Filtering Data

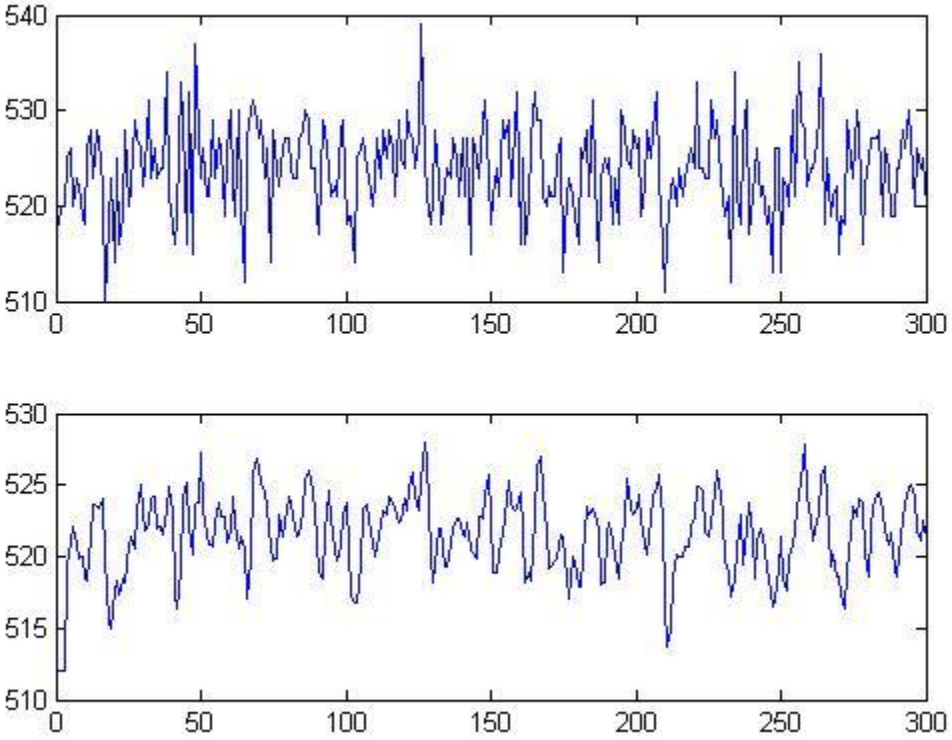
By Rahul Jain and Mark Epelbaum, EE444 S'11

Table of Contents

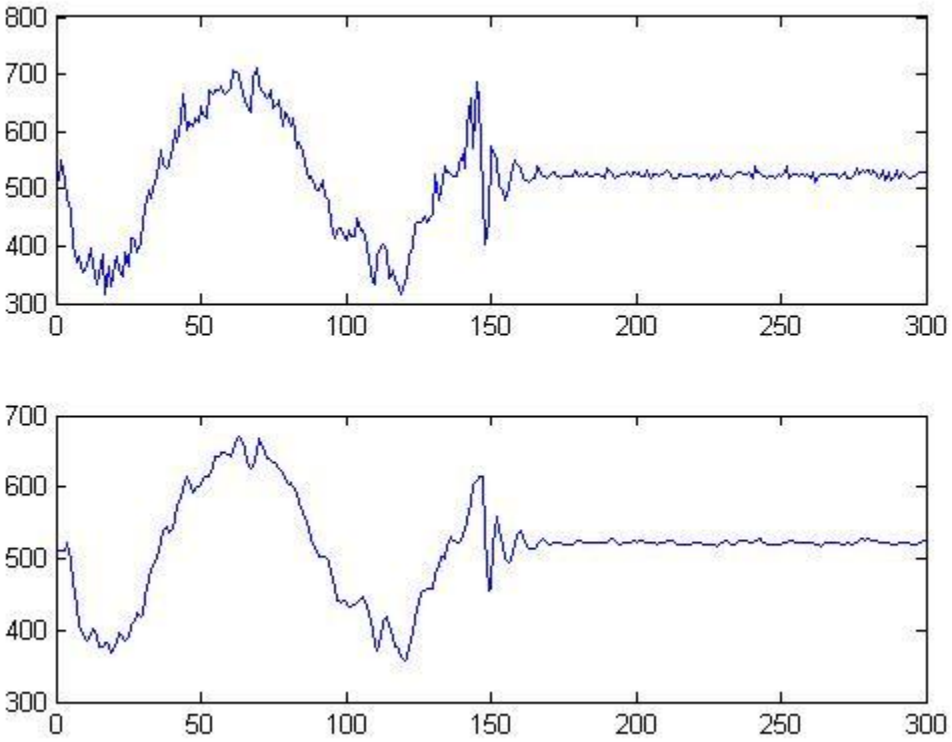
Introduction

At this point we realized the need to filter the data coming from the gyro. We began our investigation of Kalman filters (which was suggested by the professor) but decided to implement a simple averaging algorithm ($1/6$ (*vali-3* + 2 *vali-2* + 2 *vali-1* + *vali*)) in the meantime to see the effect on the data.

Simple filter filtering noise (at rest):



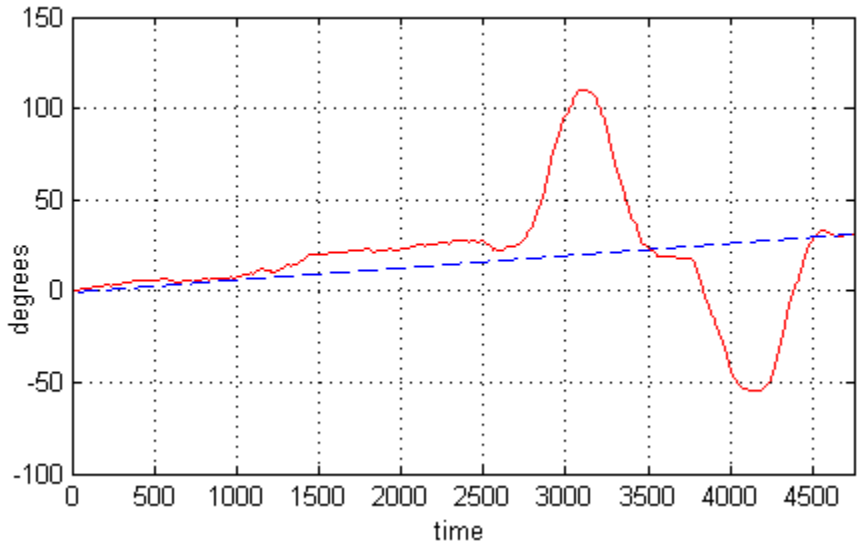
Simple filter filtering rotation:



Combining Gyro & Accelerometer

An inherent problem with gyroscopes is the tendency for their output to drift over time. This drift occurs from noise and error compounding in the calculations over the time of operation, also known as integration error. This drift exists in the calculations, not the gyro's output voltage. In the following graph, the red line corresponds to the calculated angle of rotation while the blue line shows

the drift in the baseline value over time.

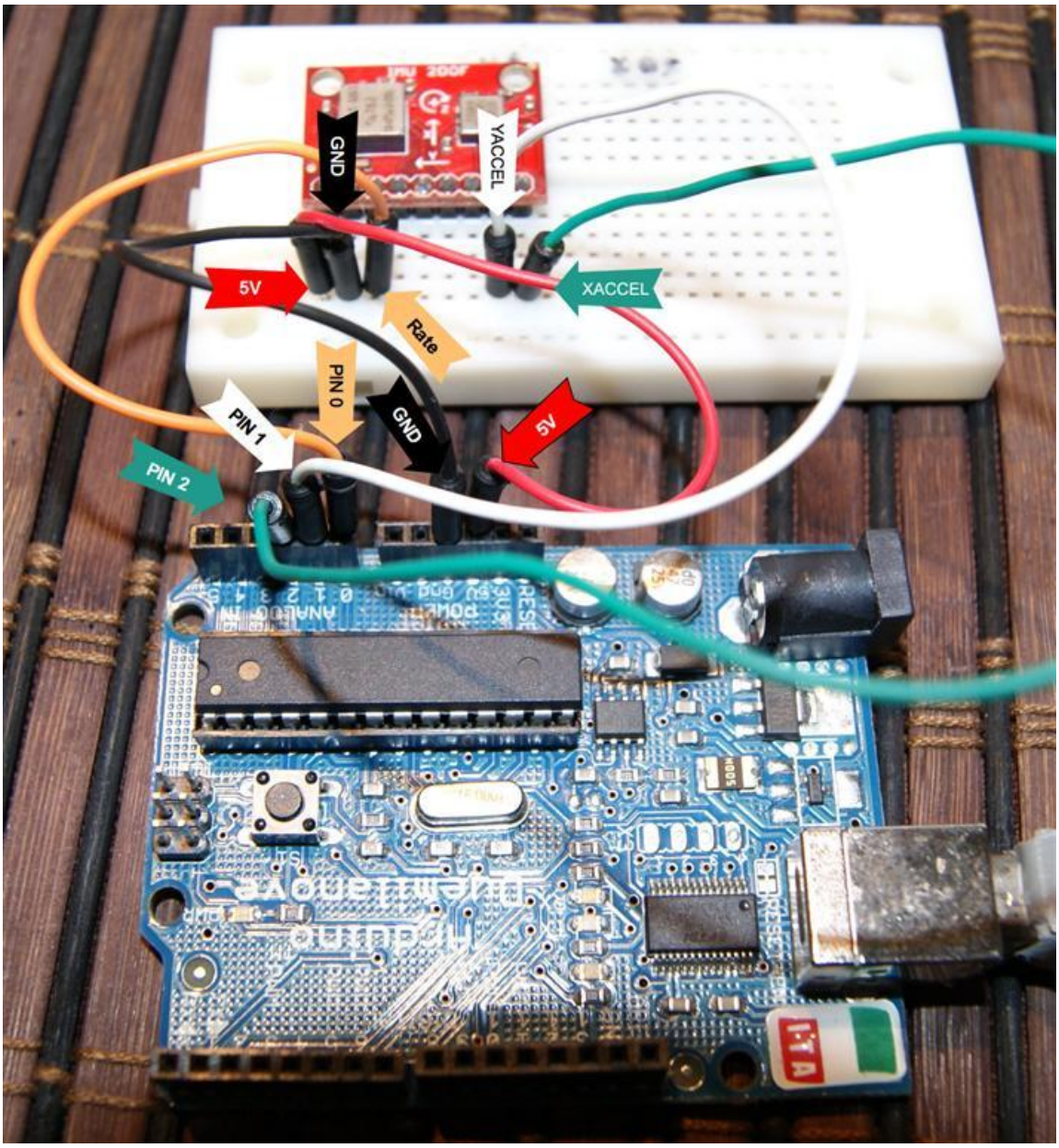


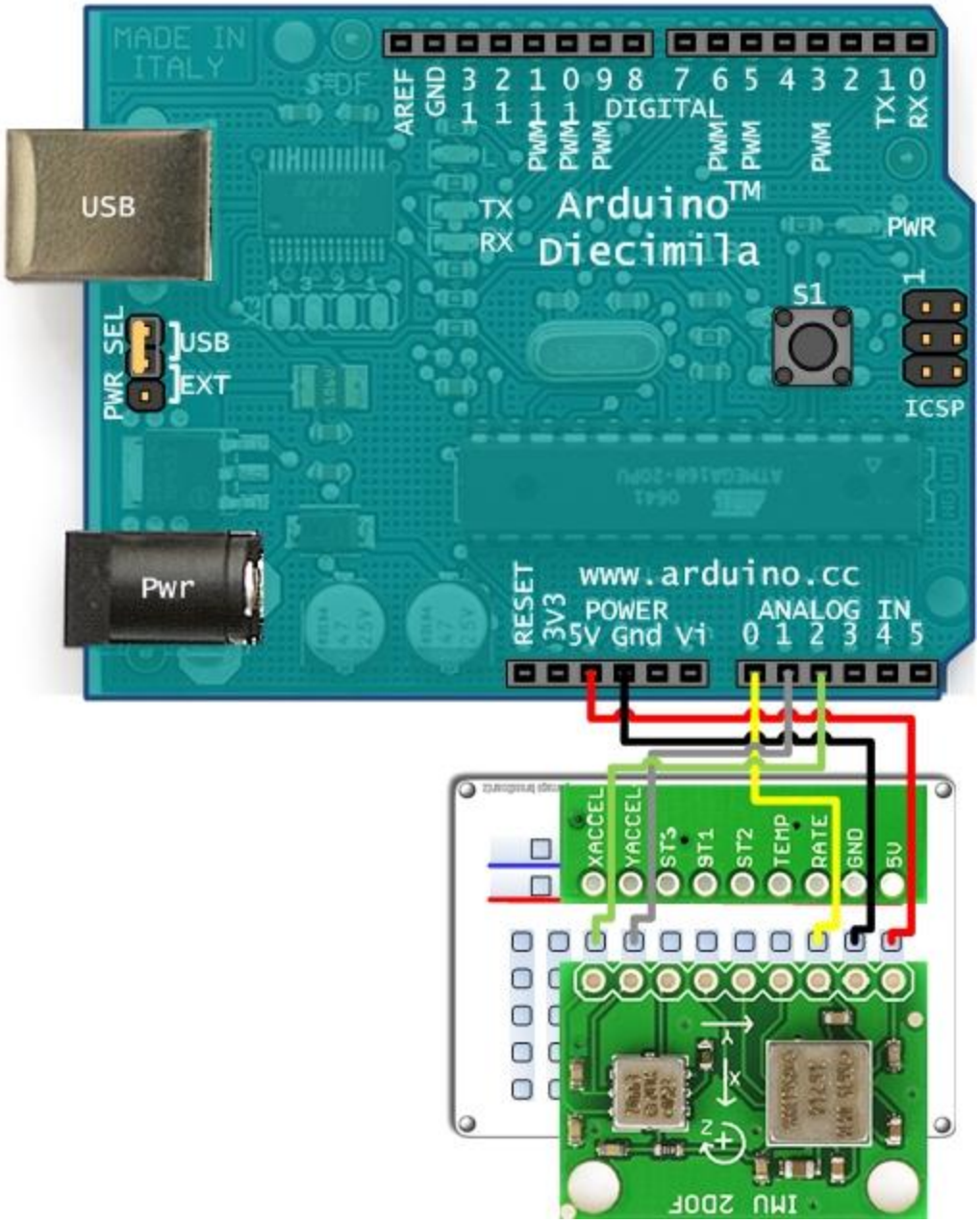
One possible way to compensate for gyro drift is by using it in conjunction with an accelerometer. An accelerometer is used to measure acceleration within a certain range of motion. In mobile phones an accelerometer can interpret the orientation of the phone to change the display from portrait to landscape mode or interpret sudden motions such as shaking for mobile application interaction. Movement is calculated similar to the gyro by using changes in the accelerometer's output voltage to determine changes in orientation. Depending on the angle of the device in relation to the ground, a microscopic amount of mass moves within the accelerometer circuit to cause this change in signal. An accelerometer can measure an absolute orientation.

A gyroscope works by interpreting the shift in positioning from a set rate of rotation within the X, Y, and Z — left/right, up/down, forward/backward — axis (3-axis gyroscopes are available on the market, the gyroscope mentioned earlier is only capable of measuring rotation on one axis). In consumer devices, this is accomplished through a constantly vibrating microscopic plate called a proof mass. For example, when a user tilts his phone toward the sky, the gyroscope is able to compare this movement to its ambient state and discern the pitch, roll, and yaw. The constant movement of the gyroscope means the orientation is never lost and the device is able to clearly discern its placement at all times. A gyroscope measures relative orientation.

When a gyroscope and accelerometer are combined, it is possible to more accurately determine/measure absolute/relative orientation/movement since the devices are used to complement each other. This combination results in a total of up to six orientation measurements at all times (depending on the Inertial Measurement Unit (IMU) used). By combining a gyroscope and accelerometer, it is possible to better stabilize cameras for clearer pictures, gaming applications can better interpret user motions for greater realism, and navigation applications can more accurately guide users by measuring their movements. A gyroscope and accelerometer are used together to create a more accurate measurement of overall movement and location through space by providing constant cross-referenced measurements of spatial placement and acceleration in the case of our rover.

We have determined the need to investigate the combination of a gyroscope and an accelerometer to provide more accurate positional data and to filter out noise and drift.



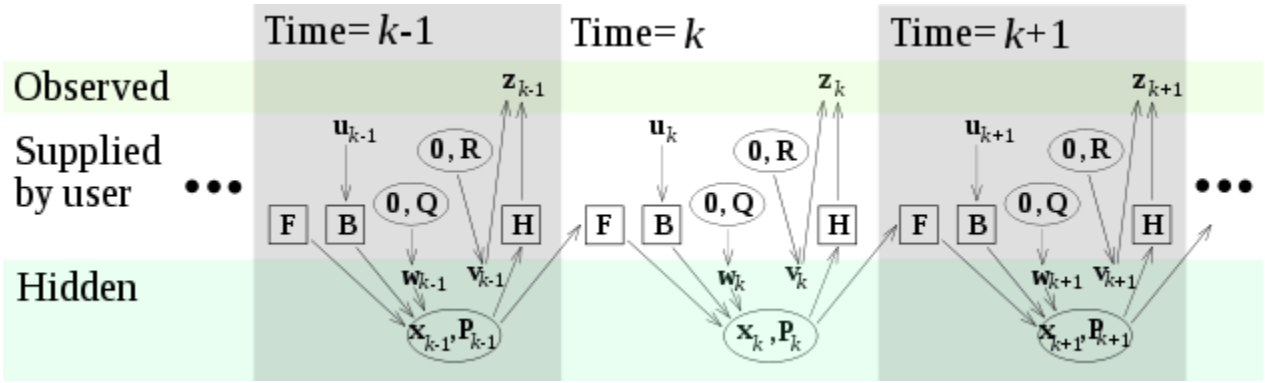


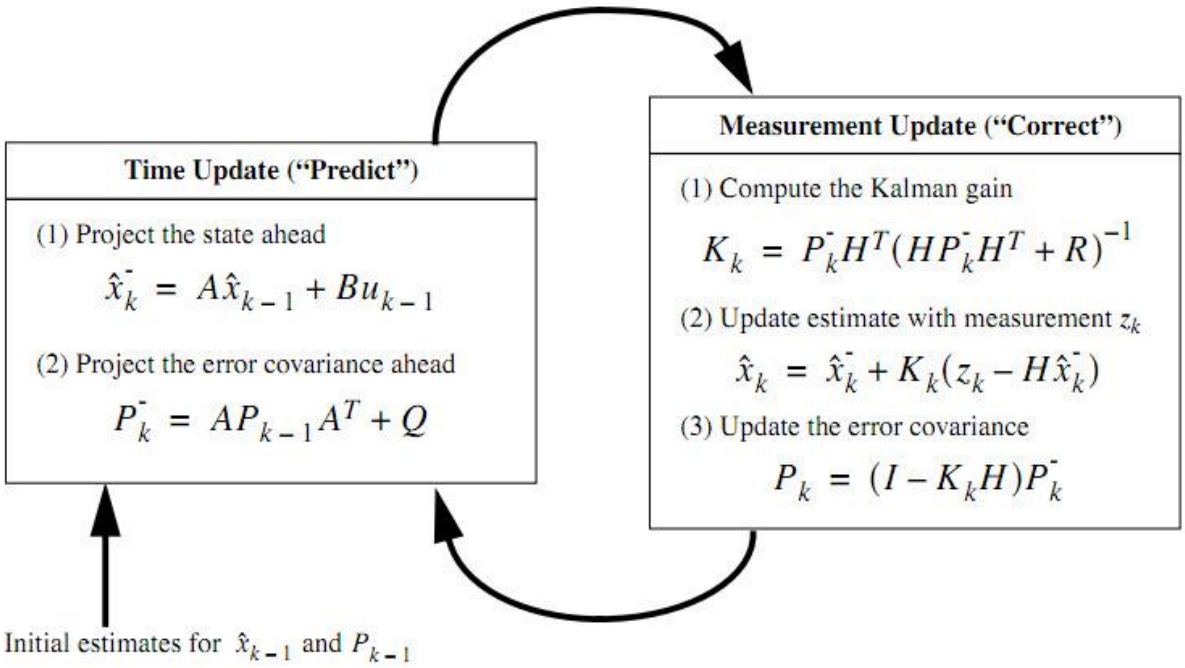
The Kalman Filter

We had to find a way to resolve the noise and drift issues presented by the gyro, because the the output was unacceptable. The rover needs the output of the gyro to be as accurate as possible to keep it going in a straight line. We did some research and found out that a gyro is commonly used with an accelerometer for a more accurate reading. We found out that the output of an accelerometer is more accurate and less noisy because the accelerometer spits out real time data rather than a time average from the Gyro. The instructor directed us to research Kalman filters for this application since they are used to minimize noise in this specific application. The Kalman filter is often used to tie the two

sensors together, the gyro measuring rotation, and the accelerometer minimizing the error that the gyro creates (accelerometer is offset from center of rotation, perhaps delta pattern).

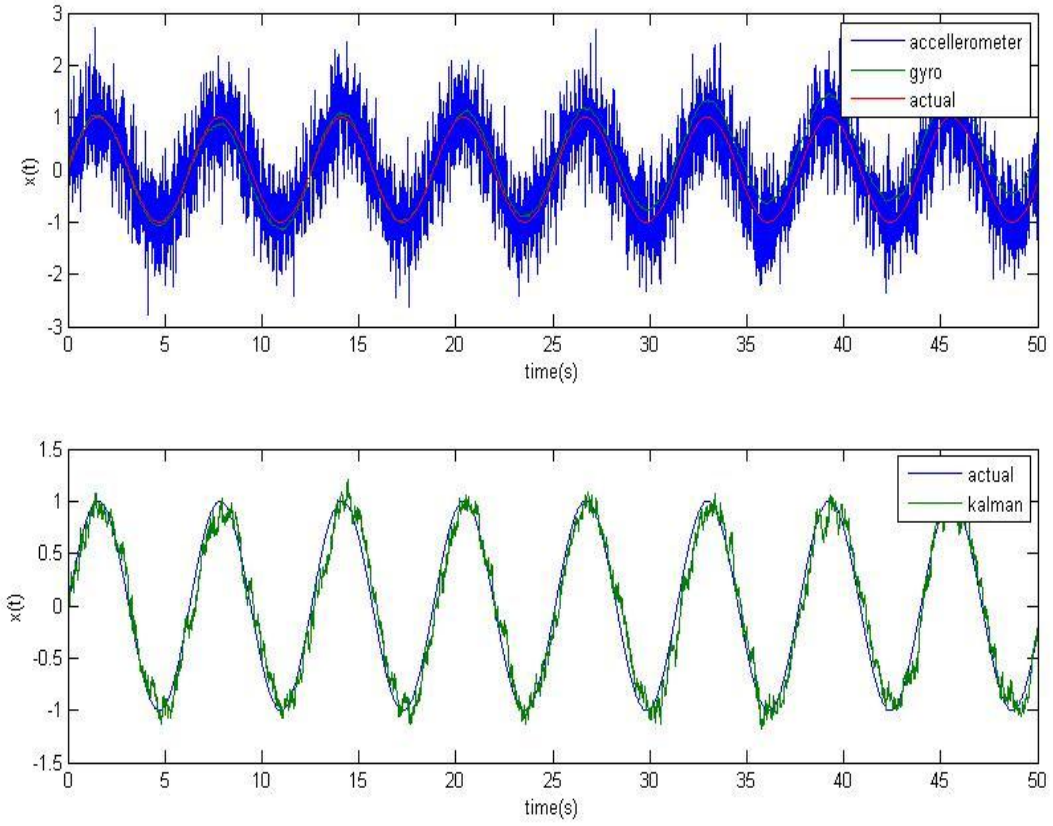
The Kalman filter is a set of mathematical equations that provides an efficient computational(recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and future states and performs even when the precise nature of the modeled system is unknown. Our research demonstrated the complexity of Kalman filters as depicted below:





After passing our noisy signal through a Kalman Filter, the final output signal was significantly cleaner than the original signal. The inputs to the Kalman filter are the outputs from the Gyro, the accelerometer, and a linear model of the system. The output of the filter is cleaner signal with little to no drift.

Sample Kalman filter:



As you can see from the graph above, the original signal (the outputs of the gyro and accelerometer) was very noisy. After passing the signal through a Kalman Filter, the resulting cleanliness of the signal and lack of noise is apparent. We compared this sample data with our signal (this sample was created for a similar application) and noticed a very close resemblance between the signals. The Kalman filter is a very useful finding and a powerful tool that we will definitely need.

We now know that that the Kalman filter is the solution to our noise problem. We have since done studying of the Kalman Filter but soon came to realize that it was a complex process requiring extensive knowledge of signal processing, random processes, and probability. We realized that instead of trying to figure out how a Kalman filter works, we just needed to understand it enough to make it work for our purposes and apply it to our data. Luckily after doing lots of searching, we found a sample MATLAB code online implementing a Kalman Filter for a gyro and accelerometer combination (producing the above graph). This code is being taken with a grain of salt as we are yet to investigate its accuracy and if we can model our system into it. This is what we are currently working on.

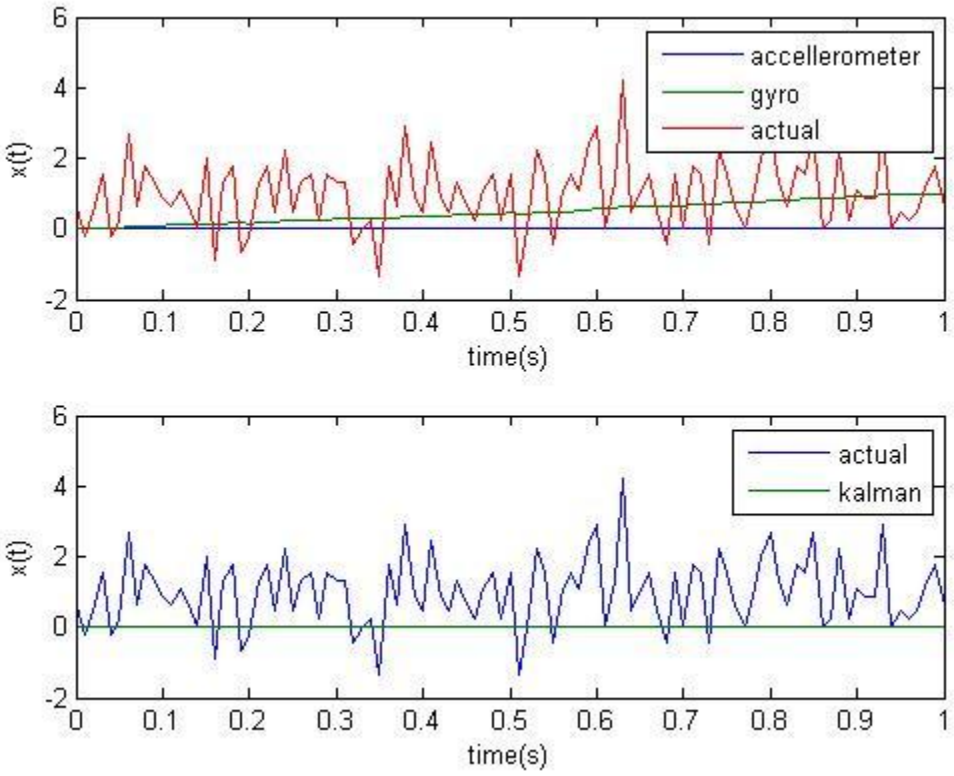
Kalman Filter - Implementation

Team PITA has made progress with implementing a Kalman filter in our system. The current Kalman filter samples data for 1 second at a rate of 100 samples a second (this is the current algorithm,

this will be tightened during testing to less than one second). This means that data will be read every .01 seconds for one second at a time to get 100 readings every second. This data, which is translated into an angle, will then be passed to the Kalman filter which will approximate the gyro's rotation from center while filtering out noise. If the gyro is off center even by a little bit we will later implement a corrective feedback mechanism where the motors can correct the rovers trajectory. This process will keep repeating every second. For now we have decided to sample 100 readings every second, but this may change when we experiment with our IMU to optimize the results. Team PITA was able to modify code from Rotomotion (tilt.m and tilt.c) to suite our needs since developing the Kalman code from scratch would have been too difficult. We preserved the Kalman Filter code, but changed the inputs to the Kalman filter by creating a separate function and isolating our code from the one we found. Our inputs to the Kalman filter were the outputs computed angles from the gyro and accelerometer.

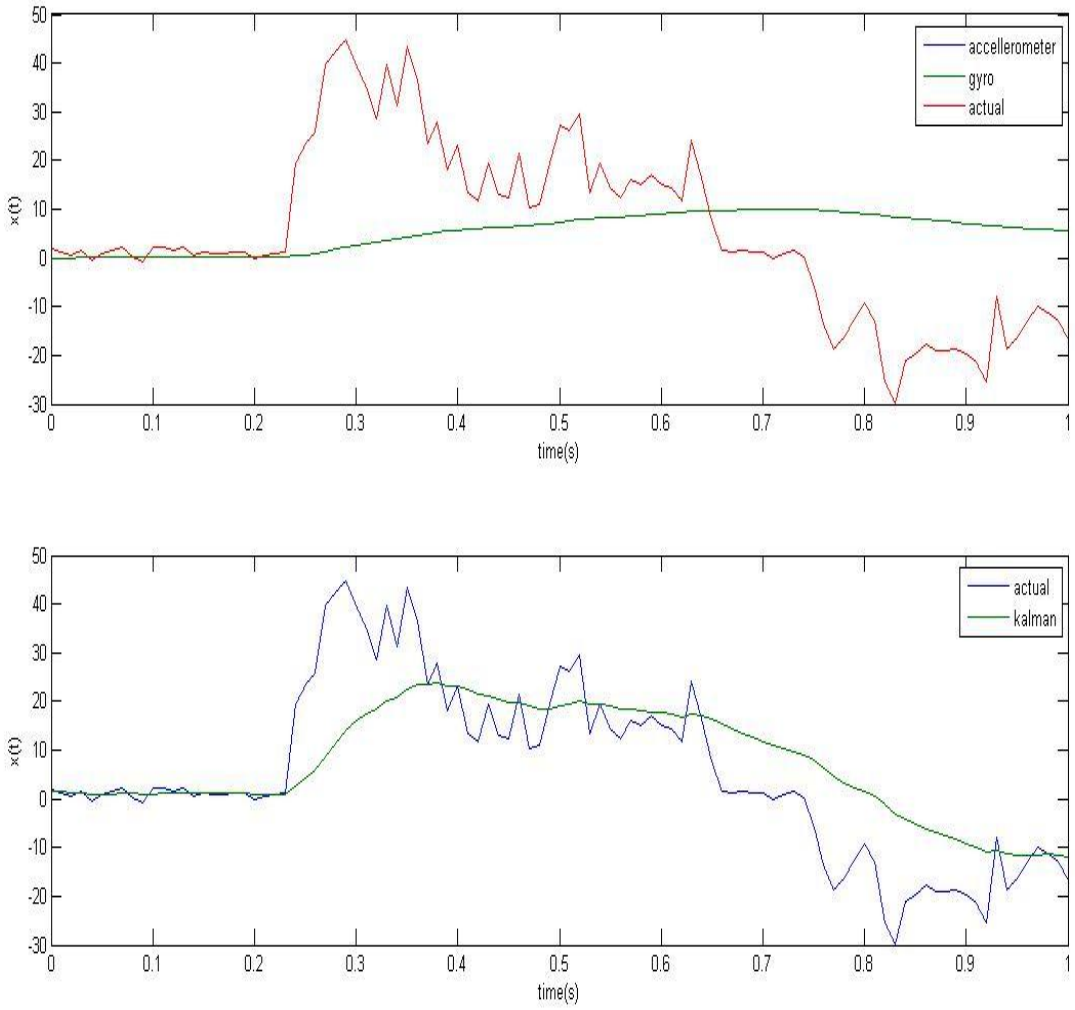
We currently have working MATLAB code that shows the (simulated) smoothing effects of the Kalman filter on the gyroscope at rest (accelerometer data simulated at 0) and rotating (accelerometer data mirroring gyroc input). Team PITA has identified and ordered an IMU to fill the accelerometer component of the equation.

These are the outputs we observed (first set is at rest and second set is in motion):



The graphs shown above shows the response of a still gyro. The red and blue jitter, represented as actual in the legend, in the two graphs above represent the data from the still gyro, which is also the input to our Kalman filter. The green gyro line in our first graph is the degree that our microcontroller detects our gyro to be off centered by. Since the gyro isn't

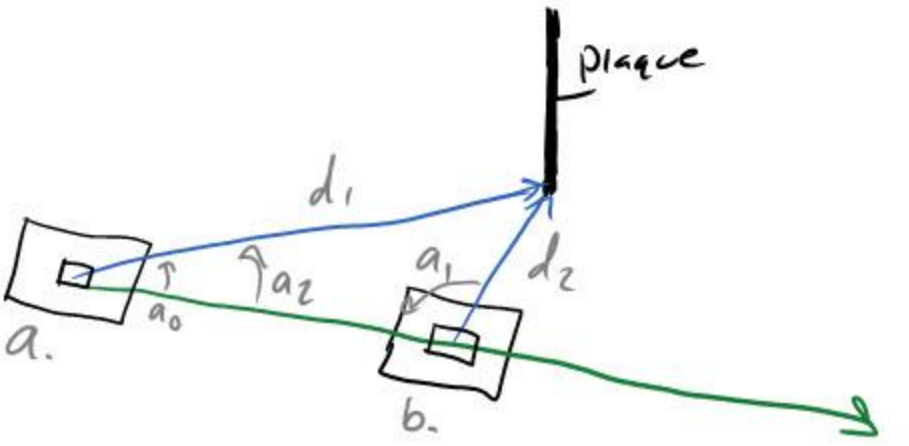
moving/rotating, it should sit at 0 degrees, but obviously the first graph it dosen't. When the gyro is rotated, we obtain these results:



Upon further investigation, we found out that we are not able to use the accelerometer as an input to the Kalman filter. This realization stemmed from the fact that the accelerometer senses changes in relative to Earth's gravitational force (such as pitch and roll for an airplane) but is not good at picking up changes perpendicular to gravity (such as the plane the rover is moving in). Kalman filter research was abandoned at this point.

IR Sensor as Input to Kalman Filter

In an attempt to revive the Kalman filter, Professor Hill suggested using the IR sensor as the second input (in addition to the gyro) for the Kalman filter. We did not have sufficient time to implement this idea but the theory follows.



In the above illustration:

- The box at point (a.) is the rover's initial position
- The box at point (b.) is the rover's current position
- Angle (a_0) is the initial measured angle (using the stepper motor's rotation)
- Angle (a_1) is the current measured angle (using the stepper motor's rotation)
- Angle (a_2) is the angle representing the rover's drift while traveling from point (a.) to point (b.). This is the drift between the expected path of travel and the actual path of travel. This angle is unknown and will be calculated.
- Distance (d_1) is the measured distance (using the IR ranger data) from the rover at point (a.) to the edge of the plaque
- Distance (d_2) is the measured distance (using the IR ranger data) from the rover at point (b.) to the edge of the plaque

The theory states that using the law of sines, we should be able to calculate angle (a_2) since all other values are measured. To calculate (a_2) we can use the following ratio:

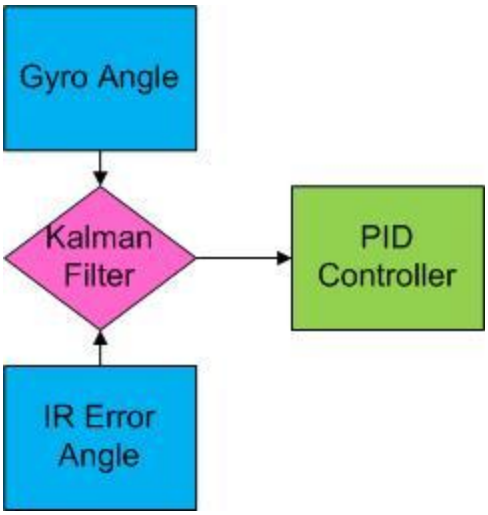
$$\frac{sin(a_1)}{d_1} = \frac{sin(a_2)}{d_2}$$

$$\Rightarrow (a_2) = sin^{-1}[\frac{d_2}{d_1} sin(a_1)]$$

The angle (a_2) is then subtracted from angle (a_0) to give the error signal:

$$IR\ Error\ angle = (a_0) - (a_2)$$

The IR Error angle signal is fed into the Kalman filter along with the Gyro's signal and the output will be a filtered error which is then fed into the PID controller as depicted:



Code

Sample Kalman Filter

(this MATLAB code generates simulated data and graphs it to display the potential effects of a Kalman filter. A version can be found in the Gyro Lab.)

```
% This program is modified from file tilt.c by Trammell Hudson
% <hudson@rotomotion.com> for 1D tilt sensor using a dual axis
% accelerometer and single axis angular rate gyro.  The two sensors
are
% fused using Kalman filter.
% Author: Ittichote Chuckpaiwong
% Last updated: 20 Feb 2007

close all;
clear;

% Simulate signal with noise
dt = 0.01;
t = (0:dt:50)';
x = sin(t); % Angle (original)
x_dot = cos(t); % Angular velocity
v_gyro = sqrt(0.3)*randn(length(t),1); % measurement noise for gyro,
variance = 0.3
gyro = x_dot + v_gyro; % Gyro measurement (pitch rate)
v_acc = sqrt(0.3)*randn(length(t),1); % measurement noise for
accellerometer, variance = 0.3
```

```
% Compute the angles computed by using only accelerometers of
gyroscope
x_acc = x + v_acc; % Angle computed from accelerometer measurement
(pitch angle)
x_gyro = cumsum(gyro*dt,1); % Angle computed by integration of gyro
measurement

% Plot the angles computed by using accelerometers or gyroscope alone
figure;
subplot(2,1,1);
plot(t,[x_acc x_gyro x]);
xlabel('time(s)');
ylabel('x(t)');
legend('accellerometer','gyro','actual');


% Following code is an implementation of Kalman filter.
% It is modify from source code downloaded from rotomotion.com
% which is originally written in C

%/*
% * Our covariance matrix.  This is updated at every time step to
% * determine how well the sensors are tracking the actual state.
% */
P = [1 0; 0 1];

%/*
% * R represents the measurement covariance noise.  In this case,
% * it is a 1x1 matrix that says that we expect 0.3 rad jitter
% * from the accelerometer.
% */
R_angle    = 0.3;

%/*
% * Q is a 2x2 matrix that represents the process covariance noise.
% * In this case, it indicates how much we trust the acceleromter
% * relative to the gyros.
% */
% Q_angle  = 0.001;
% Q_gyro   = 0.003;
Q_angle    = 0.05;
Q_gyro     = 0.5;
Q = [Q_angle 0; 0 Q_gyro];
```

```
%/*
% * state_update is called every dt with a biased gyro measurement
% * by the user of the module.  It updates the current angle and
% * rate estimate.
% *
% * The pitch gyro measurement should be scaled into real units, but
% * does not need any bias removal.  The filter will track the bias.
% *
% * Our state vector is:
% *
% *   X = [ angle, gyro_bias ]
% *
% * It runs the state estimation forward via the state functions:
% *
% *   Xdot = [ angle_dot, gyro_bias_dot ]
% *
% *   angle_dot  = gyro - gyro_bias
% *   gyro_bias_dot    = 0
% *
% * And updates the covariance matrix via the function:
% *
% *   Pdot = A*P + P*A' + Q
% *
% * A is the Jacobian of Xdot with respect to the states:
% *
% *   A = [ d(angle_dot)/d(angle)     d(angle_dot)/d(gyro_bias) ]
% *       [ d(gyro_bias_dot)/d(angle) d(gyro_bias_dot)/d(gyro_bias) ]
% *
% *     = [ 0 -1 ]
% *       [ 0  0 ]
% *
% * Due to the small CPU available on the microcontroller, we've
% * hand optimized the C code to only compute the terms that are
% * explicitly non-zero, as well as expanded out the matrix math
% * to be done in as few steps as possible.  This does make it harder
% * to read, debug and extend, but also allows us to do this with
% * very little CPU time.
% */
A = [0 -1; 0 0];

q_bias = 0; % Initialize gyro bias
angle = 0; % Initialize gyro angle
q_m = 0;
```

```
X = [0; 0];
x1 = zeros(size(t));
x2 = zeros(size(t));
for i=1:length(t),
   % ***************** Gyro update ******************
   %q_m = q_m + gyro(i)*dt; % Compute pitch angle from gyro
measurement
   q_m = gyro(i);

   q = q_m - q_bias; % /* Pitch gyro measurement */
   %     /* Unbias our gyro */
   % const float           q = q_m - q_bias;
   %
   % /*
   %  * Compute the derivative of the covariance matrix
   %  *
   %  *    Pdot = A*P + P*A' + Q
   %  *
   %  * We've hand computed the expansion of A = [ 0 -1, 0 0 ]
multiplied
   %  * by P and P multiplied by A' = [ 0 0, -1, 0 ].  This is then
added
   %  * to the diagonal elements of Q, which are Q_angle and Q_gyro.
   %  */
   Pdot = A*P + P*A' + Q;

   %/* Store our unbias gyro estimate */
     rate = q;

     %/*
     % * Update our angle estimate
     % * angle += angle_dot * dt
     % *       += (gyro - gyro_bias) * dt
     % *       += q * dt
     % */

   angle = angle + q*dt;

     %/* Update the covariance matrix */
   P = P + Pdot*dt;

   % ************ Kalman (Accelerometer) update *****

   % * The C matrix is a 1x2 (measurements x states) matrix that
```

```
   % * is the Jacobian matrix of the measurement value with respect
   % * to the states.  In this case, C is:
   % *
   % *     C = [ d(angle_m)/d(angle)  d(angle_m)/d(gyro_bias) ]
   % *       = [ 1 0 ]
   C = [1 0];
   angle_err = x_acc(i)-angle;

      %/*
      % * Compute the error estimate.  From the Kalman filter paper:
      % *
   %   *    E = C P C' + R
      % *
      % * Dimensionally,
      % *
      % *   E<1,1> = C<1,2> P<2,2> C'<2,1> + R<1,1>
      % *
      % * Again, note that C_1 is zero, so we do not compute the term.
      % */
   E = C*P*C' + R_angle;


      %/*
      % * Compute the Kalman filter gains.  From the Kalman paper:
      % *
      % *   K = P C' inv(E)
      % *
      % * Dimensionally:
      % *
      % *   K<2,1> = P<2,2> C'<2,1> inv(E)<1,1>
      % *
      % * Luckilly, E is <1,1>, so the inverse of E is just 1/E.
      % */
   K = P*C'*inv(E);

%     /*
%      * Update covariance matrix.  Again, from the Kalman filter
paper:
%      *
%      *    P = P - K C P
%      *
%      * Dimensionally:
%      *
%      *    P<2,2> -= K<2,1> C<1,2> P<2,2>
```

```
%      *
%      * We first compute t<1,2> = C P.  Note that:
%      *
%      *    t[0,0] = C[0,0] * P[0,0] + C[0,1] * P[1,0]
%      *
%      * But, since C_1 is zero, we have:
%      *
%      *    t[0,0] = C[0,0] * P[0,0] = PCt[0,0]
%      *
%      * This saves us a floating point multiply.
%      */
   P = P - K*C*P;

%     /*
%      * Update our state estimate.  Again, from the Kalman paper:
%      *
%      *    X += K * err
%      *
%      * And, dimensionally,
%      *
%      *    X<2> = X<2> + K<2,1> * err<1,1>
%      *
%      * err is a measurement of the difference in the measured state
%      * and the estimate state.  In our case, it is just the
difference
%      * between the two accelerometer measured angle and our estimated
%      * angle.
%      */
    X = X + K * angle_err;
    x1(i) = X(1);
    x2(i) = X(2);
    angle = x1(i);
    q_bias = x2(i);
end

% Plot the result using kalman filter
subplot(2,1,2);
plot(t,[x x1]);
xlabel('time(s)');
ylabel('x(t)');
legend('actual','kalman');
```

Working Kalman Filter

(this MATLAB code samples the gyro and runs the data through the above Kalman filter using simulated data for the second input.)

"getReading.m"

```
function [gyroData, accData] = getReading(duration, dt)
a=arduino('COM5');
gyroRead=(0:dt:duration)';
acc = (0:dt:duration)';
   for i=1:length(gyroRead)

       ...gyroRead(i)=a.analogRead(0);
           gyroRead(i)=(a.analogRead(0)-520)*(5/(1023*.022));
      ...acc(i)=gyroRead(i);
          acc(i)=0;
       pause(dt);

   end

    gyroData=gyroRead;
    accData=acc;
```

"tilt.m"

```
% This program is modified from file tilt.c by Trammell Hudson
% <hudson@rotomotion.com> for 1D tilt sensor using a dual axis
% accelerometer and single axis angular rate gyro.  The two sensors
are
% fused using Kalman filter.
% Author: Ittichote Chuckpaiwong
% Last updated: 20 Feb 2007


function [] = tilt(duration, dt)
% Simulate signal with noise
%dt = 0.01;
t = (0:dt:duration)';
x = sin(t); % Angle (original)
...x_dot = cos(t); % Angular velocity
...v_gyro = sqrt(0.3)*randn(length(t),1); % measurement noise for
gyro, variance = 0.3
```

```
...gyro = x_dot + v_gyro; % Gyro measurement (pitch rate)
...v_acc = sqrt(0.3)*randn(length(t),1); % measurement noise for
accellerometer, variance = 0.3

% Compute the angles computed by using only accelerometers of
gyroscope
...x_acc = x + v_acc; % Angle computed from accelerometer measurement
(pitch angle)
...x_gyro = cumsum(gyro*dt,1); % Angle computed by integration of gyro
measurement
[gyro, x_acc]=getReading(duration, dt);
x_gyro= cumsum(gyro*dt, 1);

% Plot the angles computed by using accelerometers or gyroscope alone
figure;
subplot(2,1,1);
plot(t,[x_acc x_gyro gyro]);
xlabel('time(s)');
ylabel('x(t)');
legend('accellerometer','gyro','actual');


% Following code is an implementation of Kalman filter.
% It is modify from source code downloaded from rotomotion.com
% which is originally written in C

%/*
% * Our covariance matrix.  This is updated at every time step to
% * determine how well the sensors are tracking the actual state.
% */
P = [1 0; 0 1];

%/*
% * R represents the measurement covariance noise.  In this case,
% * it is a 1x1 matrix that says that we expect 0.3 rad jitter
% * from the accelerometer.
% */
R_angle    = 0.3;

%/*
% * Q is a 2x2 matrix that represents the process covariance noise.
% * In this case, it indicates how much we trust the acceleromter
% * relative to the gyros.
% */
```

```
% Q_angle  = 0.001;
% Q_gyro   = 0.003;
Q_angle    = 0.05;
Q_gyro     = 0.5;
Q = [Q_angle 0; 0 Q_gyro];

%/*
% * state_update is called every dt with a biased gyro measurement
% * by the user of the module.  It updates the current angle and
% * rate estimate.
% *
% * The pitch gyro measurement should be scaled into real units, but
% * does not need any bias removal.  The filter will track the bias.
% *
% * Our state vector is:
% *
% *   X = [ angle, gyro_bias ]
% *
% * It runs the state estimation forward via the state functions:
% *
% *   Xdot = [ angle_dot, gyro_bias_dot ]
% *
% *   angle_dot  = gyro - gyro_bias
% *   gyro_bias_dot    = 0
% *
% * And updates the covariance matrix via the function:
% *
% *   Pdot = A*P + P*A' + Q
% *
% * A is the Jacobian of Xdot with respect to the states:
% *
% *   A = [ d(angle_dot)/d(angle)     d(angle_dot)/d(gyro_bias) ]
% *       [ d(gyro_bias_dot)/d(angle) d(gyro_bias_dot)/d(gyro_bias) ]
% *
% *     = [ 0 -1 ]
% *       [ 0  0 ]
% *
% * Due to the small CPU available on the microcontroller, we've
% * hand optimized the C code to only compute the terms that are
% * explicitly non-zero, as well as expanded out the matrix math
% * to be done in as few steps as possible.  This does make it harder
% * to read, debug and extend, but also allows us to do this with
% * very little CPU time.
% */
```

```
A = [0 -1; 0 0];

q_bias = 0; % Initialize gyro bias
angle = 0; % Initialize gyro angle
q_m = 0;
X = [0; 0];
x1 = zeros(size(t));
x2 = zeros(size(t));
for i=1:length(t),
   % *************** Gyro update ******************
   %q_m = q_m + gyro(i)*dt; % Compute pitch angle from gyro
measurement
   q_m = gyro(i);

   q = q_m - q_bias; % /* Pitch gyro measurement */
   %     /* Unbias our gyro */
   % const float           q = q_m - q_bias;
   %
   % /*
   %  * Compute the derivative of the covariance matrix
   %  *
   %  *    Pdot = A*P + P*A' + Q
   %  *
   %  * We've hand computed the expansion of A = [ 0 -1, 0 0 ]
multiplied
   %  * by P and P multiplied by A' = [ 0 0, -1, 0 ].  This is then
added
   %  * to the diagonal elements of Q, which are Q_angle and Q_gyro.
   %  */
   Pdot = A*P + P*A' + Q;

   %/* Store our unbias gyro estimate */
     rate = q;

     %/*
     % * Update our angle estimate
     % * angle += angle_dot * dt
     % *       += (gyro - gyro_bias) * dt
     % *       += q * dt
     % */

   angle = angle + q*dt;

     %/* Update the covariance matrix */
```

```
   P = P + Pdot*dt;

   % ************ Kalman (Accelerometer) update *****

   % * The C matrix is a 1x2 (measurements x states) matrix that
   % * is the Jacobian matrix of the measurement value with respect
   % * to the states.  In this case, C is:
   % *
   % *     C = [ d(angle_m)/d(angle)  d(angle_m)/d(gyro_bias) ]
   % *       = [ 1 0 ]
   C = [1 0];
   angle_err = x_acc(i)-angle;

      %/*
      % * Compute the error estimate.  From the Kalman filter paper:
      % *
   %   *    E = C P C' + R
      % *
      % * Dimensionally,
      % *
      % *   E<1,1> = C<1,2> P<2,2> C'<2,1> + R<1,1>
      % *
      % * Again, note that C_1 is zero, so we do not compute the term.
      % */
   E = C*P*C' + R_angle;


      %/*
      % * Compute the Kalman filter gains.  From the Kalman paper:
      % *
      % *   K = P C' inv(E)
      % *
      % * Dimensionally:
      % *
      % *   K<2,1> = P<2,2> C'<2,1> inv(E)<1,1>
      % *
      % * Luckilly, E is <1,1>, so the inverse of E is just 1/E.
      % */
   K = P*C'*inv(E);

%     /*
%      * Update covariance matrix.  Again, from the Kalman filter
paper:
%      *
```

```
%      *    P = P - K C P
%      *
%      * Dimensionally:
%      *
%      *    P<2,2> -= K<2,1> C<1,2> P<2,2>
%      *
%      * We first compute t<1,2> = C P.  Note that:
%      *
%      *    t[0,0] = C[0,0] * P[0,0] + C[0,1] * P[1,0]
%      *
%      * But, since C_1 is zero, we have:
%      *
%      *    t[0,0] = C[0,0] * P[0,0] = PCt[0,0]
%      *
%      * This saves us a floating point multiply.
%      */
   P = P - K*C*P;

%     /*
%      * Update our state estimate.  Again, from the Kalman paper:
%      *
%      *    X += K * err
%      *
%      * And, dimensionally,
%      *
%      *    X<2> = X<2> + K<2,1> * err<1,1>
%      *
%      * err is a measurement of the difference in the measured state
%      * and the estimate state.  In our case, it is just the
difference
%      * between the two accelerometer measured angle and our estimated
%      * angle.
%      */
    X = X + K * angle_err;
    x1(i) = X(1);
    x2(i) = X(2);
    angle = x1(i);
    q_bias = x2(i);
end

% Plot the result using kalman filter
xlswrite('done2.xls', [t gyro x1])
subplot(2,1,2);
plot(t,[gyro x1]);
```

```
xlabel('time(s)');
ylabel('x(t)');
legend('actual','kalman');
```

Sources/Resources

1. Kalman online Java tool http://www.cs.unc.edu/~welch/kalman/kftool/
2. Kalman Site: http://www.cs.unc.edu/~welch/kalman/
3. Airplanes:
   Accelerometer: http://tom.pycke.be/mav/69/accelerometer-to-attitude
   Gyros: http://tom.pycke.be/mav/70/gyroscope-to-roll-pitch-and-yaw
   Kalman: http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data
4. Summary: http://www.physicsforums.com/showthread.php?t=261788
5. Here is a comparison between the raw accelerator angle (red) and the Kalman filtered angle (blue) http://www.youtube.com/v/H5Drlqv1t3s? (from: http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1282384853/0)
6. Tilt code http://www.rcgroups.com/forums/showthread.php?t=858421
7. http://forum.sparkfun.com/viewtopic.php?t=6186
8. APPLICATION OF KALMAN FILTERING AND PID CONTROL FOR.pdf (\My Dropbox\Spring 2011\EE444\kalman filter)
9. Control Tutorials for MATLAB: http://www.engin.umich.edu/class/ctms/

· http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1282384853/0