Attitude Determination of a Land Vehicle using Inertial Measurement Units

By:

Brian Bleeker- bbleeker_02@yahoo.com
Rob MacMillan- cubsfan789@yahoo.com

Advised by:

Dr. In Soo Ahn

Date:

December 12, 2003

Table of Contents

1. Project Summary

To determine the attitude of a land vehicle with respect to navigation coordinates, angular rates and translational accelerations are measured using an inertial measurement unit. This attitude determination system can be used to supplement a global positioning system in time when there is no line of sight to GPS satellites. Similar systems can be implemented to record the movement of the vehicle to prevent it from leaving a set course or from moving into an unstable or dangerous environment. These systems may help a driver by triggering increased traction control and air bag deployment as the accelerations and attitude angles change while driving. The block diagram of this system can be seen in Fig. 1.1.1.

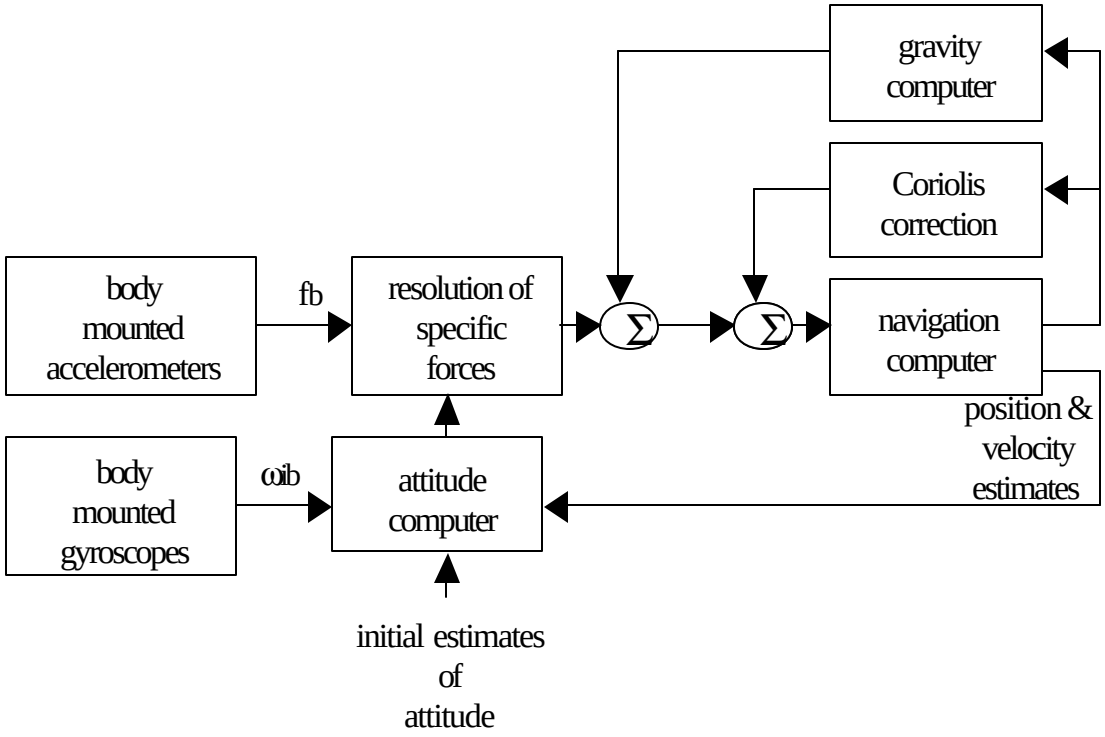

Figure 1.1.1 Overall Block Diagram of the system

2. Project Description

2.1. Inertial Measurement Unit

Movement of the land vehicle causes the output of the three accelerometers and three gyroscopes within the inertial measurement unit to output the translational accelerations and angular rates with respect to the body axis. The acceleration of gravity will be shown as acceleration on one or more of the acceleration outputs depending upon how the inertial measurement unit (IMU) is attached to the land vehicle (see figure 2.1.1).

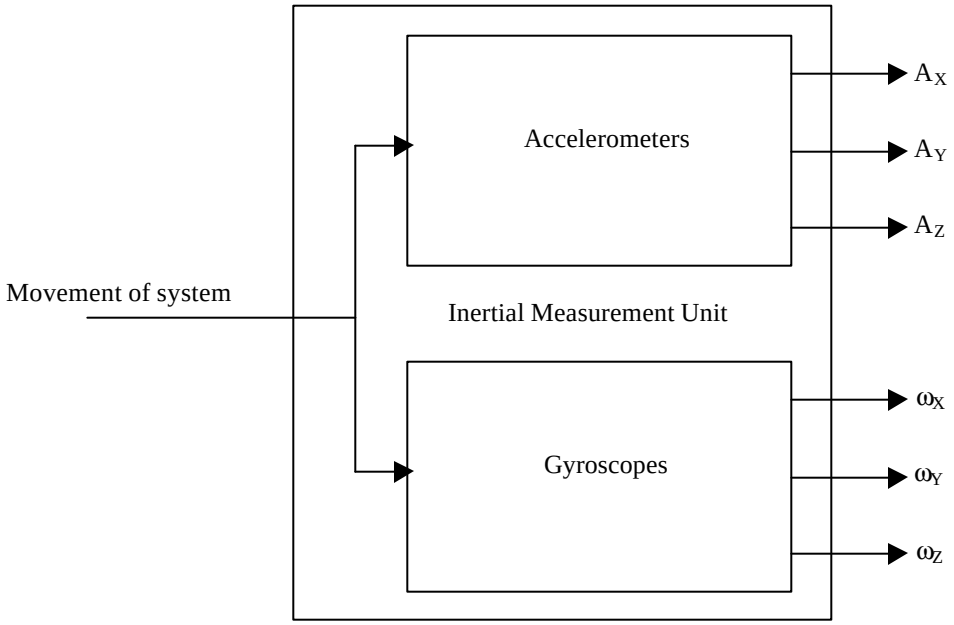

Figure 2.1.1 Inertial Measurement Unit

2.2. Attitude Computer

In order to resolve each of the translational acceleration terms as well as the angular rates, a series of filters must be used to eliminate noise present on the outputs of the Inertial Measurement Unit. These filters will be implemented using Matlab. Disturbances that need to be filtered are output noise, accelerometer bias, angular rate bias, and acceleration with respect to gravity. After the IMU outputs are filtered, the acceleration outputs must be integrated to get velocity with respect to the body frame. Velocity can be integrated to get the position of the vehicle. The integration must also include the current attitude information, known as Euler angles, in order to yield the correct velocity and position.

Euler angles are calculated by a series of equations that can be seen in Fig. 2.2.1. The turn rate of the earth with respect to the navigation frame and the turn rate of the navigation frame with respect to the earth in the navigation frame are added together and multiplied by a directional cosine matrix to relate it to the body frame. In order to obtain the updated directional cosine matrix, the previous matrix is added to the change in directional cosine matrix multiplied by the skew matrix of the turn rate of the body with respect to the navigation frame in the body frame. From this new directional cosine matrix value, the Euler angles, or updated platform angles, can be calculated. When the system is first started, initial attitude angles must be entered in order to obtain the initial directional cosine matrix.

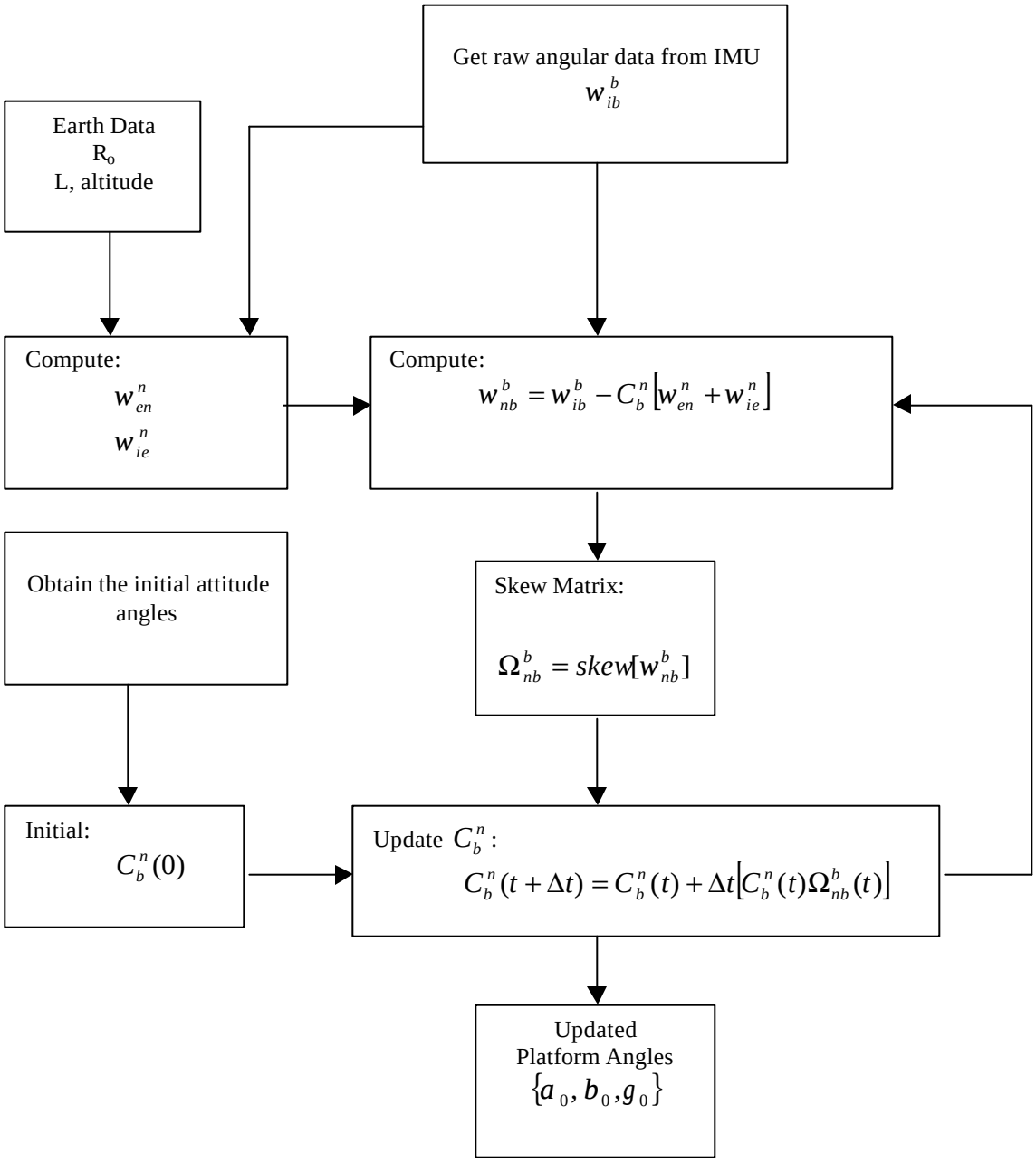

Figure 2.2.1 Attitude Computer

2.3. Navigation Computer

In order to properly compute the attitude of the land vehicle, the attitude computer must be updated continuously as the vehicle moves. Also, terms such as gravity and the coriolis force, must be factored in to properly give velocity and position estimates in the local coordinate system. With these corrections the movement of the land vehicle can be

fed back to the attitude computer in the same north, east, down coordinate system of the local navigation system and the attitude computed. This navigation computer will be implemented in Matlab. For the block diagram of the Navigation Computer see Fig. 2.3.1.

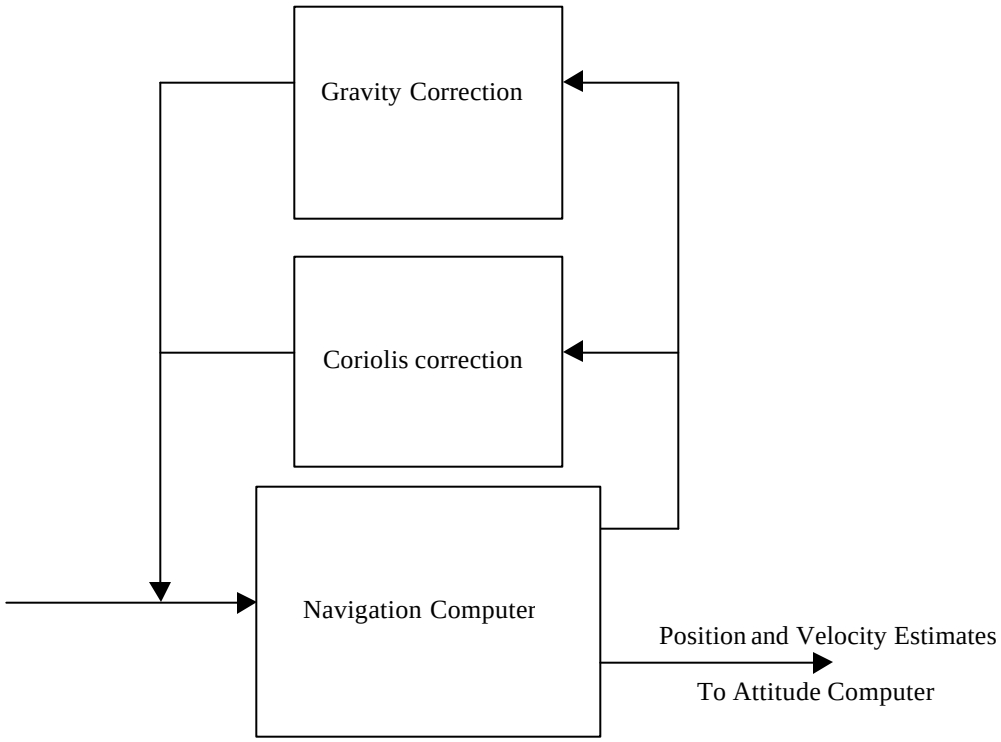

Figure 2.3.1 Navigation Computer

2.4. Completed Work

2.4.1. Product Research

A thorough Internet search was done. Most of the companies that sell IMUs make them for military grade operations. This means that they are extremely expensive for Bradley's limited electrical engineering budget. Two companies that sell IMUs around $3,000 were found.
Crossbow has two products near this price range. The IMU300CC is priced at $2,995.00 and the IMU400CC is priced at $3,995.00. The IMU400CC has a significantly better angular rate bias than the IMU300CC. The crossbow company also offers a 10% educational discount.
BEI Systron Donner also has an IMU product in the price range. The MotionPakII is priced at $2,500, but it's angular rate bias is more than twice the IMU300CC from crossbow. BEI doesn't offer any educational discount and the data sheets for their IMUs were pretty poor.

These products are significantly less expensive than military grade products because they are manufactured with silicon chips. These chips are prone to increased biases and random walk noise. For the purposes of this project the IMU300CC was purchsed.

3.1.1. Practice Matlab Code

To get familiar with writing Matlab functions, sample data taken from a GPS receiver was analyzed. The data file was read into Matlab. This data was in earth-centered earth fixed coordinates. This means that the location of the receiver was measured with respect to the center of the earth. The goal of the exercise was to convert the ECEF coordinates to local navigation coordinates. Also, another function was written to convert the local navigation coordinates back to the ECEF coordinates. The Matlab code written for this exercise and the results from the code are shown in Appendix A.

3.1.2. Attitude Computer Matlab Code

The attitude computer of the attitude determination system is written and functional. Each block of the diagram is written as individual functions. This will help in writing other programs so that each function can be called accordingly. The following list of code has been written and can be seen in Appendix B.

- The turn rate of the navigation frame with respect to the earth in the navigation frame.
- The turn rate of the earth with respect to the inertial frame in the navigation frame.
- The turn rate of the body frame with respect to the navigation frame in the body frame.
- Skew Matrix for a 1 by 3 matrix.
- Update the directional cosine matrix of the body frame in the navigation frame.
- Initialize the directional cosine matrix.
- Create the current platform angles.

With all these functions written an overall main file was written to call the functions of the attitude computer. This function will also control the transfer of data between the resololution of specific forces, navigation computer and attitude computer.

2.5. Experimental Testing

To simulate the movements of a land-based vehicle, a tested platform will be constructed. This platform will have wheels to allow the user to push it in the desired direction to test the IMU accelerometers. The testing platform will also have a rotating axis that will allow for testing the IMU gyroscopes. There will be a laptop computer on the testing

platform to collect the data. Also, a battery pack will be onboard to power the IMU. A layout for the testing platform can be seen in Fig. 2.4.1 and 2.4.2.

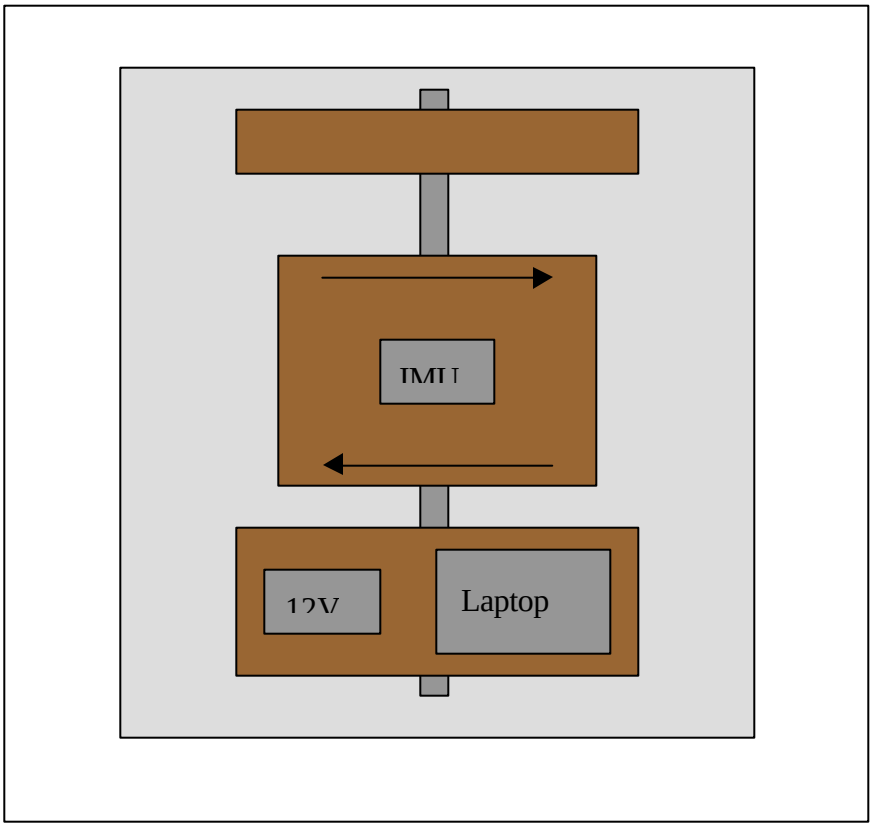

Figure 2.4.1 Top View of Testing Platform

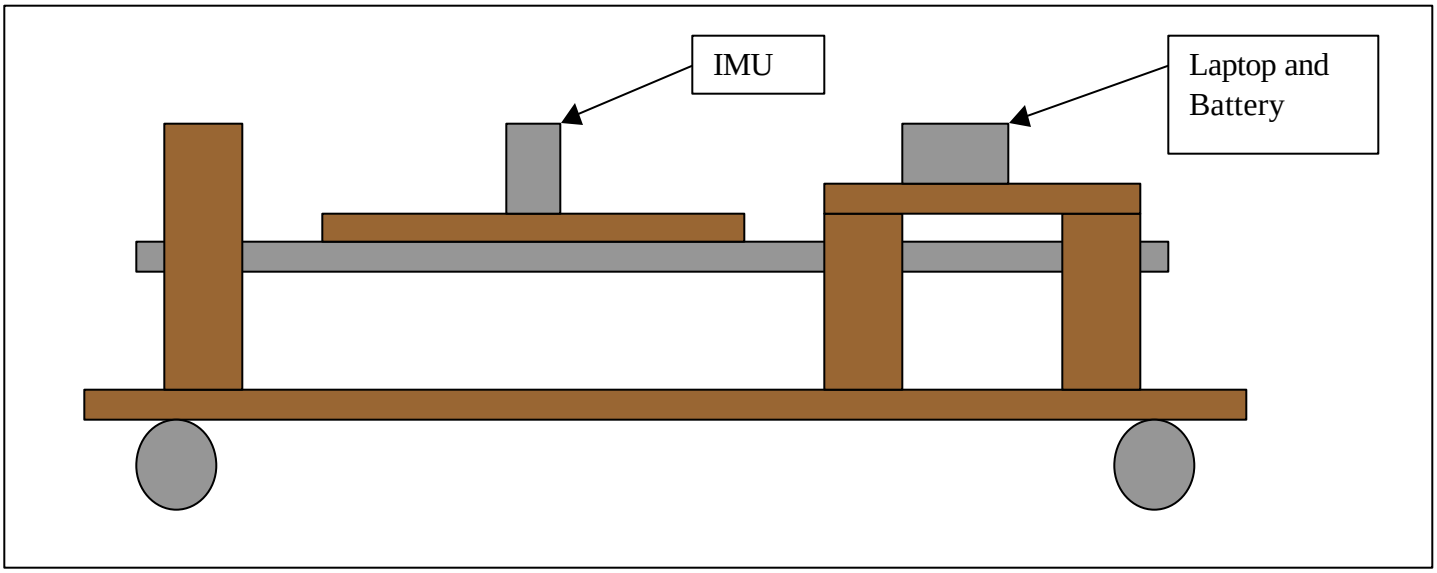

Figure 2.4.2 Side View of Testing Platform

3. Schedule

Winter Interim

- Build IMU testing platform.
- Continue developing Matlab code for the attitude computer.
- Update website.

January

01/19/03-01/26/03

- Finish the IMU testing Platform.
- Finalize individual attitude computer functions.

01/27/03-02/02/03

- Collect data using the IMU testing platform.
- Update website.

February

02/03/03-02/10/03

- Continue collecting data under various conditions.
- Interface data file to Matlab code.

02/11/03-02/18/03

- Analyze the drift from data where the system is motionless.

02/19/03-02/26/03

- Continue analysis of bias drift and try to determine how to eliminate this error.
- Update website.

March

02/27/03-03/12/03

- Eliminate bias error from collected data using Matlab.

03/13/03-03/20/03

- **Spring Break**

03/21/03-03/28/03

- Complete overall attitude computer software.
- Update website.

April

03/29/03-04/04/03

- Determine accelerations, velocities, and position using collected data from the IMU and analyzing it using Matlab code.

04/05/03-04/12/03

- Test overall system with movement.
- Analyze the results with completed code and make final adjustments.

04/13/03

- Finalize lab notebooks and website.
- Prepare for presentation.
- Write final written report.

4. Standards/Patents

4.1. Standards

A standard search was done on the Internet. Places to purchase standards were found, but there were no free standards listings found. As of right now, standards have not been purchased for this project.

4.2. Patents

Using a patent research tool linked to the Bradley University Cullom Davis Library website the following patents have similarities to this project.

- 6,463,366 Attitude determination and alignment using electro-optical sensors and global navigation satellites
- 6,417,802 Integrated inertial/GPS navigation system
- 6,292,750 Vehicle positioning method and system thereof

All of the above patents have to do with a system integrating a GPS receiver and an inertial measurement unit. For these systems the GPS acts to initialize and keep the IMU within error specs by resetting the system. This project does not depend on the error compensation abilities of the GPS systems. As of right now, there are no patents directly applicable to this project.

5. Equipment List

1 – Inertial Measurement Unit – to be ordered
1 – Laptop computer – Available through the electrical engineering department
1 – 12V battery - Available through the electrical engineering department

APPENDIX A – Practice Matlab Code

The following is code for Matlab converting the position of a vehicle in earth-centered earth fixed coordinates to Latitude Longitude and altitude.
Figure A.1 shows the position in ECEF coordinates
Figure A.2 shows the position in latitude, longitude and altitude

From the website http://www.geocode.com/eagle.html it was found that the approximate address where the data was take was 1701 E. Empire St. in Bloomington Il.

ECEF To NED

```
function [lat, lon, alt] = ecef2ned(gtime, ecefpos)

% ECEF2NED  Convert ecef coordinates to NED coordinates and diplay.
%
% [latitude, longitude, altitude] = ecef2ned(gtime, ECEF position)
%  ECEF Position Array = [x, y, z]
%
%  Example:
%     array.x
%     array.y
%     array.z
%  *** The array parameters must have *.x, *.y, and *.z.
%
% Enter the time, and the x,y,z positions in ecef coordinates.
% This function will display two figures:
%           1:  x,y,z position in ecef coordinates and
%           2:  latitude, longitude, and altitude
% A negative longitude is the Western hemispere.
% A negative latitude is in the Southern hemisphere.
%
% Written  By: Brian Bleeker
%              Rob MacMillan

b = size(gtime);
gtime = gtime(:,1) - gtime(1,1);

figure(1), subplot(311), plot(gtime, ecefpos.x), grid
title('ECEF X Position')

subplot(312), plot(gtime, ecefpos.y), grid
title('ECEF Y Position')

subplot(313), plot(gtime, ecefpos.z), grid
title('ECEF Z Position')

a = 6378137.0;                                    %semi-major axis
(equatorial) radius
```

```
b = 6356752.3142;                                   %semi-minor axis
(polar) radius
f = (a-b)/a;
e = sqrt(f*(2-f));                                  %eccentricity of
ellipsoid
len = length(ecefpos.x);                            %get length of data

for i = 1:len
    lon(i) = atan2(ecefpos.y(i), ecefpos.x(i));     %long = atan(y,x) -
direct
    lon(i) = lon(i)*180/pi;                         %convert to degrees

    h = 0;                                          %initialize
    N = a;
    flag = 0;
    j = 0;
    p = sqrt(ecefpos.x(i)^2 + ecefpos.y(i)^2);

    sinlat = ecefpos.z(i)/(N*(1-e^2)+h);            %First iteration
    lat(i) = atan((ecefpos.z(i)+e^2*N*sinlat)/p);
    N = a/(sqrt(1 - (e^2)*(sinlat^2)));
    prevalt = (p/cos(lat(i)))-N;
    prevlat = lat(i)*180/pi;

    while (flag < 2)                                %do at least 100
iterations
        flag = 0;
        sinlat = ecefpos.z(i)/(N*(1-e^2)+h);
        lat(i) = atan((ecefpos.z(i)+e^2*N*sinlat)/p);
        N = a/(sqrt(1 - (e^2)*(sinlat^2)));
        alt(i) = (p/cos(lat(i)))-N;
        lat(i) = lat(i)*180/pi;
        if abs(prevalt-alt(i)) < .00000001
            flag = 1;
        end
        if abs(prevlat-lat(i)) < .00000001
            flag = flag + 1;
        end
        j = j+1;
        if j == 100
            flag = 2;
        end
        prevalt = alt(i);
        prevlat = lat(i);
    end
end

figure(2), subplot(311), plot(gtime, lon), grid
title('NED Longitude')

subplot(312), plot(gtime, lat), grid
title('NED Latitude')

subplot(313), plot(gtime, alt), grid
title('NED Altitude')
```

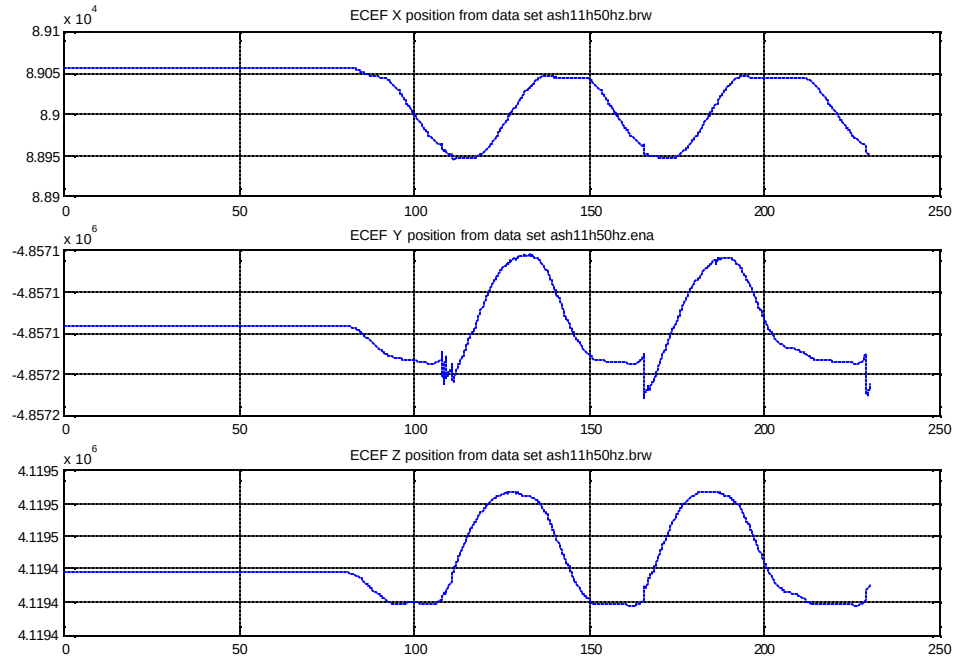

Figure A.1 Position in ECEF Coordinates

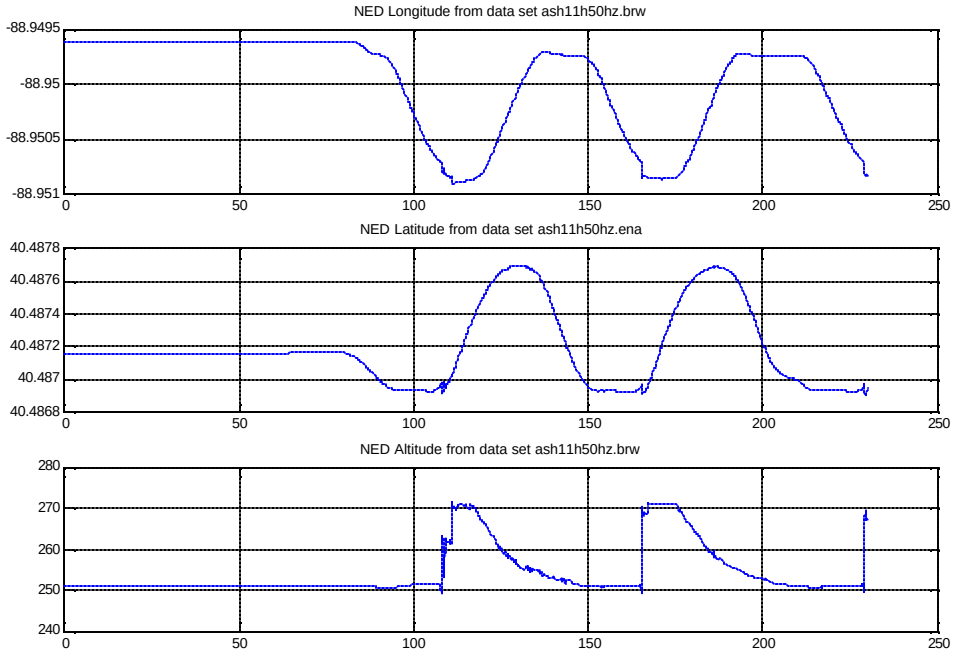

Figure A.2 position in latitude, longitude and altitude

NED To ECEF

The following code converts North, East, Down coordinates to Earth Centered Earth Fixed (ECEF) coordinates.

```
function [ecef_pos] = ned2ecef2(ned)
% NED2ECEF2 Convert NED coordinates to ECEF coordinates and diplay.
%
% [ECEF position] = ned2ecef2(ned position)
%  NED Position Array [n,e,d]
%
%  Example:
%     array.n
%     array.e
%     array.d
%  *** The array parameters must have *.n, *.e, and *.d
%
% The NED positions are latitude longitude and altitude in decimal
degrees.
% A negative longitude is the Western hemispere.
% A negative latitude is in the Southern hemisphere.
%
% Written By: Brian Bleeker
%             Rob MacMillan

%a = earth_shape; %  call earth_shape to get earth data
a = 6378137 ; %a(1);
b = 6356752.3142; %a(2);
lat=ned.n*pi/180;
lon=ned.e*pi/180;
f = (a-b)/a;
e = sqrt(f*(2-f));
N = a /(sqrt(1-e^2*(sin(lat))^2));


%lat = ned.pos(1)/b + ned.geo_ref(1); % compute the current latitude
coslat = cos(lat);
sinlat = sin(lat);
coslon = cos(lon);
sinlon = sin(lon);

x = (N + ned.d)*coslat*coslon;

y = (N + ned.d)*coslat*sinlon;  % compute the current longitude

z = (N*(1 - e^2)+ ned.d)*sinlat;

%r0 = r0 + ned.geo_ref(3)*coslat;

ecef_pos.x = x;% assign positions
ecef_pos.y = y;
ecef_pos.z = z;
```

The following sample data takes the NED coordinates for Dr. Ahn's house and converts it into ECEF coordinates. It then converts it back. From this we are able to find the error in the strictly mathematical model for processing the data.

Sample Data:

```
 » ned
ned =
    n: 40.752169
    e: -89.672332
    d: 250

» ecef=ned2ecef2(ned)
ecef =
    x: 27672.3433890974
    y: -4838712.35630367
    z: 4141776.28953698

» [lat, lon, alt] = ecef2ned(1, ecef)
lat =
         40.752176473724
lon =
             -89.672332
alt =
        249.999985948205
```

The error for the longitude calculation is very close to 0 as it is strictly an inverse tangent operation. When converting the latitude, the error is on the order of .00002%, which is caused by the iteration process. The error at this altitude is on the order of .00000006%. As coordinates and altitude change the error will increase however we calculated that the error at 2400 meters is approximately 2 millimeters and the error 12000 meters is approximately 33 centimeters. We feel that this is reasonable error as 12000 meters is approximately the ceiling for commercial air travel (35000 ft.).

APPENDIX B – Attitude Computer Matlab Code

Earth Parameters

```
function [earth]=earth_param()

   % EARTH_PARAM  Returns the constant parameters of the earth for use
   in different functions.
   %
   % [earth] = earth_param()
   %
   % Returns:
   %         a - earth equitorial radius
   %         b - earth polor radius
   %         e - eccentricity of elipsoid
   %         f - (a-b)/a
   %         o - omega - earth turn rate
   %
   % Written By: Brian Bleeker
   %             Rob MacMillan
   earth.a = 6378137.0;
   earth.b = 6356752.3142;
   earth.f = (earth.a-earth.b)/earth.a;
   earth.e = sqrt(earth.f*(2-earth.f));
   earth.o = 7.292115E-5;
```

Turn Rate of the Earth With Respect to the Inertial Frame in the Navigation Frame

```
   function [win] = trEARTH_IF_NED(lat)

   % TREARTH_IF_LGF  Find the turn rate of the earth with respect to
   the inertial
   %                 frame in the local NED frame.
   %
   % [win] = trEARTH_IF_NED(Latitude)
   %
   % Latitude is in decimal degrees.
   %
   % Written By: Brian Bleeker
   %             Rob MacMillan

   earth=earth_param;

   win(1) = earth.o * cos(lat*pi/180);
   win(2) = 0;
   win(3) = -earth.o * sin(lat*pi/180);

   win = transpose(win);
```

Turn Rate of the Navigation Frame With Respect to the Earth in the Navigation Frame

```
function [wen] = trLGF_earth(ve, vn, lat, h)

% TRLGF_EARTH  Find the turn rate of the local geo frame with
respect to the earth.
%
% [wen] = trLGF_earth(Velocity east, Velocity north, Latitude,
altitude)
%
% Latitude is in decimal degrees.
%
% Written By: Brian Bleeker
%             Rob MacMillan

earth=earth_param;
Rn = (earth.a(1-earth.e^2))/((1-
earth.e^2*(sin(lat*pi/180))^2)^(3/2));
Re =  earth.a/((1-earth.e^2*(sin(lat*pi/180))^2)^(1/2));

wen(1) = ve/(Re+h);
wen(2) = -vn/(Rn+h);
wen(3) = (-ve*tan(lat*pi/180))/(Re+h);

wen = transpose(wen);
```

Turn Rate of the Body Frame With Respect to the Navigation Frame in the Body Frame

```
function [wnb]=trBODY_NED_BODY(win,wen,wib,cbn)

% TRBODY_NED_BODY   Find the turn rate of the vehicle with respect
to the navigaton
%                   frame in the body frame.
%
% [wnb] = trBODY_NED_BODY(turn rate of the earth w.r.t inertial
frame in local geo frame,
%                         turn rate of local geo frame w.r.t the
earth,
%                         raw angular data from IMU, Directional
cosine matrix)
%
%
% Written By: Brian Bleeker
%             Rob MacMillan

wnb=wib-cbn*(wen+win);
```

Skew Matrix

```
function [omeganb] = skew(wnb)
% SKEW(WNB)    Derive the skew matrix of wnb
%     WNB = [wnb.x,wnb,y,wnd.z]
% skew(wnb)
%            0     -wnb.z     wnb.y
%            wnb.z     0     -wnb.x
%           -wnb.y  wnb.x       0
%
% Written By: Brian Bleeker
%             Rob MacMillan
omeganb = [0,-wnb.z,wnb.y;wnb.z,0,-wnb.x;-wnb.y,wnb.x,0];
```

Update Directional Cosine Matrix

```
function [cbn2]=update_cbn(cbn, delt, omeganb)

% UPDATE_CBN    Update the directional cosine matrix to compute
%               new attitude angles.
%
% [cbn] = update_cbn(cbn - old values, delta time, skew(wnb))
%
% Written By: Brian Bleeker
%             Rob MacMillan
%
cbn2 = cbn + delt[cbn*omeganb];
```

REFERENCE

D.H. Titterton, J.L. Weston, *Strapdown Inertial Navigation Technology*, Peter Peregrinus Ltd. London, England, 1997.

IMU Specifications:

IMU300CC :
http://www.xbow.com/Products/Product_pdf_files/Inertial_pdf/IMU300CC.pdf

IMU400CC :
http://www.xbow.com/Products/Product_pdf_files/Inertial_pdf/IMU400CC.pdf

IMU700CA :
http://www.xbow.com/Products/Product_pdf_files/Inertial_pdf/IMU700CA.pdf

MicroPak and MicroPak II:
http://www.systron.com/