IMU modules, AHRS and a Kalman filter for sensor fusion

2016 September 20, Hari Nair, Bangalore

This document describes how I built and used an Inertial Measurement Unit (IMU) module for Attitude & Heading Reference System (AHRS) applications. It also describes the use of AHRS and a Kalman filter to fuse data from a barometric pressure sensor and IMU sensors for tracking altitude and vertical velocity for applications like paragliding and RC model altimeter/variometers.

Sensor Choices

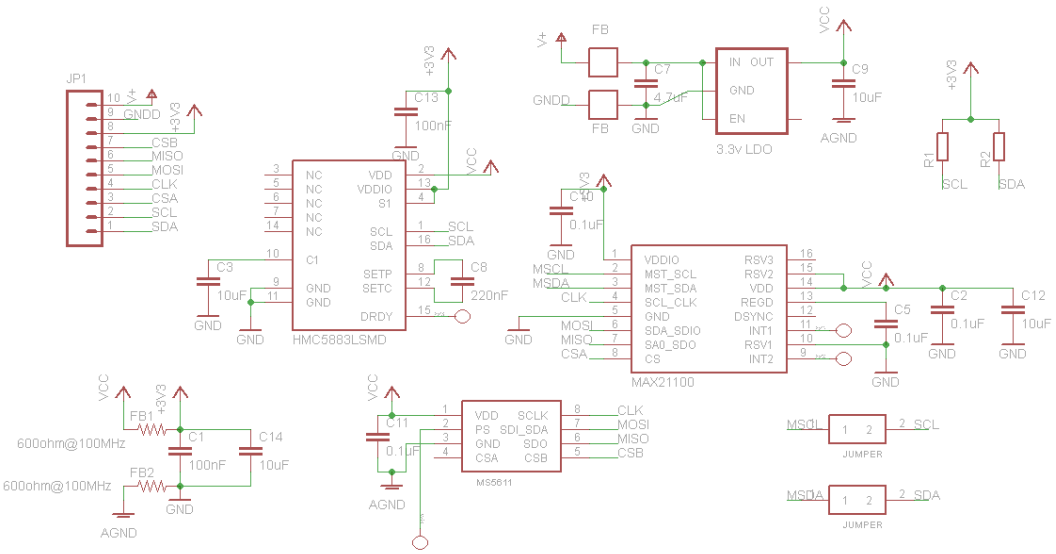
I decided to design my own sensor breakout board using the Maxim MAX21100 3-axis accelerometer + 3-axis gyroscope, Honeywell HMC5883L 3-axis magnetometer, and MEAS MS5611 barometric pressure sensor.

The MAX21100 has rather low gyroscope noise and drift specifications – this is significant for our application. And I got free samples :-).

The HMC5883L magnetometer is cheap, easily available and was recommended by the Maxim support staff.

The MS5611 barometric pressure sensor isn't the cheapest option, but it is now readily available and has superior resolution and noise specifications compared to say the BMP180. Plus we have the choice of using SPI or I2C interface.

I was planning on an SPI interface for a higher data read rate, so that helped. The MAX21100 has a master I2C interface to the HMC5883L, so accelerometer, gyroscope, and magnetometer samples can be read via the MAX21100 SPI interface.



The sensor board has an SPI interface with separate chip selects for the MS5611 and MAX21100. The I2C interfaces can also be used. The HMC5883L magnetometer has only an I2C interface. After verifying that the magnetometer was functioning using the I2C interface, I managed to get the MAX21100 master I2C interface to the HMC5883L working. This means that the magnetometer can be configured and read via the MAX21100 SPI interface. In this case, the schematic SCL and SDA pullup resistors can be omitted as the MAX21100 has internal I2C pullup resistors that can be enabled during configuration.

The ferrite beads are 600ohm@100MHz type in 0603 package. These are used to isolate analog components from digital switching noise on the VCC and ground planes. Higher impedance values can be used as long as the dc resistance is not too high, as the sensors draw little current. The boards were ordered from OSHPark.

I later received samples of the MAX21105, and it turned out to be pin-compatible with the MAX21100 layout, and with the same low gyro noise and drift specifications. So I could use the same pcb board for the MAX21105. However, it's not possible to read the magnetometer data via the MAX21105, so an I2C interface to the board is required. Or you could use SPI for the MAX21105 and I2C for the HMC5883L at the cost of more interface pins. The schematic and board accommodates these configurations.

Today, I would probably just order an IMU sensor module from ebay, e.g. the GY-91 with an MPU9250 accel+gyro+mag and BMP280 pressure sensor. As of date, the cost is ~ US$6 including shipping world-wide and the module has decent specifications.

Sensor Axis Mapping for AHRS

The datasheets for the MAX21100 and HMC5883L show a right-handed sensor coordinate axis frame with Z pointing up. When designing my sensor module PCB, I took care to place these two components with their X, Y and Z axes aligned in the same direction. However, this is not an issue, as long as the MAX21100 and HMC5883L coordinate axes are parallel. I placed a silkscreen graphic on the board with the sensor frame X-Y axes marked for reference.

Once we decide on the AHRS coordinate frame convention and what we consider to be the "forward" pointing direction of the sensor board, the individual sensor axes and signs are mapped in software to be consistent with our convention.

We can use East (X)-North(Y)-Up (Z), or North(X)-East(Y)-Down(Z) frames. Both are right-hand coordinate frames.

I decided to use the N-E-D frame, also described as an "Aerospace coordinate frame", because the literature favors this, and the applications I'm interested in are aviation related – paragliding, RC flight models.

Mark Pedley wrote an article in the August 2012 edition of Circuit Cellar, which describes how to use a 3-axis magnetometer and 3-axis accelerometer for a 3D-compass. In this article he also spells out the required steps for mapping the sensor axes/signs for consistency with the aerospace frame convention.

Here is a summary (adding the steps for a 3-axis gyroscope) :

Select the sensor board "front" (N) direction, with the (E) axis parallel to the board surface pointing to the right. Then the (D) axis points downwards through the board to complete our NED frame.

Write some test code that continuously samples the accelerometer, gyroscope and magnetometer several times a second. Display the raw signed integer X, Y and Z samples so you can monitor them as you move the board. I found it easier to just enable one of the sensors at a time – accelerometer, gyroscope or magnetometer.

GYROSCOPE Axis Mapping

positive pitch = rotation up about the board E axis

positive roll = rotation right about the board N axis

positive yaw = rotation clockwise about the board D axis

Rotate the board around the N, E and D axes while monitoring the gyroscope data samples. Map the gyroscope X, Y and Z axes and signs in software to be consistent with our pitch, yaw and roll sign conventions.

ACCELEROMETER Axis Mapping

Place the sensor board on a horizontal surface. The board D axis points straight down and the accelerometer Z axis value is equivalent to +1g. The values for the N and E axes are readings for 0 (zero) g. Here $1\,g = 980\,cm/s^2$, this is the static acceleration due to earth's gravity.

With the board N axis pointing vertically down, the accelerometer X axis value should be +1g.

When the board E axis points vertically down, the accelerometer Y axis value should be +1g.

So this tells us how to map the accelerometer sensor axes and signs to the NED frame.

MAGNETOMETER Axis Mapping

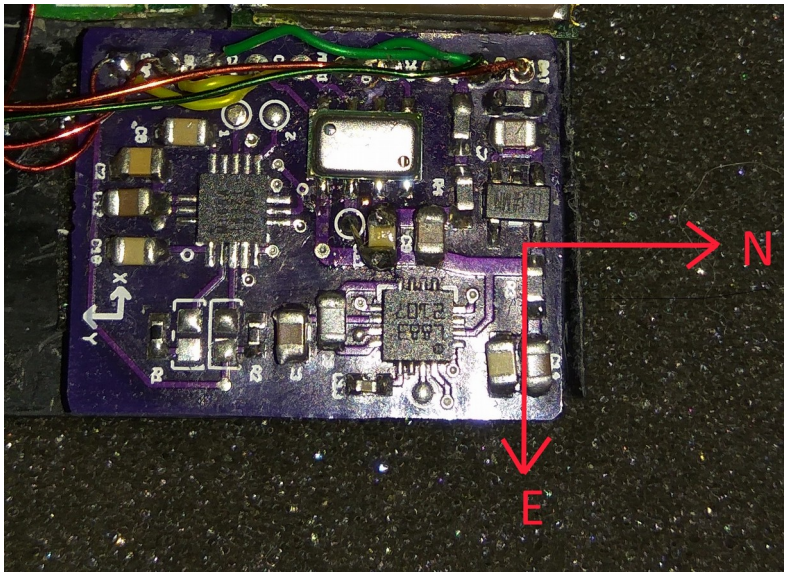
Use a compass to find out where magnetic north is. Make sure there are no magnetic materials near the sensor board. Place the board horizontal and rotate it slowly in the yaw axis (around the D axis). The magnetometer X value will be maximum when the board "front" or N axis points towards the magnetic North pole, and minimum when the board N axis points towards the magnetic South pole.

The magnetometer Y value will be minimum when the board N axis points East, and maximum when the board N axis points West.

Rotate the board so that the board D axis is roughly horizontal. Then turn the board keeping the D axis horizontal, the magnetometer Z axis value will be maximum when pointing to magnetic north.

This tells us how to map the magnetometer sensor axes and signs.

Here's a picture of my homebrew IMU sensor board in a gps-vario module for a remote-control (RC) plane application. This has a MAX21105 accelerometer+gyroscope, HMC5883L magnetometer, and MS5611 pressure sensor, with an I2C interface to the board. I've annotated in red the AHRS frame N(X) axis that points towards the front of the plane, and E(Y) axis pointing to the right. The D(Z) axis points down through the board. The PCB silkscreen marking for the sensor coordinate frame is on the left edge of the board, with Z axis pointing up from the board.



For this board, and this application, these are the software mappings from the IMU sensor coordinate frames to the AHRS NED coordinate frame:

Accelerometer

$$acc[x]_{NED} = acc[y]$$

$$acc[y]_{NED} = acc[x]$$

$$acc[z]_{NED} = acc[z]$$

Gyroscope

$$gyro[x]_{NED} = -gyro[y]$$

$$gyro[y]_{NED} = -gyro[x]$$

$$gyro[z]_{NED} = -gyro[z]$$

Magnetometer

$$mag[x]_{NED} = -mag[y]$$

$$mag[y]_{NED} = -mag[x]$$

$$mag[z]_{NED} = -mag[z]$$

Sensor Configuration

Use the minimum full scale axis configuration that will encompass the possible envelope of real-life data, so that you retain maximum possible resolution.

Use the highest sample output data rate (ODR) that your processing power allows. For example, with a 250Hz sample rate, you only have 4 milliseconds to read and process the data and then do whatever control/display operations are required. This is where the SPI interface helps as the sensor SPI clocks can go up to several MHz, while I2C clocks are often limited to 400kHz.

For a paragliding gps-vario application with the MAX21100, I've used

Gyroscope : Full scale +/-500dps, Bandwidth 14Hz , 250Hz ODR

Accelerometer : Full scale +/-4G, LPF = ODR/48 (maximum low pass filtering), 250Hz ODR

Magnetometer : Full scale +/- 1.3 Gauss, 75Hz ODR. This is the maximum data rate supported by the HMC5883L.

For an RC gps-vario application with the MAX21105, I've used

Gyroscope : Full scale +/-1000dps, 10Hz BW, 200Hz ODR

Accelerometer : Full scale +/-8G, LPF = ODR/48 (maximum low pass filtering), 200Hz ODR

Magnetometer : Full scale +/- 1.3 Gauss, 75Hz ODR

Sensor Calibration

The calibration parameters of interest are axis offsets and axis sensitivities.

An offset (bias) is the non-zero value that is read when the expected value should be zero.

Sensitivity gives us the scale factor mapping the sensor sample values to real-world units.

$$real\,value = (sample - offset) * scale \quad \text{where} \quad scale = \frac{1}{sensitivity}$$

Gyroscope

The gyroscope axis offsets are calibrated by averaging the gyroscope axis readings with the unit at rest. This should ideally be done each time on power up, with a power-on delay and some sort of indication to the user so that the unit can be undisturbed when the calibration is going on. Averaging 100 samples will take less than half a second at 250Hz ODR, so this is not a big issue. If a larger than "expected" reading is seen on any axis, the software should wait a couple of seconds and repeat the sampling.

I took the sensitivity value from the datasheet. It's the same for all 3 axes. For example, the MAX21100 datasheet specifies the gyro rate sensitivity as 60 counts per degree-per-second when full scale is configured as +/- 500 degrees per second.

Accelerometer

The nominal axis sensitivities are specified in the datasheet, but since it is easy enough to calibrate, I use calibrated values instead of the datasheet values.

1. Place the sensor board on a horizontal surface. Continuously sample the accelerometer axis values, and display averaged readings to reduce noise. The sensor X and sensor Y axis readings are the zero-g offsets x_{offset} and y_{offset}, the sensor Z reading z_{+1} is equivalent to +1g.

Turn the board upside down to get the sensor Z reading z_{-1} equivalent to -1g.

2. Place the unit with sensor X axis pointing down, get sensor X reading x_{+1} for +1g. Repeat with sensor X axis pointing vertically up, to get the X reading x_{-1} for -1g.

3. Repeat step 2 with the sensor Y axis to get the sensor Y readings y_{+1} and y_{-1} for +1g and -1g.

$$z_{sensitivity} = \frac{(z_{+1} - z_{-1})}{2} counts/g$$

$$y_{sensitivity} = \frac{(y_{+1} - y_{-1})}{2} counts/g$$

$$x_{sensitivity} = \frac{(x_{+1} - x_{-1})}{2} counts/g$$

$$z_{offset} = z_{+1} - z_{sensitivity}$$

Accelerometer calibration needs to be done only once after the unit is completely assembled, to take into account any stresses on the accelerometer via the PCB and enclosure mounting. It helps to have a rectangular case.

Magnetometer

Continuously sample the magnetometer readings while slowly and smoothly rotating the unit through a figure of 8 motion in three dimensions. Monitor the minimum and maximum readings on all three axes. Repeat the motion several times to ensure all possible orientations of the board are sampled.

$$axis_{sensitivity} = \frac{(axis_{max} - axis_{min})}{2}$$

$$axis_{offset} = \frac{(axis_{max} + axis_{min})}{2}$$

Repeat the procedure a few times to make sure you are getting consistent results, and ensure you are far away from nearby steel objects, magnets etc.

This is also called "hard-iron" calibration. This needs to be done after final assembly because of the effects of nearby circuitry with dc currents, metal connectors, etc. in the enclosure.

Obtaining AHRS measurements from an IMU sensor

The AHRS parameters of interest are yaw, pitch and roll in the NED frame. Yaw is computed as the angle in degrees [-180, 180] from magnetic north, with positive yaw being a clockwise rotation around the D axis. Pitch is computed as the angle in degrees [-90,90] rotated up or down around the E axis, with positive values corresponding to a pitch up. Roll is computed as the angle in degrees [-90,90] rotated around the N axis, with positive values corresponding to a roll to the right.

I used the Madgwick algorithm reference C code for computing the quaternion representing the orientation vector in the NED frame, and standard quaternion-to-angle formulae for computing the orientation in Yaw-Pitch-Roll parameters.

In the Madgwick AHRS algorithm, the gyroscope rotation rate data is integrated to update the quaternion orientation, while the accelerometer and magnetometer are used to provide corrections. Gyroscope data is low-noise and high sensitivity, but the noise is characterized as a random walk, not zero-mean additive noise.

Accelerometer data is noisier and has lower sensitivity, but the noise is additive and zero-mean. So the accelerometer data can be low pass filtered for long-term (relative to the sample ODR) corrections. This is why we configure the accelerometer sensor for maximum internal low pass filtering.

The accelerometer readings can give you orientation accurately when the sensor is in static acceleration due to earth's gravity. However, the sensor can experience linear acceleration, centripetal forces, etc. in real-world applications. So I've tweaked the AHRS reference code. I added a flag to allow the AHRS algorithm to use the accelerometer data only when the acceleration vector magnitude is between 0.6g and 1.4g. In low-g or high-g environments, the code discards the accelerometer data and just integrates the gyroscope rotation-rate data to update the orientation. This is why it's important to use a gyroscope sensor with low gyro noise and drift, and that's where the MAX21100/21105 specification is excellent.

Using a Kalman filter to fuse data from an AHRS module and a barometric pressure sensor

Problem Formulation : We need to estimate altitude and vertical velocity (climb or sink rate) for applications such as paragliding or RC model altimeter-variometers.

This problem can be modeled as the tracking of position and velocity of a particle moving on a linear axis with random acceleration perturbations.

In our application the linear axis we are interested in is the vertical axis (altitude) and the velocity is the climbrate/sinkrate. The random acceleration perturbations along the vertical axis come from changes in the air (thermals, sinking air, turbulence) and pilot input.

This Wikipedia page http://en.wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical

has a great explanation of a Kalman filter that does just this, given only position data samples. In our real-world application, the position data samples would be the altitude values derived from a barometric pressure sensor.

This works well enough, but the response is a bit laggy. You end up with a trade-off between variometer response delay and noise.

How can we improve on this?

Let's assume we have altitude data from a barometric pressure sensor AND vertical acceleration data from an AHRS module.

Our new Kalman filter fuses the data from the two sources to estimate and track altitude and climbrate.

From my reading and experiments, it became clear that the filter would have to estimate and track not just the desired parameters (altitude and climbrate), but also any systematic unknown offsets with unpredictable drift.

So apart from altitude and climbrate, we introduce another parameter - the offset or bias in the acceleration value measured by the accelerometer. For example, the accelerometer after calibration may measure 0.99g for vertical acceleration when the unit is at rest. This is the static acceleration due to earth's gravity and should read 1g at rest. In this case, the accelerometer bias = -0.01g. This bias is an unknown parameter and can drift unpredictably.

We now have now three Kalman *state variables* :

z_k = altitude

$\dot{z}_k$ = vertical velocity

$\ddot{z} b_k$ = acceleration bias

where k is the time sample index.

The true acceleration is the measured acceleration minus the acceleration bias $z\ddot{}_k - z\ddot{} b_k$.

These three parameters at sample time index k are related to the samples at the previous time index k-1 by the following physical equations

$$z_k = z_{k-1} + dt * z\dot{}_{k-1} + \frac{dt^2}{2} * (z\ddot{}_{k-1} - z\ddot{} b_{k-1})$$

$$z\dot{}_k = z\dot{}_{k-1} + dt * (z\ddot{}_{k-1} - z\ddot{} b_{k-1})$$

$$z\ddot{} b_k = z\ddot{} b_{k-1} + N_b$$

where dt is the time interval between the two samples k and k-1, and N_b is a random variable representing accelerometer bias noise. In vector/matrix notation, the state transition equations can be written as

$$\begin{bmatrix} z_k \\ z\dot{}_k \\ z\ddot{} b_k \end{bmatrix} = \begin{bmatrix} 1 & dt & \frac{-dt^2}{2} \\ 0 & 1 & -dt \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} z_{k-1} \\ z\dot{}_{k-1} \\ z\ddot{} b_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{dt^2}{2} z\ddot{}_{k-1} \\ dt \,.\, z\ddot{}_{k-1} \\ N_b \end{bmatrix}$$

Or

$$x_k = \mathbf{F}\ x_{k-1} + w_k$$

where our state vector $x_k = \begin{bmatrix} z_k \\ z\dot{}_k \\ z\ddot{} b_k \end{bmatrix}$, **F** is the state transition matrix and w_k represents the process noise or perturbation of the model from the environment.

Define **Q** = Process Noise Covariance Matrix = E($w_k w_k^T$)

$$= E\left(\begin{bmatrix} \frac{dt^2}{2} * z_{k-1} \\ dt * z\dot{}_{k-1} \\ N_b \end{bmatrix} \begin{bmatrix} \frac{dt^2}{2} * z_{k-1} & dt * z\dot{}_{k-1} & N_b \end{bmatrix} \right)$$

Assuming the external perturbation can be described by additive zero-mean noise,

Q = $$\begin{bmatrix} \frac{dt^4}{4}\sigma_{acc}^2 & \frac{dt^3}{2}\sigma_{acc}^2 & 0 \\ \frac{dt^3}{2}\sigma_{acc}^2 & dt^2\sigma_{acc}^2 & 0 \\ 0 & 0 & \sigma_{accbias}^2 \end{bmatrix}$$

σ_{acc}^2 is the variance of the external acceleration inputs from the environment. In our real-world application, this value will depend on factors such as the amount of air turbulence, thermal strength etc.

$\sigma_{accbias}^2$ is the variance of the accelerometer bias noise.

Define the state covariance matrix **P**

P = $$\begin{bmatrix} p_{zz} & p_{zv} & p_{za} \\ p_{vz} & p_{vv} & p_{va} \\ p_{az} & p_{av} & p_{aa} \end{bmatrix}$$

The IMU output data rate (200Hz- 250Hz in our configuration) is much higher than the pressure sensor data rate from the MS5611 (30Hz - 40Hz).

Low pass filtering and down-sampling the acceleration data gives us an acceleration data sample corresponding to each new altitude sample. Note that the Kalman filter is updated at the pressure sensor sample rate, while the AHRS Madgwick code is executed at the IMU output data rate.

Prediction

1. Compute a priori (predicted) Covariance Matrix

$$P_{k/k-1} = F P_{k-1/k-1} F^T + Q$$

2. Compute a priori (predicted) state

$$accel_{real} = accel_{meas} - \ddot{z} b_{k-1}$$

$$\dot{z}_{k-1} = \dot{z}_{k-1} + accel_{real} * dt$$

$$z_{k-1} = z_{k-1} + \dot{z}_{k-1} * dt$$

Update

The observation (measurement) variable used for feedback is the current altitude sample from the barometric pressure sensor = m_k .

$$m_k = H x_k^o + v_k$$

$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ is the matrix used to extract the equivalent variable from the state vector. This results in the following scalar equation:

$$m_k = z_k^o + N_z$$

where N_z is a random scalar variable representing altitude measurement noise.

3. Compute Innovation (error) = difference between measured altitude from barometric sensor and predicted altitude from previous step

$$y_k = m_k - H x_{k/k-1}$$

$$y_k = \begin{bmatrix} z^o_k \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_{k-1} \\ \dot{z}_{k-1} \\ \ddot{b}_{k-1} \end{bmatrix}$$

resulting in the scalar equation

$$y_k = z_k^o - z_{k-1}$$

4. Compute Innovation Covariance $S_k = H P_{k/k-1} H^T + R$

Because H only picks out one scalar component, R is also a scalar = σ^2_{zmeas} = altitude (z) measurement noise variance. This can be estimated offline or during barometric pressure sensor initialization.

Similarly, S_k is also a scalar = s_k

$$s_k = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} P_{k/k-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \sigma^2_{zmeas}$$

$$= p_{zz} + \sigma^2_{zmeas}$$

5. Compute Kalman gain $K_k = \begin{bmatrix} k_z \\ k_v \\ k_a \end{bmatrix}$

$$= P_{k/k-1} H^T S_k^{-1}$$

$$= \quad P_{k/k-1}\begin{bmatrix}1\\0\\0\end{bmatrix} \qquad \frac{1}{p_{zz}+\sigma^2_{zmeas}}$$

$$= \quad \begin{bmatrix}p_{zz}\\p_{vz}\\p_{az}\end{bmatrix} \qquad \frac{1}{p_{zz}+\sigma^2_{zmeas}}$$

6. Compute a posteriori (update) estimate of current state

$$x_{k/k} \quad = \quad x_{k/k-1} \quad + \quad K_k\, y_k$$

$$\begin{bmatrix}z_k\\ \dot{z}_k\\ \ddot{z}\, b_k\end{bmatrix} = \begin{bmatrix}z_{k-1}\\ \dot{z}_{k-1}\\ \ddot{z}\, b_{k-1}\end{bmatrix} + \begin{bmatrix}k_z\, y_k\\ k_v\, y_k\\ k_a\, y_k\end{bmatrix}$$

Note: this is the Kalman filter output for this iteration.

7. Compute a posteriori (update) covariance matrix

$$P_{k/k} \quad = \quad \left(I - K_k H\right) P_{k/k-1}$$

$$= \quad P_{k/k-1} - \begin{bmatrix}k_z\, p_{zz} & k_z\, p_{zv} & k_z\, p_{za}\\ k_v\, p_{zz} & k_v\, p_{zv} & k_v\, p_{za}\\ k_a\, p_{zz} & k_a\, p_{zv} & k_a\, p_{za}\end{bmatrix}$$

And repeat ad nauseam, for sparkly clean (or so we hope) altitude and climbrate data.

Configuring the Kalman Filter response

σ^2_{zmeas} can be determined by computing the statistical variance of about 1-2 seconds of altitude data derived from the pressure sensor, using the normal sampling rate. Do not use longer intervals, as barometric pressure can easily change enough to affect this reading.

σ^2_{acc} Is the variance of the acceleration induced noise from the environment – thermals, sinking air, turbulence etc. It is NOT the accelerometer noise variance ! σ^2_{acc} is best determined experimentally for the application and could be configured dynamically, to match the flying conditions for the day.

σ^2_{acc} and σ^2_{zmeas} can be used to shape the response of the filter to personal taste. I've had decent results by starting with σ^2_{acc} values numerically approximately equal to the σ^2_{zmeas} values.

Using larger values of σ_{acc}^2 will result in the algorithm responding faster to acceleration inputs, smaller values will result in a more damped response to acceleration inputs.

Using a σ_{zmeas}^2 value smaller than the actual sensor altitude noise variance will result in the algorithm responding faster to changes in pressure sensor data, at the cost of more jitter. Using larger values will result in a more damped response.

Since the accelerometer bias noise is expected to be low and change slowly, $\sigma_{accbias}^2$ can be set to a low value, e.g. 1.0, to enforce this. The trade-off is a longer startup delay in the algorithm estimating an accurate value of the acceleration bias. The real-world symptom for our variometer application would be the unit beeping for several seconds after power-on even with the unit at rest.

Computing Gravity Compensated Acceleration

We have altitude data derived from the barometric pressure sensor, but we also need vertical acceleration data for the Kalman filter.

The magnitude of the acceleration data vector from the accelerometer is

$$||a|| = \sqrt{a_x * a_x + a_y * a_y + a_z * a_z}$$

With the unit at rest, the acceleration magnitude should be 1g for any orientation of the sensor, as this is the static acceleration due to earth's gravity.

$$||a|| = 1$$

This is in the **sensor frame**. The net dynamic acceleration of the sensor in **earth frame** is 0.

Now let's assume the sensor is not at rest, but in vertical acceleration. The net acceleration in g's in **earth frame** is

$$||a|| - 1$$

However, there are situations where the g forces are not acting vertically on the unit, e.g. when banking in a turn. So what we really need is the vertical component of the dynamic acceleration, independent of the orientation of the unit.

The AHRS module gives us a quaternion representing the orientation of the unit in the earth frame.

$$q = \begin{bmatrix} q0 \\ q1 \\ q2 \\ q3 \end{bmatrix}$$

The accelerometer data vector gives us the acceleration vector in the sensor frame.

$$a_{sensorframe} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

What we need to do is rotate the sensor acceleration vector by the orientation quaternion to get the acceleration vector in earth frame.

$$a_{earthframe} = Rotate(a_{sensorframe}, q)$$

$$= q\, a_{sensorframe}\, q^{-1}$$

$$= \begin{bmatrix} q0^2+q1^2-q2^2-q3^2 & 2q1q2-2q0q3 & 2q1q3+2q0q2 \\ 2q1q2+2q0q3 & q0^2-q1^2+q2^2-q3^2 & 2q2q3-2q0q1 \\ 2q1q3-2q0q2 & 2q2q3+2q0q1 & q0^2-q1^2-q2^2+q3^2 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

The vertical (z) component of the acceleration is

$$az_{earthframe} = (2q1q3-2q0q2)*a_x + (2q2q3+2q0q1)*a_y + (q0^2-q1^2-q2^2+q3^2)*a_z$$

In our NED coordinate frame, the Z axis points down, so the static acceleration vector due to gravity in

earth frame $g = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Subtracting this vector from the earth frame acceleration vector gives us the gravity compensated net dynamic acceleration in earth frame.

$$a_{earthframe_{net}} = a_{earthframe} - g$$

And we are only interested in the vertical (z) component.

$az_{earthframe_{net}}$ = $az_{earthframe} - 1$ (in g's)

$$az_{earthframe_{net}} = (az_{earthframe} - 1) * 980\, cm/s^2$$

This is the acceleration data that we need to supply to our Kalman filter, along with the altitude value (in cm) derived from the barometric pressure sensor.

References

http://cache.freescale.com/files/sensors/doc/reports_presentations/ARTICLE_REPRINT.pdf

http://x-io.co.uk/open-source-imu-and-ahrs-algorithms/

http://campar.in.tum.de/Chair/KalmanFilter

http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/

http://en.wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical