The ambition for the day is to clear up the problems faced last week in trying to initialize the system and make the initial cbn correspond to the data we were receiving from the IMU. The problem with doing the initialization this way is if the data is bad or noisy then the cbn will be the same way. To resolve cbn properly we need a sensor accurate to 10^-5, with this sensor we can resolve the turn rate of the earth easily and calibrate the cbn accordingly, however the IMU300CC cannot resolve anything near this. The IMU300CC is specified to have a noise value close to 10^-3 which eliminates our hope of being able to resolve the turn rate of the earth. We are therefore reverting back to initializing our system with zeros.

We now need to try to eliminate some of this noise from our system in order to more accurately resolve the turn rates and the movement of the system. The first thing is an averaging filter which finds the min and max as well as the average of the first 200 points of the input data on all six axis. Finding this data will give us a good idea as to how the noise in the system behaves. The filter itself checks each point in the data and if it is above or below the max or min of the noise values we subtract the average to eliminate some noise. If it is below the max or min then the value of the point is truncated to zero. By ensuring that the first 200 points (1 second) of every sample take is of the sensor standing still this will minimize the loss of important data. The code as well as the cleaned up signal are shown below. For acceleration data we ran the averaging function as well as the filter after the raw data was converted to north, east, down coordinates by cbn.

```
function [ave, mx, mn] = average(acc,w,g,cbn)

% Written By: Brian Bleeker
%             Rob MacMillan

temp_x = 0;
temp_y = 0;
temp_z = 0;
temp_wr = 0;
temp_wp = 0;
temp_wy = 0;
mx.x = -10;
mn.x = 10;
mx.y = -10;
mn.y = 10;
mx.z = -10;
mn.z = 10;
mx.wr = -10;
mn.wr = 10;
mx.wp = -10;
mn.wp = 10;
mx.wy = -10;
mn.wy = 10;
```

```
for (i = 1:200)
   a = [acc.x( i);acc.y( i);acc.z( i)];
   a = cbn * a;
         temp_x = a(1) + temp_x;
   temp_y = a(2) + temp_y;
   temp_z = a(3) + temp_z;
   temp_wr = w.r(i) + temp_wr;
   temp_wp = w.p(i) + temp_wp;
   temp_wy = w.y(i) + temp_wy;
   if a(1) < mn.x
     mn.x = a(1);
   elseif a(1) > mx.x
       mx.x = a(1);
   end
   if a(2) < mn.y
     mn.y = a(2);
   elseif a(2) > mx.y
       mx.y = a(2);
   end
         if a(3) < mn.z
     mn.z = a(3);
   elseif a(3) > mx.z
       mx.z = a(3);
   end
         if w.r(i) < mn.wr
     mn.wr = w.r(i);
   elseif w.r(i) > mx.wr
       mx.wr = w.r(i);
   end
         if w.p(i) < mn.wp
     mn.wp = w.p(i);
   elseif w.p(i) > mx.wp
       mx.wp = w.p(i);
   end
         if w.y(i) < mn.wy
     mn.wy = w.y(i);
   elseif w.y(i) > mx.wy
       mx.wy = w.y(i);
   end
end
ave.x = temp_x/200;
ave.y = temp_y/200;
ave.z = temp_z/200-g;
ave.wr = temp_wr/200;
ave.wp = temp_wp/200;
```

```
ave.wy = temp_wy/200;
mx.z = mx.z-g;
mn.z = mn.z-g;
```

```
Function to filter acceleration data after cbn
function [a] = acc_filter(a, ave, mx, mn, g)

% Written By: Brian Bleeker
%           Rob MacMillan


if (a.n > mx.x) | (a.n < mn.x)
        a.n = (a.n - ave.x);
else
        a.n = 0;
end
if (a.e > mx.y) | (a.e < mn.y)
        a.e = (a.e - ave.y);
else
        a.e = 0;
end
if (a.d > g + mx.z) | (a.d < g + mn.z)
        a.d = (a.d - ave.z);
else
        a.d = g;
end
```

```
Function to filter angular rate data
unction [w] = ang_filter(w, ave, mx, mn)

% Written By: Brian Bleeker
%           Rob MacMillan


len = length(w.r);
for i = 1:len
        if (w.r(i) > mx.wr) | (w.r(i) < mn.wr)
        w.r(i) = (w.r(i) - ave.wr);
   else
     w.r(i) = 0;
   end
        if (w.p(i) > mx.wp) | (w.p(i) < mn.wp)
        w.p(i) = (w.p(i) - ave.wp);
   else
     w.p(i) = 0;
   end
```

```
        if (w.y(i) > mx.wy) | (w.y(i) < mn.wy)
        w.y(i) = (w.y(i) - ave.wy);
    else
      w.y(i) = 0;
    end
end
```

The following is a graph comparing the original yaw with movement and then a filtered graph of the same yaw movement.

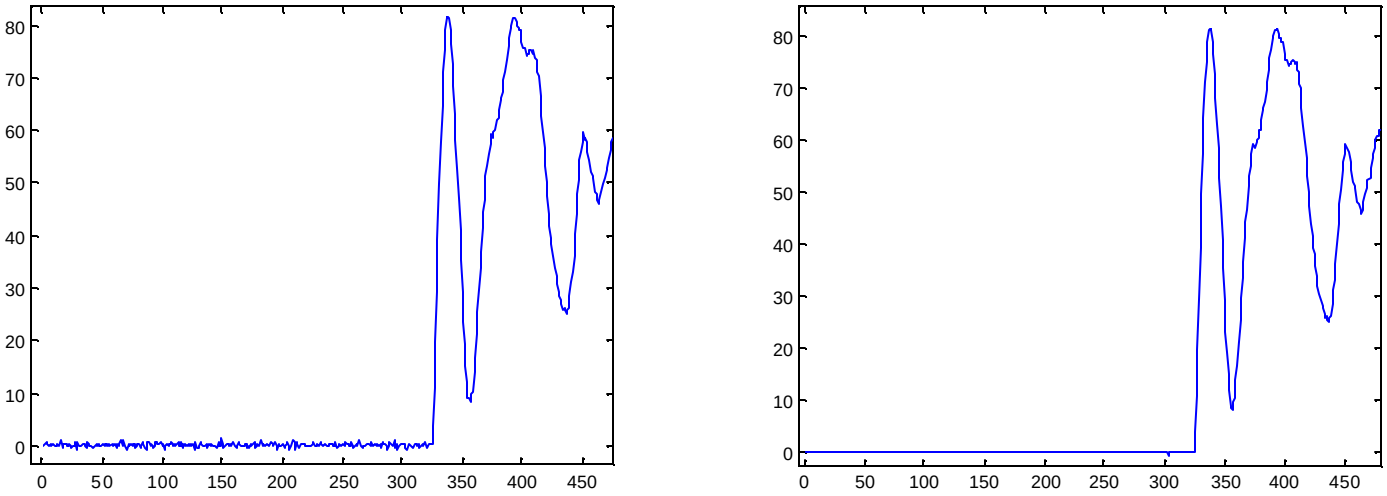
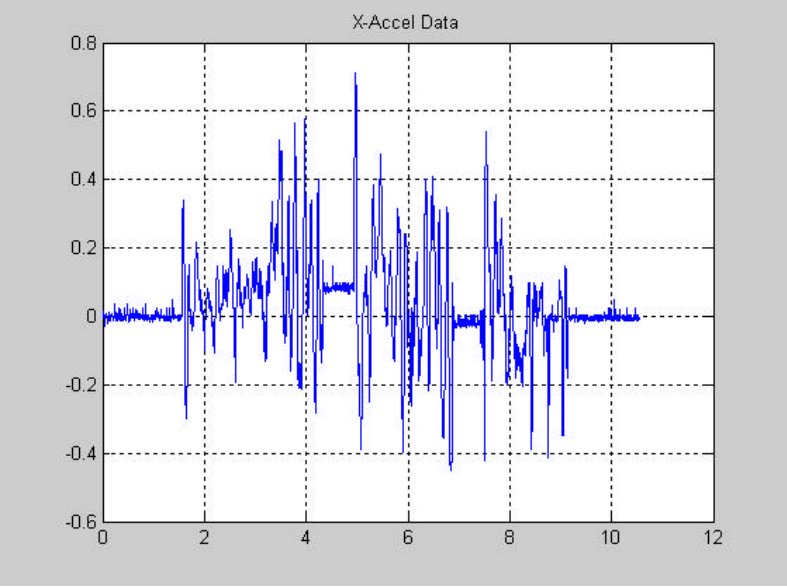


Fig - Raw Yaw Data(left), Filtered Yaw Data(right)

When running the average filter with the accelerometer data, we found that it didn't work as well. So we analyzed the original data closely. We found that when the IMU started stationary, the noise was about +/- 0.06 g's. When the IMU was moving, but not in the x-axis, spikes ranged from about 0.7 g's to about -0.5 g's. This number can't be filtered using our average filter because this noise is large enough to be movement.

Fig - x-axis acceleration data with only movement on the yaw axis.



So we decided to try to use a median filter. This is done by selecting a window range, of about 15, and looking at the data in that window. Then, we take the median of that set of data and set the entire 15 data points to the median. In theory, this should remove the spikes, but I didn't work to well. In fact, it actually made some of the testing trials even worse.

Then we decided we might want to try a statistical filter. First we checked the standard deviation of the data (when it was moving and when it wasn't). There didn't appear to be a large difference between the two. Then we tried looking at the variance of the same data. There appeared to be a large difference in the variance when the IMU was moving on that axis then when it wasn't.

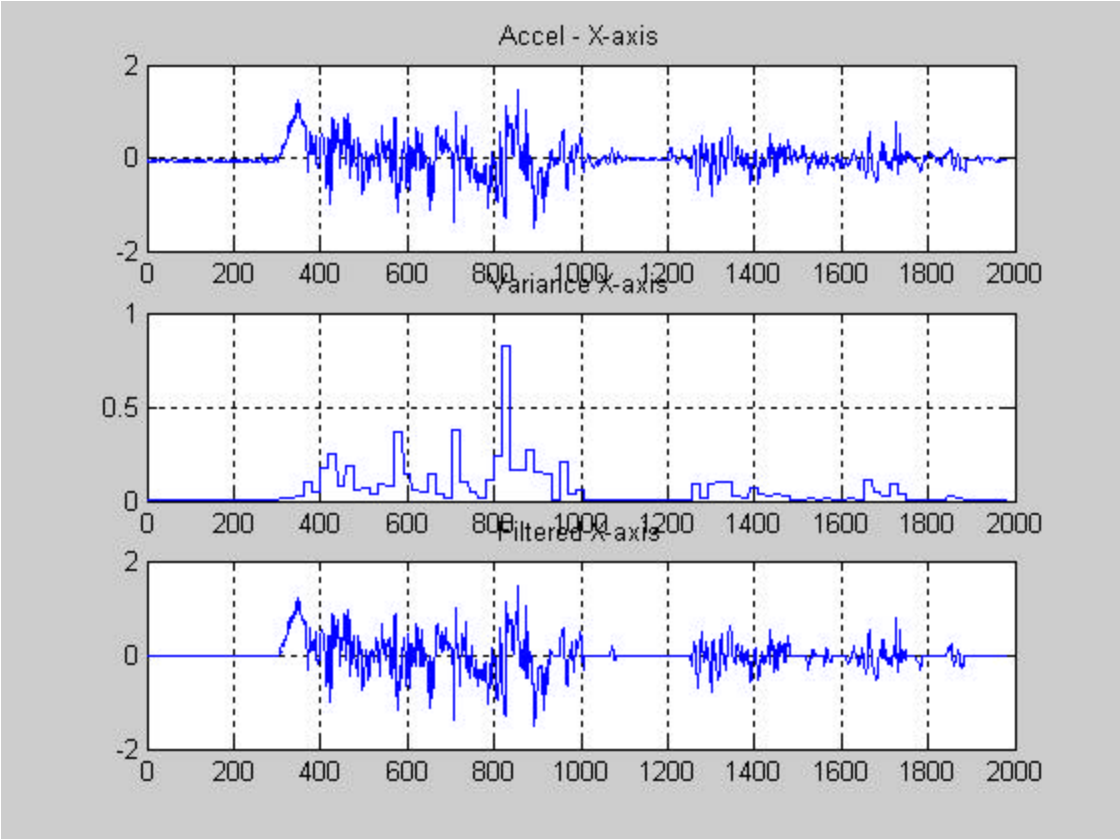


Fig - X-axis Acceleration (Raw data, variance of data, Filtered Data)

```
function [f2, w2]=cov_filter2(f,w,range,g,gtime)

% Written By: Brian Bleeker
%             Rob MacMillan

len=length( f.x);
g
```

```
for i = 1 : range + 1 : (len)
  if (i + range) < len
    temp6(i:i+range)= cov( f.x(i:i+range));
    if cov( f.x(i : (i + range))) > .01
      f2.x(i : (i + range)) = (f.x(i : (i + range)));
    else
      f2.x(i : (i + range)) = 0;
    end
    %temp5(i:i+range)= cov( f.y( i:i+range));
    if cov( f.y(i : (i + range))) > .05
      f2.y(i : (i + range)) = (f.y(i : (i + range)));
    else
      f2.y(i : (i + range)) = 0;
    end
    temp4(i:i+range)= cov( f.z(i:i+range));
    if cov( f.z(i : (i + range))) > .05
      f2.z(i : (i + range)) = (f.z(i : (i + range)));
    else
      f2.z(i : (i + range)) = g;
    end



    temp1(i:i+range)= cov( w.r(i:i+range));
    if cov( w.r(i : (i + range))) > 6.155
      w2.r(i : (i + range)) = (w.r(i : (i + range)));
    else
      w2.r(i : (i + range)) = 0;
    end
    temp2(i:i+range)= cov( w.p(i:i+range));
    if cov( w.p(i : (i + range))) > 6.155
      w2.p(i : (i + range)) = (w.p(i : (i + range)));
    else
      w2.p(i : (i + range)) = 0;
    end
    temp3(i:i+range)= cov( w.y( i:i+range));
    if cov( w.y(i : (i + range))) > 6.155
      w2.y(i : (i + range)) = (w.y(i : (i + range)));
    else
      w2.y(i : (i + range)) = 0;
    end

  else
    temp6(i : len) = cov(f.x(i : len));
    if cov( f.x(i : (len))) > .01
      f2.x(i : (len)) = (f.x(i : (len)));
```

```
    else
      f2.x(i : (len)) = 0;
    end
    %temp5(i : len) = cov(f.y(i : len));
    if cov(f.y(i : (len))) > .05
      f2.y(i : (len)) = (f.y(i : (len)));
    else
      f2.y(i : (len)) = 0;
    end
          temp4(i : len) = cov(f.z(i : len));
    if cov(f.z(i : (len))) > .05
      f2.z(i : (len)) = (f.z(i : (len)));
    else
      f2.z(i : (len)) = g;
    end

    temp1(i : len) = cov(w.r(i : len));
    %cov(w.r(i:len))
    if cov(w.r(i : (len))) > 6.155
      w2.r(i : (len)) = (w.r(i : (len)));
    else
      w2.r(i : (len)) = 0;
    end
    temp2(i : len) = cov(w.p(i : len));
    %cov(w.p(i:len))
    if cov(w.p(i : (len))) > 6.155
      w2.p(i : (len)) = (w.p(i : (len)));
    else
      w2.p(i : (len)) = 0;
    end
          temp3(i : len) = cov(w.y(i : len));
    %cov(w.y(i:len))
    if cov(w.y(i : (len))) > 6.155
      w2.y(i : (len)) = (w.y(i : (len)));
    else
      w2.y(i : (len)) = 0;
    end
  end
end
```

We have completed the testing platform. We will eventually get a digital picture of it for the website and will obtain the dimensions of it next week.