Measuring the Tilt Dynamics of a Quadcopter

William Premerlani, March 4, 2012

Background

In designing multicopter controls and in evaluating the performance of multicopter airframes, it is useful to be able to measure the dynamic response of the airframe to tilt control signals. What would be useful to have is a model that describes what the roll and pitch rate response of the quad will be to small changes in roll and pitch inputs to the speed controls of the roll and pitch motors. This paper describes a technique for computing the parameters of that sort of model that uses flight data, without requiring any intrusion into the controls or any special sort of maneuvers and without having to make too many approximations and assumptions.

Theory

The first step is to select the form of the dynamic model that will approximate the tilt behavior. I decided to try a linear, time-invariant, finite-impulse model. The linear model should be a reasonable approximation if it is taken as a first order Taylor expansion around a base point, because a quad is very responsive, and my experience in flying them has been that only small inputs are needed to control tilt. As will be seen shortly, the actual model is an infinite-impulse model, but there is exponential decay, so if we take a wide enough time window, we can approximate an infinite impulse response with a finite response.

With digital measurement and control, the dynamic model can be expressed in terms of an impulse response function h[]. The output is the convolution of the input with h[], as shown in equation 1:

$$y[k] = \sum_{l=0}^{M} h[l] \cdot x[k-l] \qquad \text{Equation 1}$$

The samples x[] are the small roll and pitch signals added to the total values of the PWM signals sent to the motors. The samples y[] are the roll rate and pitch rate responses of the quad, as measured by the roll and pitch gyros. It is assumed that there are no offsets in x[] or y[]. Offsets can be handled separately by removing the average values from the x[] and y[] samples. It is assumed that there is no noise in x[] or y[]. (More about that later.)

What we are looking for is a non-intrusive way to determine the function h[]. It can be shown that can be done with data taken during a normal flight, with or without feedback

controls running, without injecting any special signals. We would like a method that could be used in flight, but would settle for starters a method that will work off-line. Actually, although the method described here is an off-line method, it could easily be extended to an in flight method. More on that later.

The starting point is to compute the correlation function between the inputs and outputs for a long sequence of input and output samples, as shown in equation 2:

$$\phi_{yx}[N_1, N_2 : n] = \sum_{k=N_1}^{N_2} y[k] \cdot x[k-n] \qquad \text{Equation 2}$$

The correlation function is shown as a function of n, with N1 and N2 as parameters. What equation 2 says to do is to time shift the input sequence by n samples, multiply it by the output sequence, and sum from N1 to N2. Computation of equation 2 for actual data shows that the correlation goes to zero as n becomes large. For roll and pitch, the correlation is zero after a couple of seconds.

Next, substitute equation 1 into equation 2, as shown in equation 3.

$$\phi_{yx}[N_1, N_2 : n] = \sum_{k=N_1}^{N_2} \left(\sum_{l=0}^{M} h[l] \cdot x[k-l] \right) \cdot x[k-n] \qquad \text{Equation 3}$$

Because the summations are finite, the order of the two summations can be reversed, leading to equation 4.

$$\phi_{yx}[N_1, N_2 : n] = \sum_{l=0}^{M} h[l] \sum_{k=N_1}^{N_2} x[k-l] \cdot x[k-n] \qquad \text{Equation 4}$$

We can compute the second summation from the input data, and define an auto-correlation function on the input sequence of data, as shown in Equation 5.

$$\phi_{xx}[N_1, N_2 : l, n] = \sum_{k=N_1}^{N_2} x[k-l] \cdot x[k-n] \qquad \text{Equation 5}$$

We note that equation 5 defines a symmetric matrix, with subscripts l and n. (Note that, because of the dependence on N1 and N2, we cannot in general express the auto-correlation as a function of l-n, although we will find that is approximately true.

If the determinant of the matrix is not zero, the inverse of the matrix can be computed. Experience with actual data shows that the matrix is approximately a diagonal matrix, with the off-diagonal elements rapidly falling to zero the farther they are from the diagonal. So, it is easy to compute the inverse of the auto-correlation function.

$$\psi_{xx}[N_1, N_2 : l, n] = \phi_{xx}^{-1}[N_1, N_2 : l, n] \qquad \text{Equation 6}$$

From the preceding equations, it can be shown that, assuming no noise, the function h[] can be exactly computed from the cross correlation function and the inverse of the auto correlation matrix, using equation 7.

$$h[m] = \sum_{l=0}^{M} \psi_{xx}[N_1, N_2 : m, l] \cdot \phi_{yx}[N_1, N_2 : l] \qquad \text{Equation 7}$$

So, we have a process for computing the impulse response function, h[]. Compute the cross-correlation vector using equation 2. Compute the auto-correlation matrix using equation 5. Compute the inverse of the auto-correlation matrix, and multiply it by the cross-correlation vector to produce the impulse response vector, h[].

Implementation

Armed with equation 7, I reasoned that the next thing to do was collect some flight data and try to compute h[] for my draganflier-based quad. I made a few 10 minute flights with stabilization controls operating. There were two types of flights that I tried. For some flights I applied aggressive roll and pitch commands. For other flights I simply hovered for 10 minutes. Mostly I tried to stay out of ground effect, because I reasoned that ground effect violates the assumptions of equation 1. Data was collected at the frame rate of the controls, 40 samples per second. I used a spread sheet to implement equation 7. The first results were interesting.

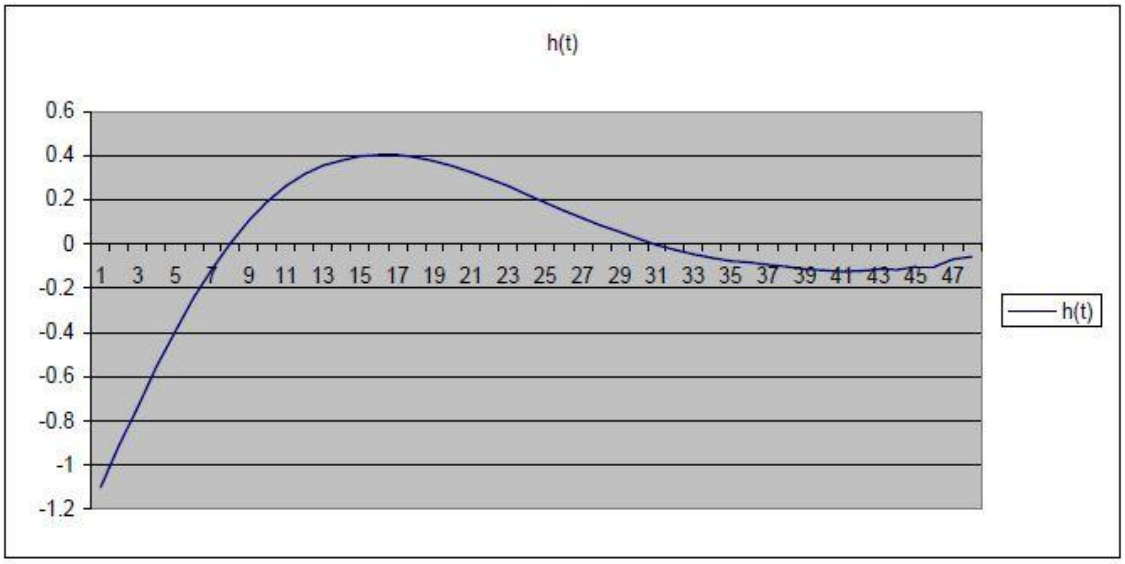

Figure 1 - Computed response function, h[n] for pitch control computed from flight data

The results of the first pass at using equation 7 on quad flight data to estimate h[] for pitch response produced the plot shown in Figure 1. The x axis is sample number, at 40 samples per second.

The plot is beautifully smooth, and at first glance seemed to be about the right overall shape. Except it seemed to me to be upside down! Also, a closer shows that the response was not at all what I expected it to be. I checked, and rechecked my computations, but I could not find any sort of sign error. I checked my spreadsheet, it seemed ok.

I thought the next thing to do would be to use equation 1 with the computed h[] on the actual input data, signals to ESC, to see how well it would predict the output data, gyro rate. Surprisingly, the predicted response closely matched the actual response, as shown in Figure 2. So, what was wrong?

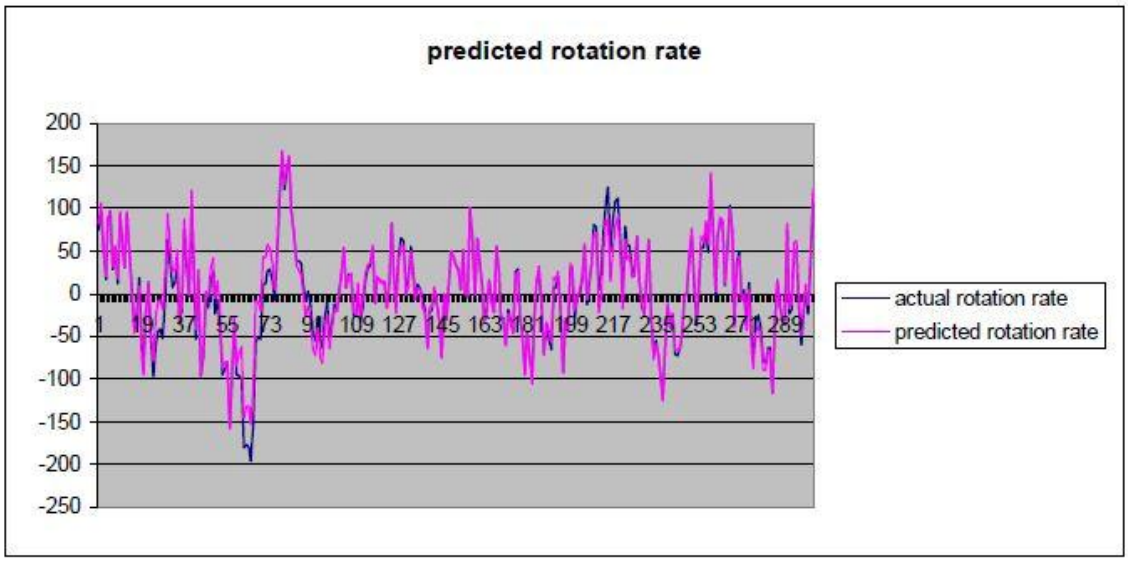

Figure 2 - Comparison of actual and predicted pitch rotation rates

As I studied Figure 2, I realized that my draganflier, which uses a slow, geared propellers, had quite a bit vibration. To test that, I measured the gyro signals when I shut the motors off and let them spin down, producing the plot in Figure 3.

The evidence of vibration in Figure 3 is very strong. So it hit me what was going on. There was a large amount of vibration in my draganflier. The vibration did not fit equation 1, because it was not a response to the controls. However, because of the large amount of vibration, there was a large response of the controls to the vibration. So, what I was really measuring was not the response of the quad to the controls, but rather the response of the controls to the quad. In other words, the vibration represented an error signal that "colored" the control inputs.

So, what to do next? I really wanted to know what the dynamic response was of my quad. I considered two approaches. One was to see if I could eliminate the vibration. That

seemed a lot of work. The other was to eliminate the response of the controls to the vibration. That turned out to be rather easy. The vibration was a fairly high frequency, so it could be blocked from the control inputs with a low pass filter. So, I modified the feedback controls accordingly. That did make the quad harder to fly, but it made it possible to compute the response of the quad to the controls, rather than vice-versa. The resulting computed impulse function for pitch is shown in Figure 4. The x axis is time. The units of the y axis correspond to the units used internally in MatrixPilotQuad. There are two plots. One is a plot of the computed response. The other is a damped sinusoidal response curve that I fitted to the computed response.

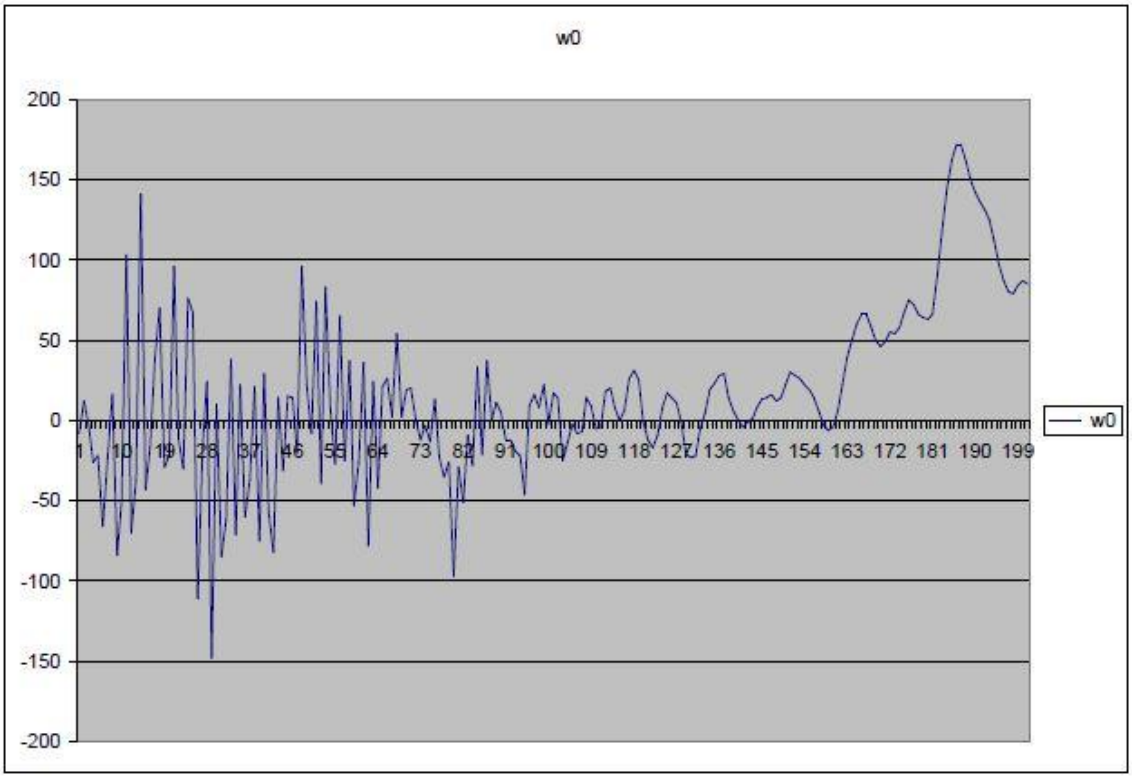

Figure 3 - Pitch gyro response during motor spin down

Finally! The shape of the response curve is a good match to theory. There are two inertial elements in the tilt dynamics of a fixed pitch quad. The first one is the inertial response of the speed of the motors and propellers to the torque developed by the motors. The second one is the inertial response of the rotation rate of the quad to the torque developed by the propellers. Aerodynamic forces, motor resistance, and ESC characteristics provide damping. It all adds up to a second order dynamic response.

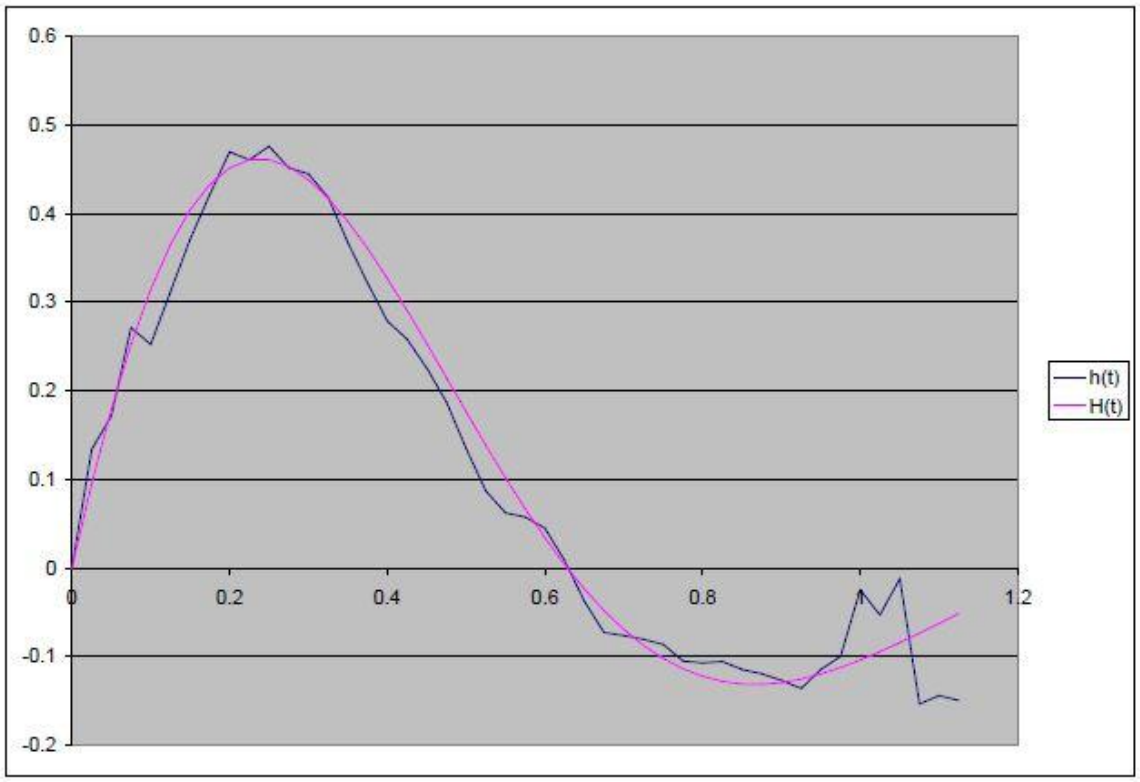

Figure 4 - Measured draganflier pitch impulse response

In Figure 4, the measured impulse response is h(t). The fitted response curve is H(t). I found that I got a good fit with an exponentially decaying sinusoidal function. By adjusting the parameters of that function, I found that h(t) of my draganflier tilt response was well approximated by equation 8:

$$H(t) = 0.8 \cdot \exp(-2t) \cdot \sin(5t) \qquad \text{Equation 8}$$

The corresponding Laplace transform is given by equation 9:

$$H(s) = \frac{5}{s^2 + 4s + 29} \qquad \text{Equation 9}$$

So, equation 9 is the Laplace transform of the tilt response of my draganflier, in units used internally by MatrixPilotQuad.

There is one dangling thread....

I mentioned earlier that it would be nice to be able to compute H(t) in flight in real time. It is possible to extend the method just described to do that, I will explain how in a subsequent report.