We completed most of the attitude/specific forces computer. We still have to add the latitude/longitude converter, update the gravity, and a new initialization of the directional cosine matrix. The code for the computer can be seen below.

```
%%%READ DATA
clear global
clear all
global angles;

[gtime, w.r, w.p, w.y, f.x, f.y, f.z, temp, timecounts]...
   =textread('c:/temp/IMU/pitch.txt','%f %f %f %f %f %f %f %f
%f','headerlines',8);%,'delimiter',',');

b = size(gtime);
gtime = gtime(:,1) - gtime(1,1);
f.x=f.x*9.8;
f.y=f.y*9.8;
f.z=f.z*9.8;
%%%ATTITUDE COMPUTER

%Enter the roll, pitch, yaw
yaw = 0;
pitch = 0;
roll = 0;
lat = 40;
delt = 1/207;
h = 200;
i = 1;
time = 0;
b=b-1

earth = earth_param;                        %obtain earth data
g = gravity(h, lat);
cbn = cbninit(yaw, pitch, roll);        %inital directional cosine matrix
a = [0;0;g];
a = cbn * a;
acc.n=a(1);
acc.e=a(2);
acc.d=a(3);
cnb = cbn';                             %get cnb to relate NED to body for wnb
euler(cbn, i);                     %obtain the euler angles


for i = 1:b
    if i == 1
        vtemp.n = 0;
        vtemp.e = 0;
        vtemp.d = 0;
        wib = [0,0,0]';

        [pos, vtemp] = position(acc, vtemp, lat, h, g, delt);

        p.n(1) = 0;
        p.e(1) = 0;
        p.d(1) = 0;

     else
        %IMU data
        a = [f.x(i); f.y(i); f.z(i)];
        a = cbn * a;
        acc.n = a(1);
```

```
        acc.e = a(2);
        acc.d = a(3);

        [pos, vtemp] = position(acc, vtemp, lat, h, g, delt);

        if rem(i,2000)==0
            i
        end

        p.n(i) = pos.n + p.n(i-1);            %positions
        p.e(i) = pos.e + p.e(i-1);
        p.d(i) = pos.d + p.d(i-1);

        wib = [w.r(i)*pi/180; w.p(i)*pi/180; w.y(i)*pi/180];
%IMU angular rates
    end

    win = trEARTH_IF_NED(lat);                  %turn rate of the earth w.r.t
inertial frame in the NED frame
    wen = trLGF_earth(vtemp, lat, h);           %turn rate of the NED frame w.r.t
earth in the NED frame
    wnb = trBODY_NED_BODY(win,wen,wib,cnb); %turn rate of the body w.r.t NED in
the body frame
    skewwnb = skew(wnb);                        %skew wnb
    cbn = update_cbn(cbn, delt, skewwnb);   %update the directional cosine matrix
    cnb = cbn';
    euler(cbn, (i+1))                     %obtain the euler angles


end

%Roll, Pitch, Yaw
figure(1), subplot(311), plot(gtime, angles.r), grid
title('Roll - Time in seconds')

subplot(312), plot(gtime, angles.p), grid
title('Pitch - Time in seconds')

subplot(313), plot(gtime, angles.y), grid
title('Yaw - Time in seconds')

%Position
figure(2), subplot(311), plot(p.n), grid
title('Position Movement North')

subplot(312), plot(p.e), grid
title('Position Movement East')

subplot(313), plot(p.d), grid
title('Position Movement Down')
```

We also changed the function that takes the directional cosine matrix to obtain the euler angles. We added the state where the pitch is approaching +-90 degrees. At that state we need an alternate route in finding the roll and yaw. Below shows the code we have thus far.

```
function euler(cbn, i)
% euler(cbn, position)    Derive the euler angles from cbn
%
% Written By: Brian Bleeker
%             Rob MacMillan
```

```
global angles;

angles.p(i) = (asin(-cbn(3,1)))*180/pi;

if abs(angles.p(i)-90) <= 1
   if rem(i,2)==0
      angles.r(i) = angles.r(i-1);
      angles.y(i) = atan2((cbn(2,3)-cbn(1,2)),(cbn(1,3)+cbn(2,2)))*180/pi +
angles.r(i);
   else
      angles.y(i)=angles.y(i-1);
      angles.r(i) = -atan2((cbn(2,3)-cbn(1,2)),(cbn(1,3)+cbn(2,2)))*180/pi +
angles.y(i);
    end
elseif abs(angles.p(i)+90) <= 1
   if rem(i,2)==0
      angles.r(i) = angles.r(i-1);
      angles.y(i) = atan2((cbn(2,3)+cbn(1,2)),(cbn(1,3)-cbn(2,2)))*180/pi -
angles.r(i);
   else
      angles.y(i)= angles.y(i-1);
      angles.r(i) = atan2((cbn(2,3)+cbn(1,2)),(cbn(1,3)-cbn(2,2)))*180/pi -
angles.y(i);
    end

else
    angles.y(i) = (atan2(cbn(2,1),cbn(1,1)))*180/pi;
    angles.r(i) = (atan2(cbn(3,2),cbn(3,3)))*180/pi;
end
```

We did several tests on the IMU to see if the attitude computer and euler angle functions are working properly. First we did a 360 degree test of the yaw axis. We rotated the IMU, by hand, 360 degrees. As seen in the graph below, everything worked pretty well.

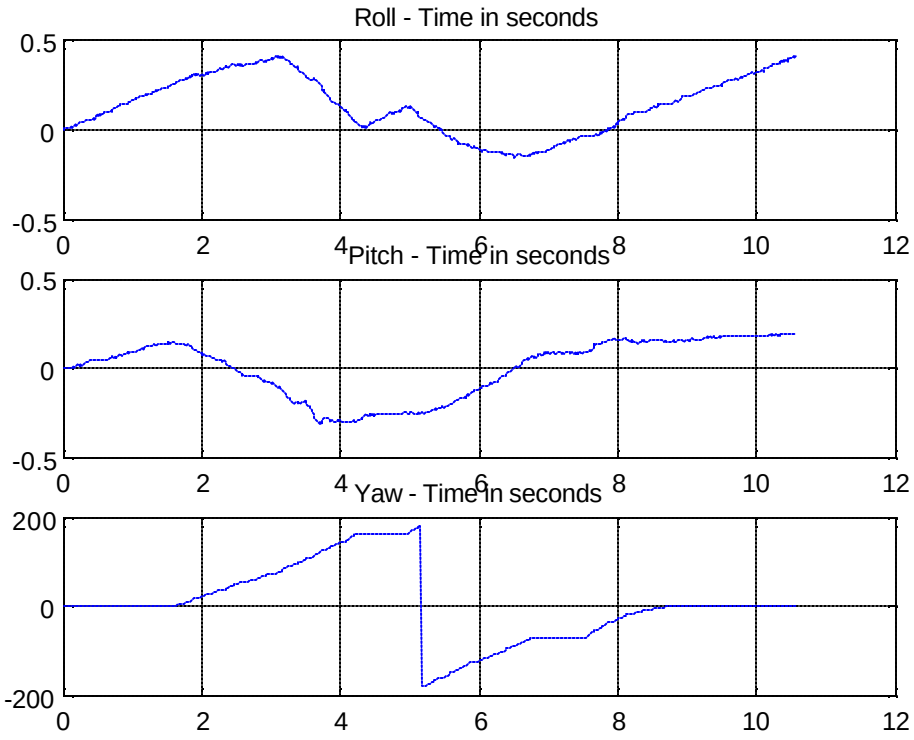
Roll - Time in seconds
0.5
0
-0.5
0
2
4
6
8
10
12
Pitch - Time in seconds
0.5
0
-0.5
0
2
4
6
8
10
12
Yaw - Time in seconds
200
0
-200
0
2
4
6
8
10
12

Fig - Roll, Pitch, and Yaw angles for 360 degree yaw turn

The roll, and pitch did change less than a half degree over the 10 second test, but that is very minimal for what is supposed to be a noisy system.
Then we did a test on the roll. Once again, by hand, we rotated the IMU 360 degrees in the roll axis. Shown on the next page is the graph obtained from this test. It is seen that the roll angle rotates a full 360 degrees as expected. The pitch does drop almost 10 degrees, but that is ok because the IMU roll rotation was tested by hand and wasn't rotated exactly just on it's roll axis.
Next, we tested the pitch axis. Once again, by hand we rotated the IMU 360 degrees. Show on the next page is a graph of this test. It can be see that at 90 degrees the roll and yaw change unexpectedly at this point. We'll have to do further tests later to find out why this is.
We wanted to do another test to make sure that at least part of the pitch euler angle program was working properly. So we moved the IMU, about 45 degrees up and then back down to level ground. This worked properly as expected.

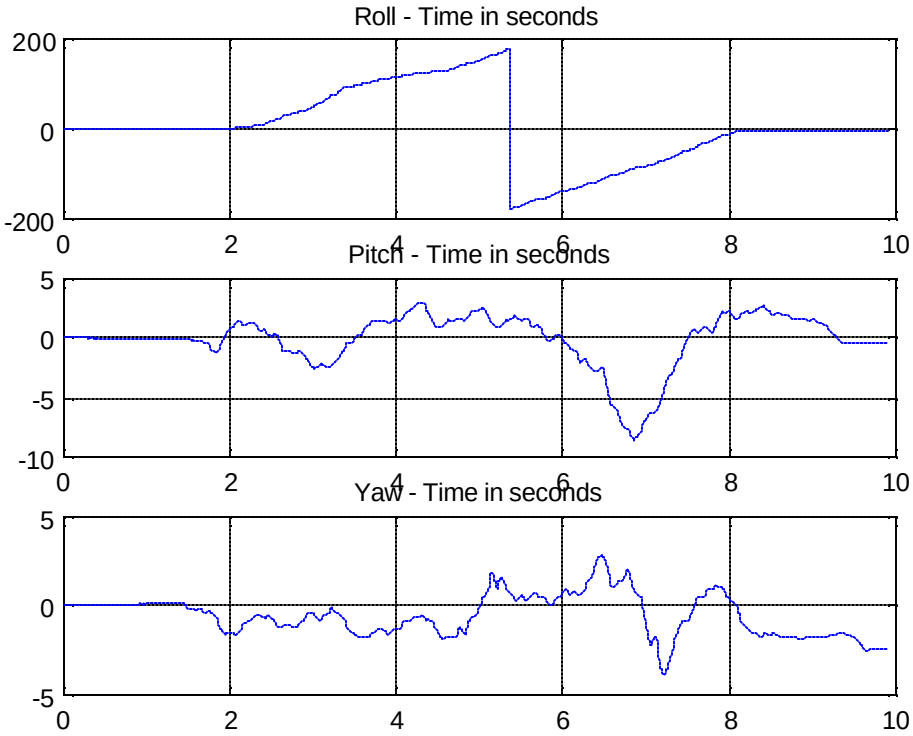

Fig - Roll, Pitch, and Yaw angles for 360 degree roll test

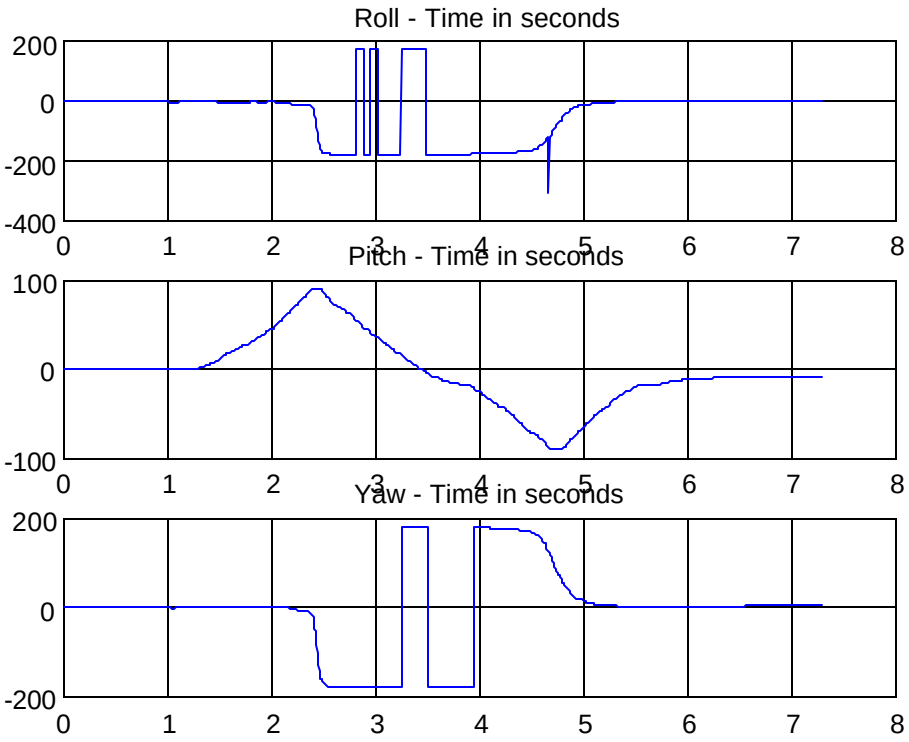

Fig - Roll, Pitch, and Yaw angles for 360 degree roll test

The next step we want to take is to fix the directional cosine matrix initialization. This involves a series of equations, taking data from the IMU when it is in a fixed position and aligning the axis of the IMU to that of NED. Below is the code for this process.

```
function [cbn]=angle_adjust(a, g, win, lat)
% angle_adjust(a, g, win)    Create Cbn from initial data
%
```

```
% a= the x,y and z acceleration of the body measured in G's
% g= the acceleration of gravity at our latitude
% win= turn rate of the earth w.r.t inertial frame in the NED frame
% lat= latitude
%
% Written By: Brian Bleeker
%             Rob MacMillan
earth= earth_param;

%third column vector
C31 = -a(1)/g;
C32 = -a(2)/g;
C33 = -a(3)/g;

%first column vector
C11 = (win(1)/(earth.o*cos(lat)))-((a(1)*tan(lat))/g);
C12 = (win(2)/(earth.o*cos(lat)))-((a(2)*tan(lat))/g);
C13 = (win(3)/(earth.o*cos(lat)))-((a(3)*tan(lat))/g);

%resolution of second column vector
C21 = (-C12*C33) + (C13*C32);
C22 = (C11*C33) - (C31*C13);
C23 = (-C11*C32) + (C31*C12);

%initial CBN matrix

cbn=[C11,C12,C13;C21,C22,C23;C31,C32,C33];
```