Fast Rotations

William Premerlani
June 18, 2011

One of my personal interests during the development of MatrixPilot was to stabilize my plane during a sustained, high speed rotation. Part of the motivation was a challenge presented by a member of diydrones to control a plane during a sustained turn. Soon after the first version of MatrixPilot was done, I began experimenting with high speed spins. What I found was that the stabilization behaved strangely after about 10 high speed revolutions, so I put the question on my list of things to investigate.

Recently, I found time to research the question some more. I set up a bench-testing arrangement with the UDB plus a magnetometer mounted on a record player that I could spin at 33, 45 or 78 RPM. I connected an OpenLog to the UDB, and spun it in various orientations of both the board with respect to the record player, and the orientation of the record player itself. The first few tests produced some surprising results.

“Dizzy” Testing

In one of the first tests, I simply placed the UDB horizontally on the turn table, and spun it horizontally for about 10 minutes, and then stopped it. The reported values of the bottom row of the direction cosine matrix, as the rotation slowed down and stopped, are shown in Figure 1.

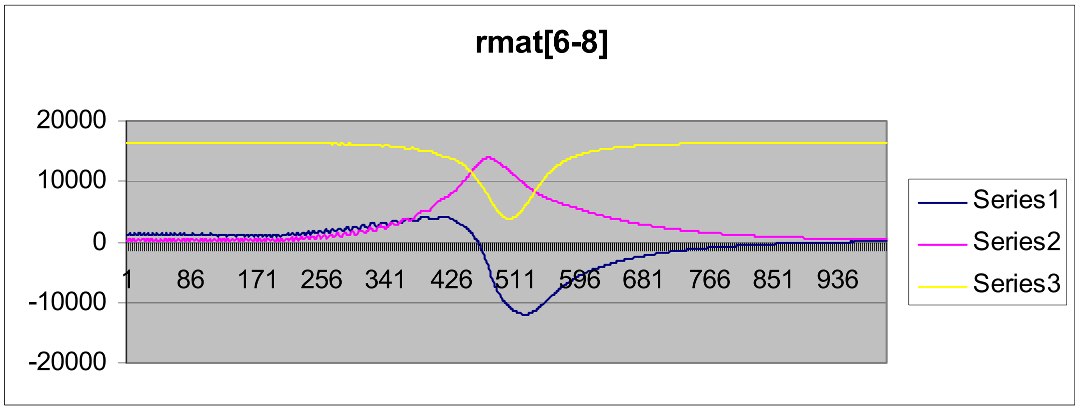

Figure 1. Bottom row of the direction cosine matrix as the UDB slows down after spinning horizontally at 78 RPM for 10 minutes.

The bottom row of the direction cosine matrix (elements rmat[6], rmat[7] and rmat[8]) indicates the earth’s vertical axis as seen in the body frame of reference. For a horizontal UDB, those values should be 0, 0 and 16384. Interestingly enough, those were the values reported by the DCM algorithm while the UDB was spinning. Everything seemed to be fine then. But as the spinning slowed down and stopped, the UDB became “dizzy”, and those three values went completely wrong. Eventually the drift compensation portion restored everything back to what it should be, but it took several minutes. Figure 1 raised several questions, including what was making the UDB “dizzy”, and why did the effect present itself after the spinning stopped?

The first clue to what was going on is in the plot of the roll and pitch gyro signals, shown in Figure 2.

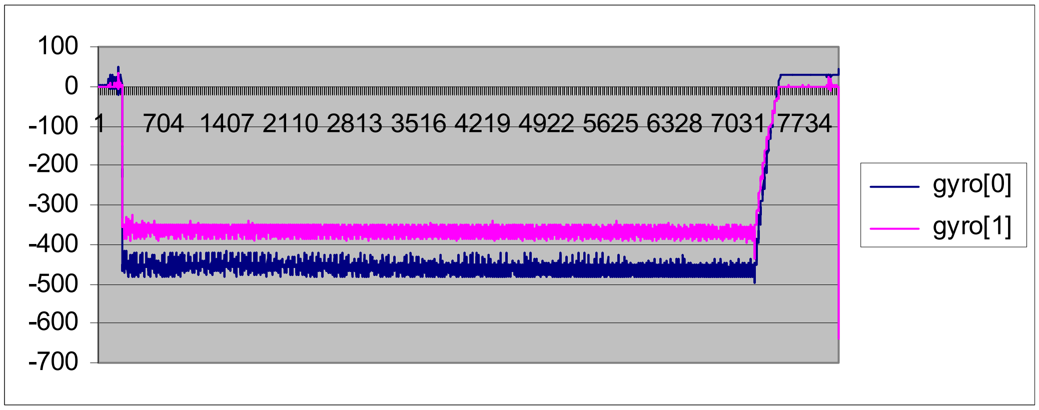

Figure 2. Roll and pitch gyro signals during horizontal spin

During a horizontal spin, the roll and pitch gyro signals should be equal to 0, but they were not. Actually, they were pretty close to zero, relatively speaking. The Z axis gyro signal was 9000, so the roll and pitch signals represented about a 5% “cross coupling”. Where was the cross coupling coming from? A glance at the board revealed that it was not quite level. Because of the wires to the board and the flexibility of the velcro mounting, the board was tilted by about 1.5 degrees. That was not much tilt, but it was enough to produce significant signals in the roll and pitch gyros.

Now, if the board was “really” tilted (which in some sense it was), the roll and pitch signals would produce the correct values of the elements of the matrix. However, the problem was that the board was also tilted during power up. The net effect was a slight misalignment between the gyro axes and the accelerometer offsets. This was readily apparent in the behavior of the roll and pitch matrix elements during the start of the spin, shown in Figure 3.

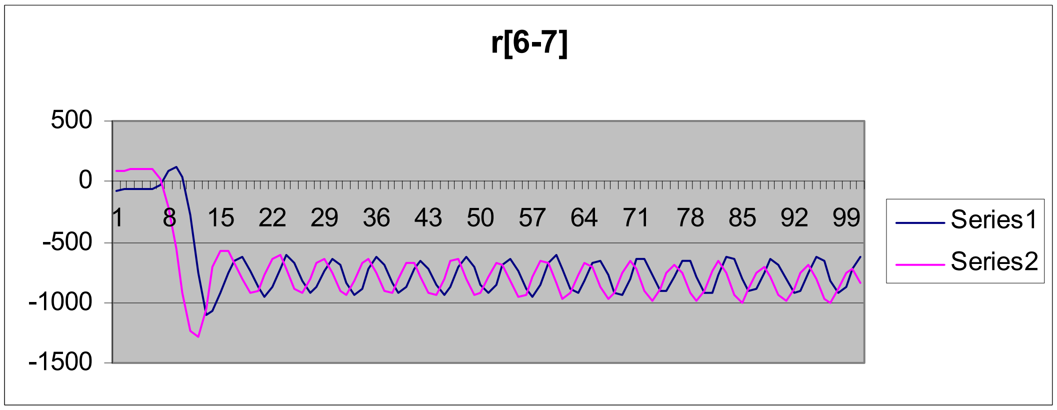

Figure 3. Roll and pitch indications at the start of the spin

Just before the rotation started, roll and pitch were very close to zero, because the UDB was very close to the same orientation that it was in during power up. However, as soon as the spinning started, there was a slight shift due to the misalignment during power up. Later on, after some math, the explanation for the various features of Figure 3 will become clearer.

Ok, so there is a slight error in roll and pitch. Why should that cause so much trouble? The reason is that because of an effect that I will explain shortly, the PI feedback drift compensation cannot remove the slight errors while the UDB is spinning rapidly, and it "precesses" instead. This is quite evident in the plot of the integral feedback, shown in Figure 4.

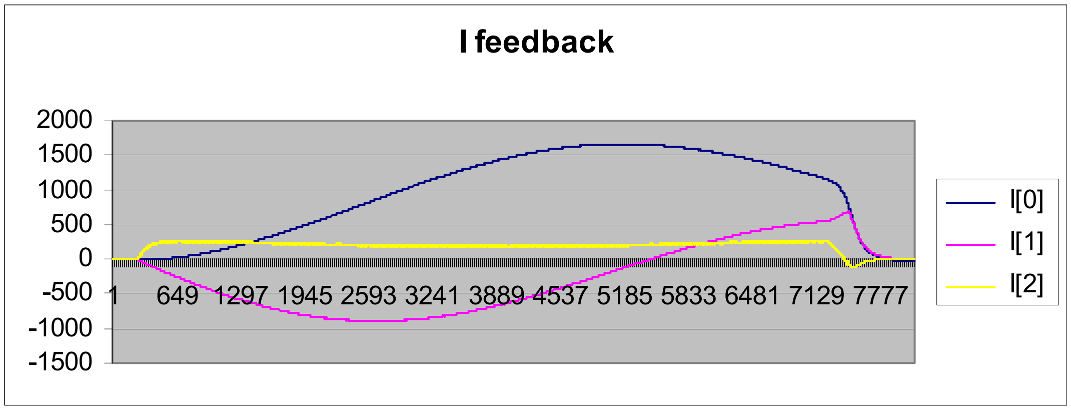

Figure 4. Integral feedback during extended horizontal spinning

There are several interesting features of Figure 4. First, the integral feedback for the yaw axis is correct and well behaved. It is compensating for a slight error in the calibration of the yaw axis gyro. However, the integral feedback for roll and pitch is all over the map. It is actually "precessing" around the roll and pitch error signals caused by the misalignment during power up. It is trying to null out those errors, but it cannot, because the UDB is spinning too fast. A little bit of math will quickly show what is going on.

Spinning Math

Lets start by analyzing a spinning board that is approximately level and that has some misalignment at power up. For now, we will ignore the integrators in the PI feedback control, and look at just what happens if we consider just the proportional term. What we want to understand is how the misalignment manages to resist the feedback controller. If we represent the roll as X and the pitch as Y, and account for the DCM algorithm plus proportional feedback, we find the differential equations governing the dynamics are given by Equation 1.

Physically, the first term on the right side of each equation is due to the proportional term in the feedback controller. The second term represents the rotation update computation. The third term is the gyro error signal. It is the combination of the second and third terms that causes the problem.

$$\begin{aligned}
&\frac{dX}{dt} = -K_P \cdot X - \omega_Z \cdot Y + \omega_Y \\
&\frac{dY}{dt} = -K_P \cdot Y + \omega_Z \cdot X - \omega_X \\
&X = \text{roll} \\
&Y = \text{pitch} \\
&K_P = \text{proportional drift compensation gain} \\
&\omega_Z = \text{yaw rotation rate} \\
&\omega_X, \omega_Y = \text{roll, pitch gyro error signals}
\end{aligned} \qquad \text{Equation 1}$$

The solution to equation 1 is:

$$\begin{aligned}
&X(t) = X_\infty + r(t) \cdot \cos(\omega_Z \cdot t + \theta_0) \\
&Y(t) = Y_\infty + r(t) \cdot \sin(\omega_Z \cdot t + \theta_0) \\
&r(t) = r_0 \cdot \exp(-K_P \cdot t) \\
&X_\infty = \frac{K_P \cdot \omega_Y + \omega_X \cdot \omega_Z}{K_P{}^2 + \omega_Z{}^2} \\
&Y_\infty = \frac{-K_P \cdot \omega_X + \omega_Y \cdot \omega_Z}{K_P{}^2 + \omega_Z{}^2} \\
&\sqrt{X_\infty{}^2 + Y_\infty{}^2} = R_\infty = \frac{\sqrt{\omega_X{}^2 + \omega_Y{}^2}}{\sqrt{K_P{}^2 + \omega_Z{}^2}}
\end{aligned} \qquad \text{Equation 2}$$

In other words, the roll-pitch trajectory spirals into a fixed point whose distance from 0 is equal to R_∞. We would prefer the fixed point to be zero, but it is not. The question is, how far from 0 is it? Lets take a closer look at the expression for the distance. We can rewrite it in terms of the tilt error at power up:

$$\begin{aligned}
&tilt = \frac{\sqrt{\omega_X{}^2 + \omega_Y{}^2}}{\sqrt{\omega_Z{}^2}} \\
&R_\infty = tilt \cdot \frac{\sqrt{\omega_Z{}^2}}{\sqrt{K_P{}^2 + \omega_Z{}^2}}
\end{aligned} \qquad \text{Equation 3}$$

We can see that the radius of the fixed point goes to zero when the board is not spinning. This is consistent with the observation that MatrixPilot works just fine, it does not get dizzy, when the board is not spinning. We can also see that the radius of the fixed point grows as the yaw rate increases, until it reaches an asymptotic value equal to the amount of tilt on power up. For MatrixPilot, Kp is approximately equal to 0.15, so the asymptotic value is reached at about 0.15 radians per second, or a spin rate of about 8 degrees per second.

Physically, what is going on is that the rotational update of the misalignment error presents the drift compensation algorithm with a “moving target”. As long as the target is moving slowly, drift compensation can keep up. But as the spinning rate increases, the target begins to “blur” because of the filtering properties of the feedback.

At first, this worried me, because at first glance I thought that the drift compensation would stop working all together when the board is spinning, but that is not the case. The key feature of the misalignment effect is that it presented the drift controller with a non-stationary signal. All of the real sources of error present the controller with slowly varying error signals that it can lock onto.

So far, there is no cause for alarm. The implications of equation 3 is that while the UDB is spinning, the initial misalignment will cause a small roll pitch error that is no bigger than the original misalignment. And in fact, there is no problem right away. The problem that eventually arises is due to the behavior of the integrator in the PI feedback loop. Lets analyze that case. Suppose that the spin rate is very high. In that case, the offsets in equation 2 are given approximately by:

$$
\begin{aligned}
X_\infty &= \frac{\omega_X}{\omega_Z} = tilt_X \\
Y_\infty &= \frac{\omega_Y}{\omega_Z} = tilt_Y
\end{aligned}
\qquad \text{Equation 4}
$$

If we now include the effect of the integral feedback, after the initialization transients have died out, it can be shown that the differential equations of the dynamics are given by:

$$
\begin{aligned}
\frac{dIX}{dt} &= K_I \cdot \left(tilt_Y + \frac{IY}{\omega_Z} \right) \\
\frac{dIY}{dt} &= -K_I \cdot \left(tilt_X + \frac{IX}{\omega_Z} \right) \\
IX &= \text{roll integral feedback} \\
IY &= \text{pitch integral feedback}
\end{aligned}
\qquad \text{Equation 5}
$$

The solution to equation 5 is a circular trajectory, which is pretty much what we see! In other words, it is a precession of the misalignment vector. It is interesting to note what the rate of precession is:

$$
\omega_{precession} = \frac{K_I}{\omega_Z} \qquad \text{Equation 6}
$$

So, the faster the UDB spins, the slower the misalignment vector precesses. Which is exactly like a spinning top, and is exactly what I saw in my tests.

Now is a good time to go back and review figures 1, 3 and 4, starting with the key figure, Figure 4. In figure 4, while the board is spinning, the integrators for roll and pitch are precessing in a circle at a very slow rate, given by equation 6. When the board stops spinning, the energy that is was trapped in the integrators decays. Referring to Figure 1, the matrix elements are approximately correct while the board is spinning. That is because the error due to the initial misalignment is simply precessing. The energy in the integrators can do no harm while the board is spinning, because the rotation "deflects" the integrator effects. In other words, suppose the integrator causes the attitude estimate to begin to roll to the right. After the board makes a 180 yaw rotation, the effect is to roll to the left, so the net effect is zero. However, as the board stops rotating, this "gyroscopic" effect goes away, and the trapped energy in the integrators causes large roll and pitch errors.

Referring to Figure 3, what we are seeing at the startup of the spinning is a spiraling to the fixed point given by equation 2. In other words, there must be a small roll and pitch error to get the drift compensation controller to produce signals to cancel the signals produced by the gyros, shown in Figure 2. Notice in Figure 3 that after start up of the spinning there is still a small rotation of roll and pitch that is not included in equation 2. That is because my turntable itself was not perfectly level, so it was not spinning in the horizontal plane. That was evident in the output of the accelerometer during the test. When I accounted for a tilted turntable, I produced analytic expressions that exactly matched Figure 3.

I also did some simulations which closely matched test results. For example, Figure 5 is the roll-pitch trajectory in the X-Y plane for one of the spin tests:

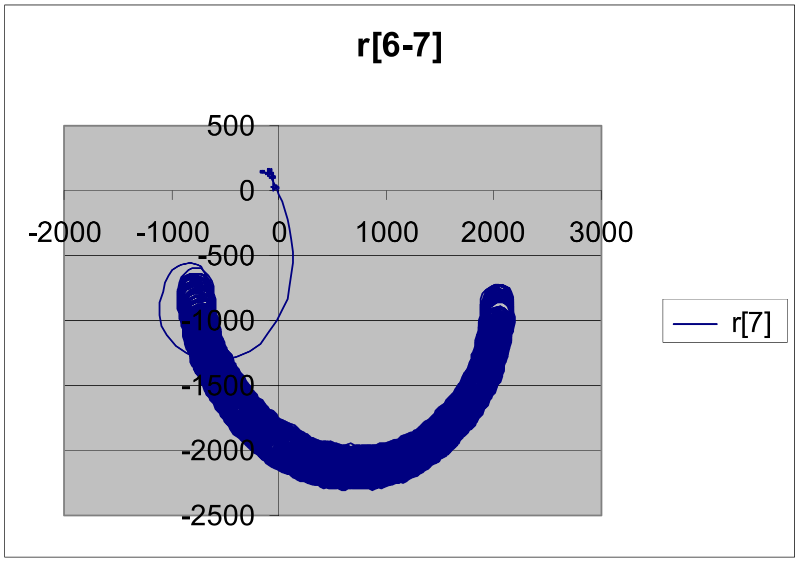

Figure 5. Trajectory of roll-pitch attitude estimate during horizontal spin test

Note the initial spiral from 0 to the roll-pitch misalignment error. After that, the trajectory orbits a focal point that itself slowly orbits in the X-Y plane, just as predicted by the theory. Very similar behavior is shown in the result of a simulation in Figure 6.

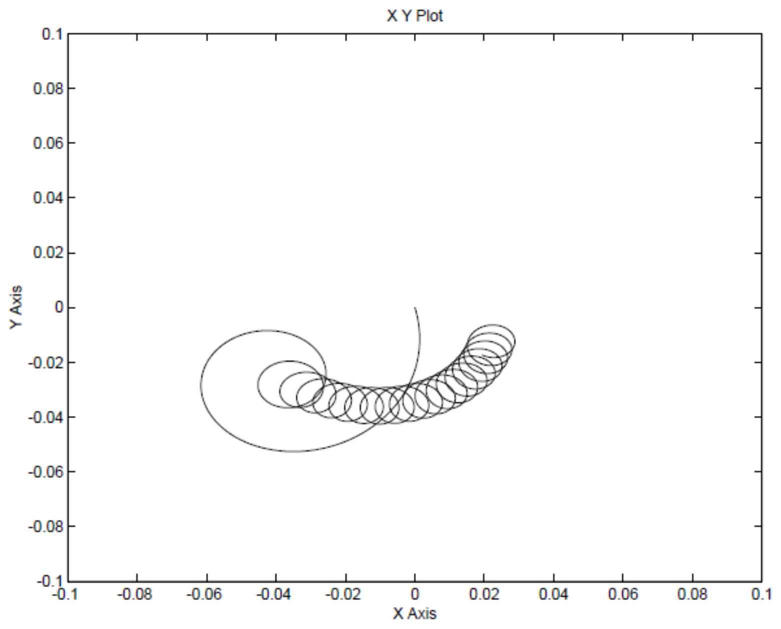

Figure 6. Simulated roll-pitch trajectory during 78 RPM spinning

So, now the problem is well understood. We could say that the root cause of the problem is the fact that board was not level at power up. But there will always be errors of that sort, it would not be reasonable to expect users to achieve “perfect alignment”, so we will look for a robust solution.

The simplest solution is to simply turn off the drift compensation integrators when the board is spinning too fast for the drift compensation controller to lock onto non-stationary error signals. In any case the integrators are only useful when the board is not spinning. At that point, gyro calibration errors are much more important than the gyro offsets that the integrators are targeting.

I thought about getting rid of the integral terms in the drift controller all together, but then there would not be any compensation for gyro offsets, which should be taken into account when the board is not spinning.

So, what I tried was to simply turn off the integrators when the spin rate around the spin axis exceeded the rate that the controller could handle, which is around 10 degrees per second. Both ground tests and flight tests showed that was a good solution. Figures 7 and 8 show the results of one of the ground tests. Figure 7 shows the gyro signals, mostly to indicate when the spinning started and stopped.

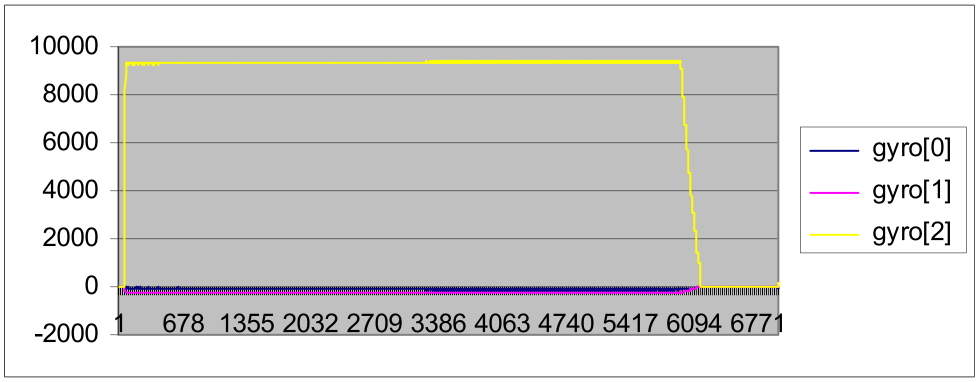

Figure 7. Gyro signals during spin test at 78 RPM

Figure 8 shows the bottom row of the direction cosine matrix after implementing the simple change to make the computations “dizzy proof” All that was done was to turn off the integrators when the spin rate was greater than 10 degrees per second. Compare it with Figure 1 to see the dramatic improvement when spinning stops. The new implementation is “dizzy proof”

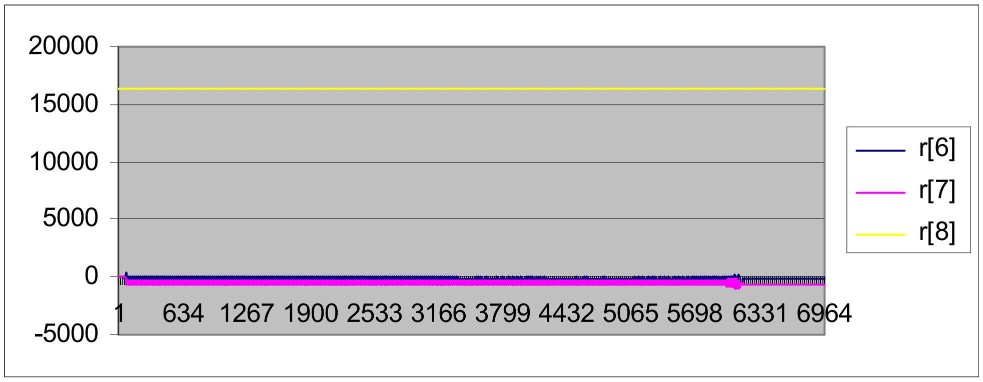

Figure 8. Results of 78 RPM spin test of “dizzy proof” implementation

Variable Proportional Feedback Gain

Ok, so by turning off the drift compensation integrators during spins, we can make the DCM algorithm “dizzy proof”. The next question is, how can we improve the accuracy beyond what we already have?

The calibration of the gyros is a known limiting factor. There is some small variation (on the order of a few percent) from one gyro chip to another. Under most conditions, a slight calibration error is of no consequence. That is because the errors that result tend to cancel out during normal flights. In order to accumulate error due to calibration error, you need to rotate your plane around some axis in a continuous fashion at a high rate. The easiest way to do this is to put your plane into a spinning dive at a high yaw rate.

For example, suppose you spin your plane at 500 degrees/second, and suppose that the gyro calibration is off by 5%. Then the yaw rate reported by the gyros will have a 25 degree per second error, which will produce a yaw estimate error that will begin to grow at the rate of 25 degrees per second. The drift compensation feedback will detect and correct for the error, mostly through the proportional term, which has a gain of about 0.15. That means that, in the worst case of a 5% gyro gain error, the angle error will grow to 167 degrees before it stops increasing. That is very close to losing lock.

There is a simple way to improve the situation a great deal: make the proportional feedback variable, and increase it with increasing rotation rate. The logic behind this is that the gyro error is proportional to rotation rate, so by making the feedback proportional to rotation rate, the angle error becomes a smaller number, independent of rotation rate. Tests showed that the DCM algorithm was stable over a wide range of gains, so variable gain is a viable option. The only reason for not simply raising the gains once and for all and using constants is that impairs the performance when the plane is not spinning, especially during a hand launch. After much testing, I settled on raising the feedback gains linearly from 50 degrees per second to 500 degrees per second. Below 50 degrees per second, the gains are constant with the same values that we have been using. Between 50 degrees per second, the gains are multiplied by the rotation rate divided by 50, up to a boost of 10 times at 500 degrees per second. This reduces the angle error at 500 degrees per second from 167 degrees to 16.7 degrees, definitely a step in the right direction. I implemented the changes, bench tested them, and found it worked quite nicely.

Automatic Gyro Calibration

During all of the testing that I was doing, I developed a simple manual method to determine the gyro calibration during sustained rotations. I noticed that after the board had been spinning for a few seconds, the output of the feedback controls would settle out to be equal to the gyro error. For example, if the gyro gain was off by 5%, with a rotation rate of 500 degrees per second, the output of the feedback controller was a signal equal to 25 degrees per second. So, it was a simple matter to manually estimate the gyro calibration. Why not automate it, and let the gyros calibrate themselves?

The starting point in the theory is the notion that the gain error is equal to the feedback divided by the rotation rate. In one dimension, the concept is simple enough. But the rotation group is a nonlinear mathematical object, and some care must be taken in three dimensions to honor the properties of the rotation group. For example, suppose the rotation is not aligned with one of the axes of the board. What are the implications? After some mathematical analysis, I arrived at equation 7 for estimating the gain errors.

$$\Omega = [\omega_X \quad \omega_Y \quad \omega_Z]$$
$$\|\Omega\| = \sqrt{\omega_X{}^2 + \omega_Y{}^2 + \omega_Z{}^2}$$
$$\omega_X, \omega_Y, \omega_Z = \text{spin rates for each axis}$$
$$\mathbf{n} = \frac{\Omega}{\|\Omega\|} = [n_X \quad n_Y \quad n_Z] = \text{axis of rotation} \qquad \text{Equation 7}$$
$$\mathbf{\Delta gain} = \left[n_X \cdot \frac{P_X}{\|\Omega\|} \quad n_Y \cdot \frac{P_Y}{\|\Omega\|} \quad n_Z \cdot \frac{P_Z}{\|\Omega\|} \right]$$
$$P_X, P_Y, P_Z = \text{proportional feedback for each axis}$$

Note that gain errors are computed separately for each gyro axis. In general we would not expect the gyro gains to be the same for each axis.

Once the gain errors are estimated, it is a simple matter to use them to adjust the gyro gains. This was done by computing the gyro gains as the outputs of integrators, with the inputs computed by multiplying the outputs of Equation 7 by a filter gain. The integrator is active only when the rotation rate is high enough to provide calibration information, so the calibration is “locked in” when the plane is not spinning. The operation of the gain error estimator plus the feedback integrator drives the attitude estimation error to zero. Limits were placed on the integrator to keep them within plus or minus 10 percent of the nominal calibration. Results from one of the first tests are shown in figure 9 and 10. Figure 9 shows the yaw rotation rate, which started from rest, jumped to 33 RPM and held for a while, jumped to 78 RPM and held, then coasted to a stop. Figure 10 shows the computed calibration. Not shown in either of the figures is the proportional feedback, which was driven to zero. In other words, the gyros became calibrated exactly.

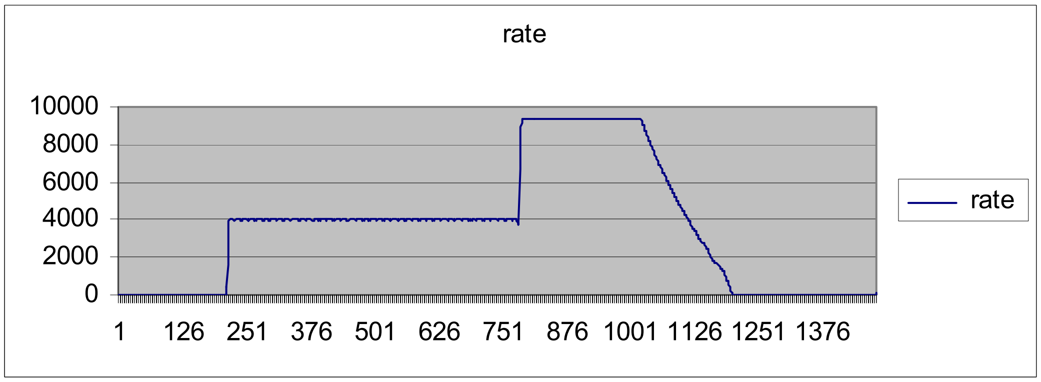

Figure 9. Yaw rotation rate profile for automatic gyro calibration test

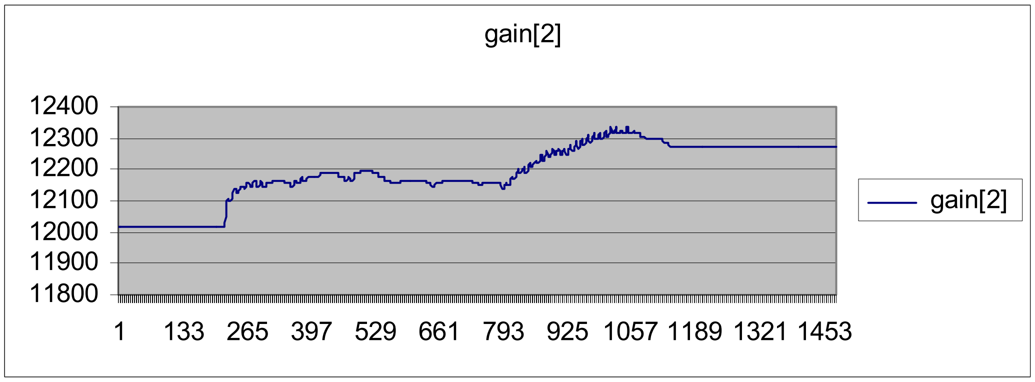

Figure 10. Computed yaw gyro gain

There are a few things you should notice about Figure 10. The gain computation is rather stable. The nominal gain was not far off to begin with. The maximum gain change was about 2.5%. Also notice that the computed gains increased with rotation rate. I was expecting that, because of an effect that I was aware of. So, now it was time to address it.

Nonlinear Effects

The matrix used to perform the update on the direction cosine matrix is a linearization of the nonlinear matrix that expresses a finite rotation. For low rotation rates it is fine, but for high rotation rates, it underestimates the amount of rotation. If we want to get the gyro calibration into the 0.5% range, we will have to take the effect into account. Ultimately we plan on using the full, nonlinear expressions for finite rotations. In the meantime, we can improve things by accounting for the error created by the approximation.

The linear approximation implicitly assumes:

$$\begin{aligned}&\tan(\Delta\theta) \approx \Delta\theta \\ &\Delta\theta = \omega \cdot \Delta t\end{aligned} \qquad \text{Equation 8}$$

A better approximation for our purposes is given by:

$$\frac{\tan(\Delta\theta)}{\Delta\theta} \approx \left(1 + \frac{\Delta\theta^2}{3}\right) \qquad \text{Equation 9}$$

The right hand side of equation 9 represents a “boost factor” in computing the angles that go into the update matrix. At 500 degrees per second, it amounts to an increase of 1.5%. So, I implemented equation 9 in the computation of the update matrix. Tests showed much improved results at high rotation rate, as shown in Figure 11 and Figure 12. Finally, that’s what we want!

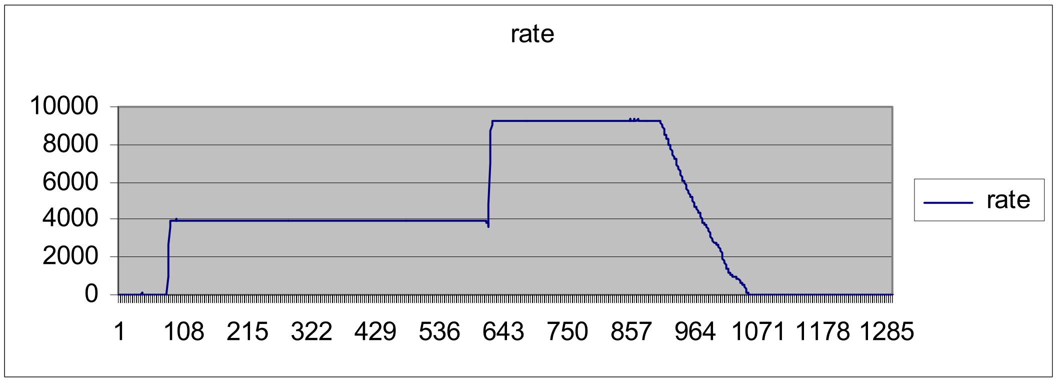

Figure 11. Yaw rotation rate profile for automatic gyro calibration test

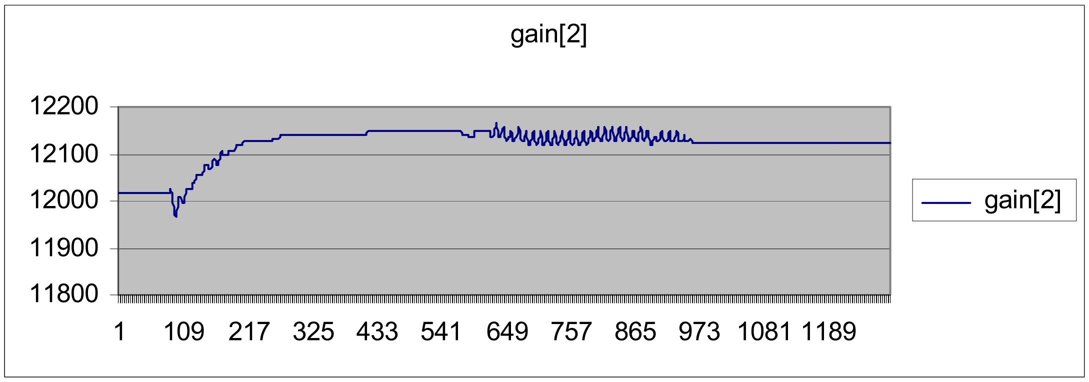

Figure 12. Computed yaw gyro gain

At this point we have extensions to the DCM algorithm that produce dramatic improvements in accuracy during extended periods of high rotation. But we are not done yet.

Magnetometer Latency

The last piece of the puzzle was magnetometer latency. The way the communications interface with the magnetometer works, there is some latency between the time that the magnetometer takes a reading and when it is used to perform yaw drift compensation. That is because the magnetometer operates in continuous reading mode, but it will not put the results into the reading registers until the previous results are read out. So each time the magnetometer is read, it is the previous result that comes out. In the case of MatrixPilot, the latency is 0.085 seconds. At 500 degrees/second, that represents a yaw error of 42.5 degrees.

The latency was measured optically by flashing an LED as the UDB was spinning, much like a timing light can be used to adjust the timing of the ignition system of an automobile. A camera was used to take an extended exposure picture. First, the board was rotated slowly, two full revolutions. The reference pattern is shown in Figure 13.

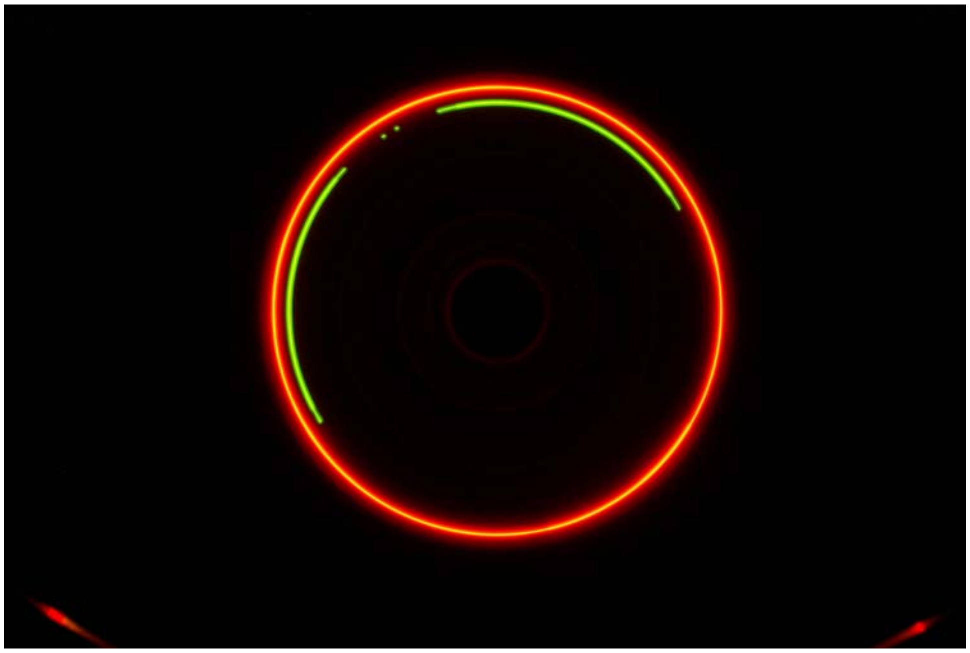

Figure 13. Reference pattern during two slow revolutions

Next, the board was spun at 78 RPM and an extended exposure revealed that the latency was 0.085 seconds. It was a simple matter to adjust for the latency in the yaw drift compensation by taking a snapshot of the direction cosine matrix elements 0.085 seconds before the magnetometer reading became available. Figure 14 shows an extended exposure of 10 revolutions after the latency compensation was implemented.

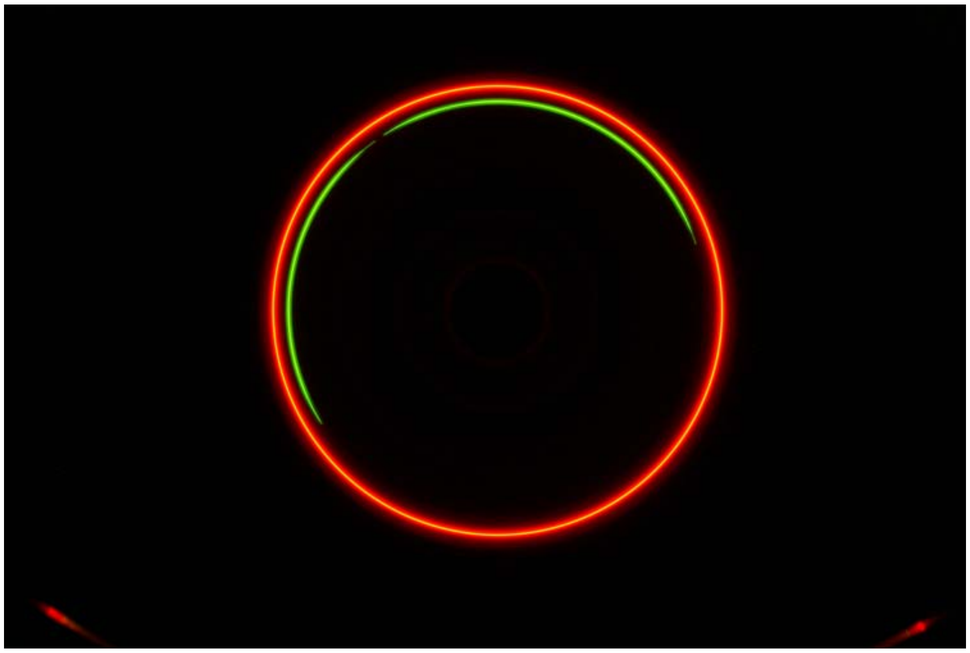

Figure 14. Timing pattern of 10 revolutions at 78 RPM

Final Testing

After all of the improvements were implemented, I ran a number of spinning tests to verify that everything was working properly. Figures 15 and 16 are from one of the tests. Figure 15 is a plot of rmat[8], which is the projection of the earth's Z axis onto the board's Z axis. Figure 16 is a plot of rmat[5], which is the projection of the earth's "north" axis with the board's Z axis.

The board was mounted directly on the record player and spun at 78 RPM, with the turntable horizontal, spinning axis vertical. After letting the board spin that way for several minutes, I tilted the turntable 90 degrees, onto its side, so the axis of spin was horizontal. As you can see from Figure 15, rmat[8] correctly tracked the tilt. By the way, the "sawtooth" disturbance in the waveform is real, as confirmed by the accelerometers. When the turntable was on its side, everything was wobbling somewhat because of the velcro mounting, and everything was deliberately slightly misaligned.

After the turntable was tilted 90 degrees, I slowly rotated it 360 degrees around the vertical, all the while the board was spinning at 78 RPM. This is correctly reflected in Figure 16.

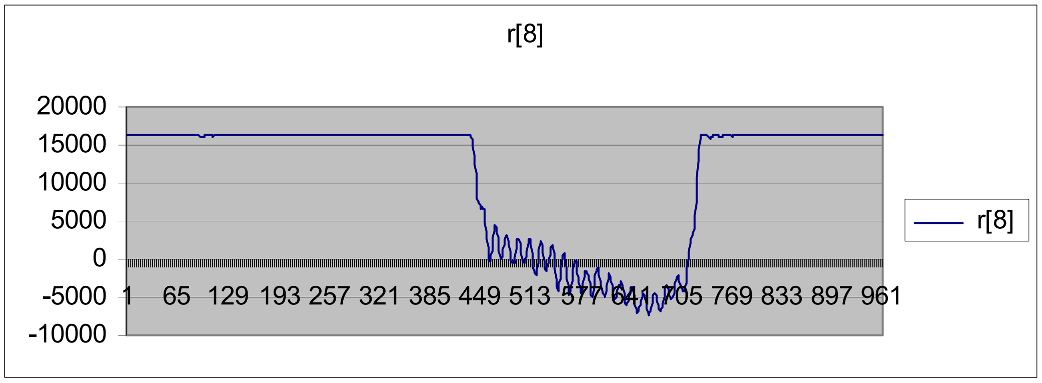

Figure 15. Rmat[8] during advanced testing at 78 RPM

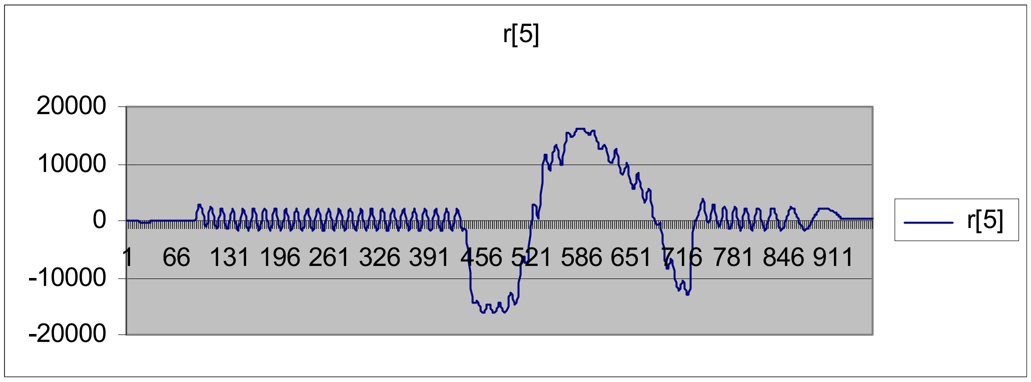

Figure 16. Rmat[5] during advanced testing at 78 RPM