}N\\E\\D\end{bmatrix}=T_{E2L}\begin{bmatrix}X\\Y\\Z\end{bmatrix} \qquad T_{E2L}=\begin{bmatrix}-\sin\varphi\cos\lambda & -\sin\varphi\sin\lambda & \cos\varphi\\ -\sin\lambda & \cos\lambda & 0\\ -\cos\varphi\cos\lambda & -\cos\varphi\sin\lambda & -\sin\varphi\end{bmatrix}$$

$\varphi=latitute\ of\ NED\ frame\ origin$
$\lambda=longitude\ NED\ frame\ origin$

3. Azimuth Angle Calculation

$$Az=\tan^{-1}\left(\frac{N}{E}\right)\ rads$$

4. Elevation Angle Calculation

$$Ev=\tan^{-1}\left(\frac{-D}{\sqrt{N^2+E^2}}\right)rads$$

5. Function that maps Angle to Pulse Width, constants obtained with Matlab's polyfit function.

$$Az_{uS}=.9195Az^2+121.18Az+1308\ uS$$
$$Ev_{uS}=-1.223Ev^2-589.16Ev+2377\ uS$$