2014 October 17

Hari Nair, Bangalore

Here are some notes on

1. using sensors for AHRS applications

2. developing a Kalman filter based sensor fusion for tracking altitude and climb-rate(sink-rate) for a recreational paragliding altimeter/variometer.

Sensor Choices

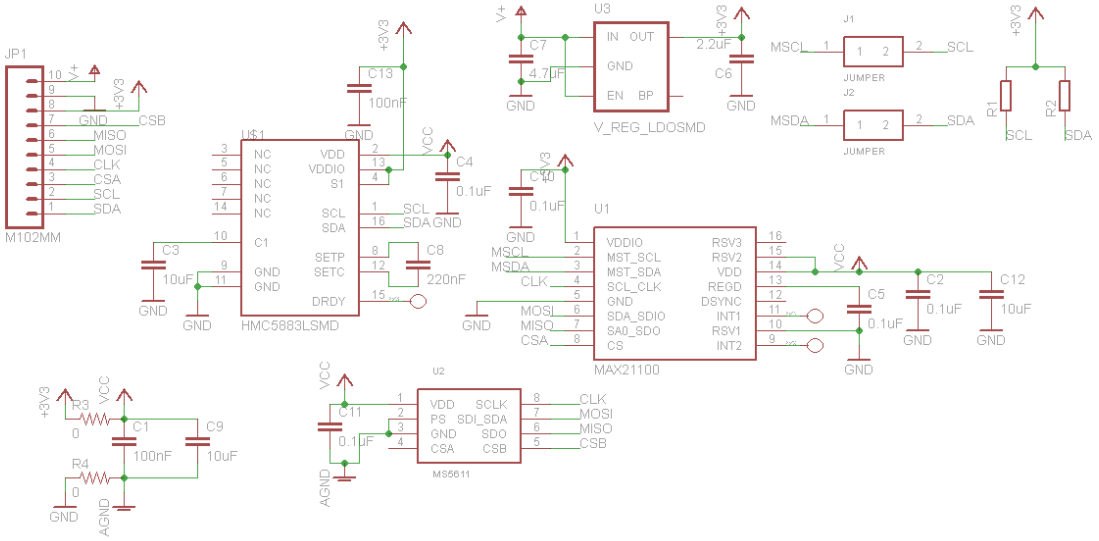
I decided to design my own sensor breakout board using the Maxim MAX21100 3-axis accelerometer + 3-axis gyroscope, Honeywell HMC5883L 3-axis magnetometer, and MEAS MS5611 barometric pressure sensor.

The MAX21100 seems to have rather good gyroscope noise and drift specifications and the possibility of internal 9DOF sensor fusion and AHRS (Attitude & Heading Reference System) estimation. This would free up the application microcontroller for other compute-intensive tasks, or allow the use of an inexpensive microcontroller. Free samples also helped in making this choice :-). The current versions of the MAX21100 datasheet and register programming documents aren't that informative, but Maxim support staff answered most of my queries. They still haven't released all the documentation about their internal 9DOF fusion, so that remains to be investigated.

The HMC5883L magnetometer is cheap, easily available and fairly standard, and was recommended by the Maxim support staff. This connects to the master I2C interface of the MAX21100.

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

I was planning on an SPI interface, so that helped.



The sensor board has an SPI interface with separate chip selects for the MS5611 and MAX21100. I also brought out I2C bus pins to the HMC5883L, but after verifying that the magnetometer was functioning, I managed to get the MAX21100 I2C interface to the HMC5883L working. This means that the magnetometer can be configured and read via the MAX21100 SPI interface.

The MAX21100 has internal 6K I2C pullup resistors, so the external resistors R1 and R2 can be omitted. R3 and R4 are ferrite chip isolators, I used 600ohm@100MHz 0603 package variants. In retrospect, I should have taken the analog supply and ground directly from the LDO regulator output as that would be the cleanest source.

Sensor Mapping for AHRS

The datasheets for the MAX21100 and HMC5883L show a right-handed local sensor coordinate axis frame with Z pointing up. When designing the PCB, I took care to place these two components so that their X, Y and Z axes were aligned, and placed a silkscreen graphic with the XY axes marked for reference. However, as we will see, this is really not an issue (as long as the sensor coordinate axes are parallel). Once we decide on the IMU AHRS coordinate system convention and what we consider to be the "forward" pointing direction of the sensor board, the individual sensor axes and signs are simply mapped in software to be consistent with these decisions.

You could go with East (X)-North(Y)-Up (Z). Or North(X)-East(Y)-Down(Z). Both are right-hand coordinate frames. The N-E-D convention represents the so-called "Aerospace coordinate frame". I went with this. Paragliding is after all, sort of related to aviation ;-).

An article by Mark Pedley in the August 2012 edition of Circuit Cellar spells out the required steps for mapping the sensor axes/signs for consistency with the aerospace frame convention.

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

Here is a summary :

1. Having decided on the N(X)E(Y)D(Z) frame and arbitrarily selecting a sensor breakout board "front" direction,

a. +ve pitch = rotation up about the board Y axis

b. +ve roll = rotation right about the board X axis

c. +ve yaw = rotation clockwise about the board Z axis

The gyroscope X, Y and Z axes and signs are re-mapped in software to be consistent with this.

2. With the board resting horizontally (Z axis down), the accelerometer Z axis value is assumed to represent +1g (the others are 0g). Similarly, with the board X axis pointing vertically down, the accelerometer X axis value should be +1g, and when the board Y axis points down, the accelerometer Y axis value should be +1g. So this tells us how to map the accelerometer sensor axes and signs.

3. Place the board flat and rotate it slowly in the yaw axis (around Z). The magnetometer X value should be maximum when the board "front" or X axis points towards magnetic North, and minimum when the X axis points South. The Y value will be a minimum when the board front points East, and maximum when the board front points West. Note the Z axis value, it will be small and relatively constant. Now invert the board and check the Z axis reading. If in the northern hemisphere, this Z axis reading should be lower. This is because in the northern hemisphere the magnetic field vector points slightly downwards from horizontal. This gives us the software mapping of magnetometer sensor axes and signs.

For my sensor breakout board and desired "front" direction, these were the resultant mappings :

Accelerometer

Ax	-Y
Ay	-X
Az	Z

Gyroscope

Gx	Y
Gy	X
Gz	-Z

Magnetometer

Mx	Y
My	X

Mz	-Z

Sensor Configuration

Gyroscope : +/-500dps, 22Hz BW, 250Hz ODR (this gives us 4mSecs for external AHRS computation code)

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

Magnetometer : +/- 1.3 Gauss, 75Hz ODR

Sensor Calibration

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.

The gyroscope sensitivity could be calibrated by using a rotation table, but I chose to simply use the datasheet sensitivity figures. E.g. for our selected +/-500 dps full scale, the datasheet specifies the sensitivity as 60 counts per milli dps.

Accelerometer

The desired accelerometer calibration parameters are axis offsets and axis sensitivities. The sensitivity is specified in the datasheet, but since it is easy enough to calibrate, I use the calibrated values instead of the datasheet values.

1. Place the unit horizontal, i.e. the accelerometer should be horizontal. Average several readings. The X and Y axis values are the X and Y zero G biases, the Z reading is equivalent to +1g (we are using an airspace coordinate frame with Z axis pointing down). Turn the unit upside down to get the Z -1g reading.

2. Place the unit with X axis pointing down, average several readings to get X +1g. Repeat with X axis pointing vertically up, to get the X -1g reading.

3. Repeat step 2 with the Y axis to get the Y +1g and -1g readings.

Z axis sensitivity = (Zplus1g reading - Zminus1g reading)/2 counts/g

X axis sensitivity = (Xplus1g reading - Xminus1g reading)/2 counts/g

Y axis sensitivity = (Yplus1g reading - Yminus1g reading)/2 counts/g

Z axis 0g bias = Zplus1g reading - Z axis sensitivity

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. So 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 (waving it around in your hand). Monitor the minimum and maximum readings on all three axes. Repeat the motion several times.

Mx sensitivity = (Mxmax - Mxmin)/2

Mx bias = (Mxmax + Mxmin)/2

My sensitivity = (Mymax - Mymin)/2

My bias = (Mymax + Mymin)/2

Mz sensitivity = (Mzmax - Mzmin)/2

Mz bias = (Mzmax + Mzmin)/2

Repeat the procedure a few times to make sure you are getting consistent results.

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.

This breakout board could be used in different ways :

1. Tilt compensated "e-compass". This fuses data from accelerometer and magnetometer.

Ref : Freescale App Note AN4248 : Implementing a tilt-compensated e-compass using accelerometer and magnetometer sensors.

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

2. AHRS quaternion computation using Sebastian Madgwick's algorithm and sample code

http://www.x-io.co.uk/quaternions/

https://github.com/kriswiner/MPU-9250/blob/master/quaternionFilters.ino

3. Gravity compensated acceleration estimation (FreeIMU repository, varesano.net)

Application Problem Formulation

We are trying to estimate altitude and climbrate for a paragliding altimeter/variometer. This problem can be modeled as the tracking of position and velocity of a particle moving linearly 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.

This Wikipedia page has a great explanation of a basic Kalman filter that does just this, given position data samples.

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

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 could we improve on this ? Supply additional sensor data that can be used along with the pressure sensor.

Let's assume we have both altitude and vertical acceleration data input.

Let's also assume that we have a 32bit microcontroller with FPU and running at 80+MHz :-). In our case, it happens to be the embedded Sparc core on the Navspark GPS breakout board, which happens to have some free cycles over from executing GPS/GLONASS tracking and location computations.

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

Here are a couple of Kalman filter tutorials on the net :

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

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

From reading these articles, and my own initial experiments, it became clear that the sensor fusion Kalman filter would have to track not just the desired parameters (altitude and climbrate), but also any systematic offsets with unpredictable drift.

So apart from altitude and climbrate, we introduce another parameter - the systematic bias in the acceleration value measured by the accelerometer. The bias is an unknown parameter and can drift unpredictably. There are now three Kalman *state variables* :

z_k (altitude), $\dot{z}_k$ (climbrate) and $\ddot{z}b_k$ (acceleration bias)

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

These three parameters at a given sample time index can be related to the samples at the previous time index by the following equations :

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

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

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

where dt is the time interval between the two samples, and η_b is a random variable representing accelerometer bias unpredictability (noise). In vector/matrix notation, the state transition equations can be written as

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

Or

$$\boldsymbol{x}_k = \mathbf{F}\,\boldsymbol{x}_{k-1} + \boldsymbol{w}_k$$

where our state vector $\boldsymbol{x}_k = \begin{bmatrix} z_k \\ \dot{z}_k \\ \ddot{z}b_k \end{bmatrix}$

F is the state transition model

$\boldsymbol{w}_k$ represents the perturbation of the model

Q = Process Noise Covariance Matrix

$$\mathbf{Q} = E(\boldsymbol{w}_k\, \boldsymbol{w}_k^T)$$

$$= E\left(\begin{bmatrix} \frac{dt^2}{2}\ddot{z}_{k-1} \\ dt.\ddot{z}_{k-1} \\ \eta_b \end{bmatrix} \begin{bmatrix} \frac{dt^2}{2}\ddot{z}_{k-1} & dt.\ddot{z}_{k-1} & \eta_b \end{bmatrix}\right)$$

Assuming zero mean Gaussian noise,

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

$= \begin{bmatrix} \boldsymbol{Q}_{acc} & 0 \\ 0 & \boldsymbol{Q}_{accbias} \end{bmatrix}$

State Covariance Matrix **P**

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

The following sequence of operations is executed once for each iteration of the Kalman filter, i.e. once for each new pressure sensor-derived altitude sample.

The accelerometer sample data rate in practice is much higher. In my current implementation, I get 6 accelerometer samples for each altitude sample. Low pass filtering and down-sampling results in an acceleration data sample corresponding to each altitude sample.

Prediction

1. Compute a priori (predicted) Covariance Matrix

$\boldsymbol{P}_{k|k-1} = \boldsymbol{F} \quad \boldsymbol{P}_{k-1|k-1}\boldsymbol{F}^T + \boldsymbol{Q}$

2. Compute a priori (predicted) state

$accel \quad = accelmeas \quad - \quad \ddot{z}_{bk-1}$

$\dot{z}_{k-1} \mathrel{+}= \; accel * dt$

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

Update

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

$\boldsymbol{m}_k = \mathbf{H}\, \boldsymbol{x}'_{\boldsymbol{k}} \quad + \boldsymbol{v}_k$

H = [1 0 0] is the matrix used to extract the equivalent variable from the state vector. This results in the following scalar equation:

$m_k = \; z'_k \quad + \eta_z$

where η_z is a random scalar representing altitude measurement noise.

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

$\boldsymbol{y}_k = \boldsymbol{m}_k - \mathbf{H}\boldsymbol{x}_{k|k-1}$

$$= \begin{bmatrix} z'_k \\ 0 \\ 0 \end{bmatrix} - [1 \quad 0 \quad 0] \begin{bmatrix} z_{k-1} \\ \dot{z}_{k-1} \\ \ddot{z}b_{k-1} \end{bmatrix}$$

resulting in the scalar equation

$y_k = z'_k - z_{k-1}$

4. Compute Innovation Covariance $\boldsymbol{S}_k = \boldsymbol{H} \quad \boldsymbol{P}_{k|k-1}\boldsymbol{H}^T + \boldsymbol{R}$

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

Similarly, $\boldsymbol{S}_k$ is also a scalar = s_k

$$s_k = [1 \quad 0 \quad 0]\boldsymbol{P}_{k|k-1}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \sigma^2_{zmeas}$$

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

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

$= \boldsymbol{P}_{k|k-1}\boldsymbol{H}^T\boldsymbol{S}_k^{-1}$

$$= \boldsymbol{P}_{k|k-1}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \frac{1}{p_{zz} + \sigma^2_{zmeas}}$$

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

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

$\boldsymbol{x}_{k|k} = \boldsymbol{x}_{k|k-1} + \boldsymbol{K}_k\boldsymbol{y}_k$

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

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

7. Compute a posteriori (update) covariance matrix

$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_{k|k-1}$

$$= \boldsymbol{P}_{k|k-1} - \begin{bmatrix} k_z p_{zz} & k_z p_{zv} & k_z p_{za} \\ k_v p_{zz} & k_v p_{zv} & k_v p_{za} \\ k_a p_{zz} & k_a p_{zv} & k_a p_{za} \end{bmatrix}$$

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

Computing Gravity Compensated Acceleration

We could simply use the magnitude of the acceleration data vector from the accelerometer.

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

For an ideally calibrated accelerometer, with the unit at rest, the acceleration magnitude should be 1g for any orientation of the sensor.

$$\|a\| = 1$$

This is in the **sensor frame**.

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

$$\|a\| - 1$$

Assuming altitude is tracked in cm, the compatible acceleration value is then

$$(\|a\| - 1) * 980 \text{ cm}/s^2$$

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 gravity compensated 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.

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

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

$$\boldsymbol{a}_{sensorframe} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

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

Assuming Z axis points down, the static acceleration gravity vector in earth frame **g** = $\begin{bmatrix}0\\0\\1\end{bmatrix}$

Subtract the gravity vector from the earth frame acceleration vector, and the resultant vector gives us the net dynamic acceleration in earth frame.

The z component of this vector gives us the vertical acceleration component. This is what we need to supply to our Kalman filter fusing altitude and acceleration data.

$\boldsymbol{a}_{earthframe}$ = Rotate($\boldsymbol{a}_{sensorframe}, \boldsymbol{q}$)

= $\boldsymbol{q}\ \ \boldsymbol{a}_{sensorframe}\boldsymbol{q}^{-1}$

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

The z component of the acceleration vector is

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

The net vertical acceleration

$a_{z_{earthframe\_net}}$ = $a_{z_{earthframe}}$ - 1 (in g's)

Or in cm/s^2

= ($a_{z_{earthframe}}$ - 1)*980 cm/s^2

AHRS Computation when acceleration is not equal to 1g

In the Madgwick AHRS algorithm, the gyroscope is used to update the quaternion orientation, while the accelerometer and magnetometer are used to provide corrections. Gyroscope data is low-noise and high sensitivity, but can drift unpredictably (random walk, not zero mean noise). Accelerometer data is noisy, lower sensitivity but stable, so can be low pass filtered for long term (compared to the gyroscope ODR) corrections.

However, to obtain *orientation* from the accelerometer and magnetometer, the system must be in a 1g environment.

In practice, as long as the magnitude of the acceleration vector is in an "acceptable" range (0.5g to 1.5g), the accelerometer and magnetometer data can be used to correct the gyroscope readings.

When the acceleration magnitude falls outside this range, the solution is to use gyroscope data integration alone, i.e. coast open-loop to estimate the orientation.

Over time the gyroscope drift will accumulate, so there is a limit for the time spent in this high or low g environment where the AHRS error is acceptable. This depends on the gyroscope drift and noise. A good reason to choose the MAX21100 is its low gyro drift and noise specification. A comparable component might be the MPU9250.