With support of NSF Award no. EEC-0754741

COMPACT ATTITUDE SENSOR SYSTEM USING SR-UKF

NSF Summer Undergraduate Fellowship in Sensor Technologies
Christopher Baldassano (Electrical Engineering) – Princeton University
Advisor: Dr. Mark Yim

ABSTRACT

This paper describes the selection and integration of attitude sensors for a novel flying device. The overall goal of the experimental flyer project is to create a small, cheap, and lightweight device that is capable of controlled flight. This project focuses on the development of a state estimation system for the flyer that allows for precise and rapid attitude measurement. A sensor package consisting of a three-axis magnetometer, a three-axis accelerometer, and three one-axis rate gyroscopes is presented. The sensor measurements are filtered and fused together using a Square-Root Unscented Kalman Filter (SR-UKF) developed in MATLAB. The feasibility of porting the filter to a Microchip dsPIC30F microcontroller is discussed, and the performance of the filter is evaluated.

1. INTRODUCTION

Figure 1: The Experimental Flyer

The objective of the experimental flyer project is to develop a miniature, highly-manueverable flying craft for indoor use. The main body of the flyer is cylindrical, with a length of about seven inches and a diameter of about three inches (see figure 1). Lift is generated by two propellers (one at each end of the cylinder) which spin in opposite directions. The flyer's novel torque generation scheme relies on a hinge in the center of the flyer, which allows the upper and bottom halves of the cylinder to pivot with respect to one another. The direction and magnitude of the torque can be easily calculated by conservation of angular momentum; the momenta of the propellers normally cancel since they are oriented in precisely opposite directions, but when the hinge pivots this cancelation is no longer exact, producing a compensating torque to maintain zero total angular momentum. The design of the experimental flyer is much simpler than that of traditional helicopters, which should allow it to be fabricated more cheaply and easily.

This paper describes the development of the attitude estimation sensor system for the flyer. The project consisted of the design, implementation, and testing of both the hardware and software elements of the system. The overal design goals of the system were:

- Size – must fit within the diamter of the flyer and be thin enough not to interfere with other components
- Weight – must not contribute significantly to the load on the flyer (which has a mass on the order of 300g)
- Speed – must be able to sense disturbances on the order of 100 Hz
- Precision and accuracy – must have sufficient resolution to allow for visibly stable flight
- Power – must require much less battery power than the motors (which consume power on the order of 50W)

Section 2 discusses the hardware and algorithms currently available for attitude estimation. Section 3 describes the selection and performance of the sensor hardware. Section 4 covers the implementation of the filtering algorithm and its feasibility for use on a microcontroller. Section 5 reviews the results of the system, and Section 6 offers suggestions for future development.

2. BACKGROUND

2.1 Attitude Sensors

The size, weight, and power requirements of the project necessitate the use of miniature state sensing devices. A variety of small sensors based on Micro-Electrical-Mechanical Systems (MEMS) are available for attitude and rate of rotation sensing. The most common are magnetometers, accelerometers, and gyroscopes.

2.1.1 Magnetometers

Miniature magnetometers can generate measurements by taking advantage of a number of different types of magnetic effects. Magnetostrictive magnetometers take advantage of the fact that certain ferromagnetic materials undergo an elongation or contraction along an axis parallel to a magnetic field and an opposite deformation along in the transverse direction, which can be sensed through the deflection of an attached cantilever [1]. Lorentz-force magnetometers measure the magnetic force on a current through a suspended aluminum bar, which causes the bar to vibrate in its fundamental mode [1]. Anisotropic magnetoresistive sensors utilize a thin film that changes resistance in the presence of a magnetic field [2]. Magnetoinductive magnetometers measure the inductance of a solenoid whose core changes permeability with a magnetic field [2].

2.1.2 Accelerometers

Micromachined accelerometers are produced in great volume for a wide variety of industrial and consumer applications. They are generally constructed of a suspension beam with an effective spring constant attached to a proof mass, whose position can be determined by measuring characteristics such as 1) Varying resistivity of silicon piezoresistors in the suspension beam as it is stressed; 2) Varying capacitance between the proof mass and an electrode as the mass moves; and 3) Varying quantum-mechanical tunneling current between the proof mass and an electrode as the mass moves [3]. Capacitive sensors are among the most popular, since they have high sensitivity and low power dissipation, and are offered with interface circuitry on a single integrated chip [3].

2.1.3 Gyroscopes

Most micromachined gyroscopes sense rotation by vibrating a mechanical element in one mode and then detecting the transfer of energy to other modes due to the Coriolis acceleration. For example, in a tuning-fork vibratory gyroscope, the tines are driven at a fixed amplitude, and the Coriolis force produces a bending of the tines or rotation around the stem of the tuning fork. Capacitive, piezoresistive, or piezoelectric sensors can be used to measure vibrations in these modes [3].

2.2 Sensor Fusion

Choosing the best attitude estimate for the flyer based on noisy sensor measurements is a nontrivial problem. One widely-used approach for modeling state dynamics and measurements is Kalman filtering.

2.2.1 Kalman Filters

Kalman filters are a method for estimating a dynamic state using indirect measurements [4]. At every discrete time step, two operations are performed: first, the current state of the system is predicted from the last state based on a process model, and then the state is corrected based on a measurement. Kalman filters explicitly take noise into account, and adjust the influence of various measurements based on their statistical properties. Both the mean vector and covariance matrix for the state are recorded at each step.

2.2.2 Euler-Rodrigues Symmetric Parameters

The state of the Kalman filter must be represented internally by some set of coordinates. One common parameterization of attitude is Euler-Rodrigues Symmetric Parameters, usually represented as a four-dimensional quaternion with a normalization constraint to give three degrees of freedom [5], [6]. Quaternions are a four-dimensional extension of complex numbers, and are used here to represent a scalar and a three-dimensional vector.

$$q = [q_0 \quad \rho] \qquad \rho = [q_1 \quad q_2 \quad q_3] \tag{1),(2}$$

Quaternions have a number of advantages over traditional Euler rotation angles. The most important for this application is that they do not have singularities under translation (Euler angles are degenerate every 2π) and do not require the use of transcendental functions [7], which are very expensive on microprocessors; the cosine function, for example, can take up to 3200 cycles [8].

The time-evolution equation for quaternions, given a rotation vector $\omega(t)$, is given in [9]:

$$\dot{q} = \frac{1}{2}\Xi(q) * \omega = \frac{1}{2}\Omega(\omega) * q \tag{3}$$

where

$$\Xi(q) = \begin{bmatrix} -\rho^T \\ q_0 * I_{3x3} + [\rho\times] \end{bmatrix} \qquad \Omega(w) = \begin{bmatrix} 0 & -\omega^T \\ \omega & -[\omega\times] \end{bmatrix} \tag{4),(5}$$

and the cross-product matrix is given by

$$[a\times] = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{6}$$

The measurement model for the magnetometer and accelerometer (when stationary) amounts to a rotation of a reference vector (magnetic or gravitational field direction) into the frame of the sensors, given in [10]:

$$x_{meas} = A(q) * x_{ref} \tag{7}$$

where the rotation matrix is defined by

$$A(q) = \Xi^T(q) * \Psi(q) \qquad \Psi(q) = \begin{bmatrix} -\rho^T \\ q_0 * I_{3x3} - [\rho\times] \end{bmatrix} \qquad (8),(9)$$

2.2.3 SR-UKF

Since the classic Kalman filter assumes that the process and measurement equations are linear in the state variables, a generalization of the Kalman filter must be used that can account for the non-linear quaternion measurement model. One approach is to approximate the non-linearities to first order using an Extended Kalman Filter (EKF), but this approach has been mostly replaced by sigma-point methods such as the Unscented Kalman Filter (UKF) [11]. Sigma-point filters pass a set of points representing the input distribution through the non-linear functions, and then approximate the output statistics. The UKF is accurate to third order for any nonlinearity, but only requires computational resources on par with the EKF [12].
Figure 2 gives an example of poor modeling by an EKF. The mean and covariance calculated by the sigma-point filter (green) much more closely approximate the actual statistics (black) than those of the EKF (pink).

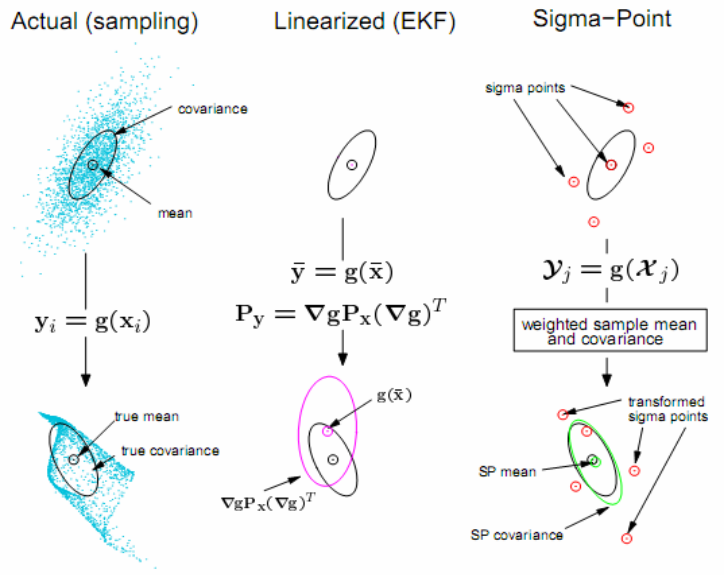

Figure 2: Comparison of EKF and UKF [12]

Due to numerical round-off errors, it is possible for the state covariance matrix to cease to be positive definite, causing the UKF algorithm to fail when taking a square root of the covariance to calculate the sigma points. To prevent this possibility, the square root of the covariance matrix can calculated directly during every time step, without using the actual covariance matrix. The efficient implementation of van der Merwe and Wan makes use of Cholesky factor updating, allowing for better performance than even a normal UKF [11]. The equations from van der Merwe and Wan's paper are described below.
The filter is initialized with a mean vector and the square root of a covariance.

$$\hat{x}_0 = \mathrm{E}[x_0] \qquad S_0 = chol\{\mathrm{E}[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T]\} \qquad (10),(11)$$

The Cholesky factorization decomposes a symmetric, positive-definite matrix into the product of a lower-triangular matrix and its transpose. This triangular matrix is used directly to calculate the sigma points:

$$\chi_{k-1} = [\hat{x}_{k-1} \quad \hat{x}_{k-1} + \eta S_k \quad \hat{x}_{k-1} - \eta S_k] \qquad (12)$$

The scaling constant η is calculated from

$$\eta = \sqrt{L\alpha^2} \qquad (13)$$

where α is a tunable parameter less than one. The sigma points are then passed through the non-linear process model, which predicts the current attitude based on each sigma point and the last command from the flyer's control system, u_{k-1}.

$$\chi_{k|k-1} = F[\chi_{k-1}, u_{k-1}] \qquad (14)$$

The estimated mean and square root covariance are calculated from the transformed sigma points using

$$\hat{x}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \chi_{i,k|k-1} \qquad S_k^- = qr\{[\sqrt{W_1^{(c)}}(\chi_{1:2L,k|k-1} - \hat{x}_k^-) \quad \sqrt{R^v}]\}$$

$$S_k^- = cholupdate\{S_k^-, \chi_{0,k} - \hat{x}_k^-, W_0^{(c)}\} \qquad (15),(16),(17)$$

where

$$W_0^{(c)} = 2(1 - \alpha^2 + \frac{1}{2}\beta) \quad W_0^{(m)} = 1 - \alpha^2 \quad W_i^{(m)} = W_i^{(c)} = \frac{1}{2L\alpha^2} \quad (18),(19),(20)$$

The parameter α is the same as above, and β is another tunable parameter used to incorporate prior knowledge of the state distribution (β=2 is optimal for Gaussian distributions). The matrix R^v is the process noise covariance. The QR factorization decomposes a matrix into the product of an orthogonal matrix and a triangular matrix; only the triangular matrix is used here. Since the zero weight may be negative, the separate Cholesky update operation is needed; the Cholesky update operation efficiently transforms the Cholesky decomposition of the matrix A into the Cholesky decomposition of the matrix $A + x^T x$, where x is a row vector.

The transformed sigma points are then used to predict what measurements the sensors will make, using the nonlinear measurement model:

$$\mathrm{Y}_{k|k-1} = H[\chi_{k|k-1}] \qquad (21)$$

The expected measurement $\hat{y}_k^-$ and square root covariance of $\tilde{y}_k = y_k - \hat{y}_k^-$ (the difference between the actual and expected measurements, also called the innovation) are given by the unscented transform equations just as for the process model:

$$\hat{y}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \mathrm{Y}_{i,k|k-1} \qquad S_{\tilde{y}_k} = qr\{[\sqrt{W_1^{(c)}}[\mathrm{Y}_{1:2L,k} - \hat{y}_k] \quad \sqrt{R_k^n}]\}$$

$$S_{\tilde{y}_k} = cholupdate\{S_{\tilde{y}_k}, \mathrm{Y}_{0,k} - \hat{y}_k, W_0^{(c)}\} \qquad (22),(23),(24)$$

In order to determine how much to adjust the predicted mean and covariance based on the actual sensor input, the Kalman gain matrix K_k is calculated:

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} [\chi_{i,k|k-1} - \hat{x}_k^-][\mathrm{Y}_{i,k|k-1} - \hat{y}_k^-]^T \qquad (25)$$

$$\mathrm{K}_k = (P_{x_k y_k} / S_{\bar{y}_k}^T) / S_{\bar{y}_k} \qquad (26)$$

Note that $S_{\bar{y}_k}$ is square and triangular, so efficient back-substitutions can be used rather than a costly matrix inversion. Finally, the state mean and covariance are updated using the actual sensor input and the Kalman gain matrix:

$$\hat{x}_k = \hat{x}_k^- + \mathrm{K}_k (y_k - \hat{y}_k^-) \qquad (27)$$

$$U = \mathrm{K}_k S_{\bar{y}_k} \qquad S_k = cholupdate\{S_k^-, U, -1\} \qquad (28),(29)$$

3. ATTITUDE SENSOR SELECTION AND INTEGRATION

In this project, two categories of sensors have been selected. A magnetometer and accelerometer are used to give an absolute orientation estimate by measuring the direction of the magnetic field and gravitational field vectors, and three one-axis gyroscopes are used to estimate relative changes in orientation between updates.

Sensor Type	Sensor Name	Voltage (V)	Typical Current (mA)	Units/LSB	Interface
Magnetometer	MicroMag3	3 or 5	<.5	.032 μT	SPI
Accelerometer	SCA3000-E04	3 ± .6	.12	2 mg	SPI
Gyroscope	MLX90609	5 ± .25	16	.31 °/sec	SPI

Table 1: Sensor specifications

3.1 Magnetometer

PNI Corporation's MicroMag3 magnetometer uses three magneto-inductive sensors, one oriented along each axis [14]. It uses an SPI interface, draws very little current, and has a high maximum resolution.

Communicating with the MicroMag3 requires six wires: the SS line, which must be pulled low to enable SPI communication; the RESET line, which must be pulsed high at the start of each measurement; the MOSI line, which receives an SPI command; the DRDY line, which is raised high by the MicroMag3 after a successful measurement; the MISO line, which outputs the measurement back to the microcontroller; and the SCLK line, which provides timing information for the MOSI and MISO lines [14]. The protocol is summarized in Figure 3.

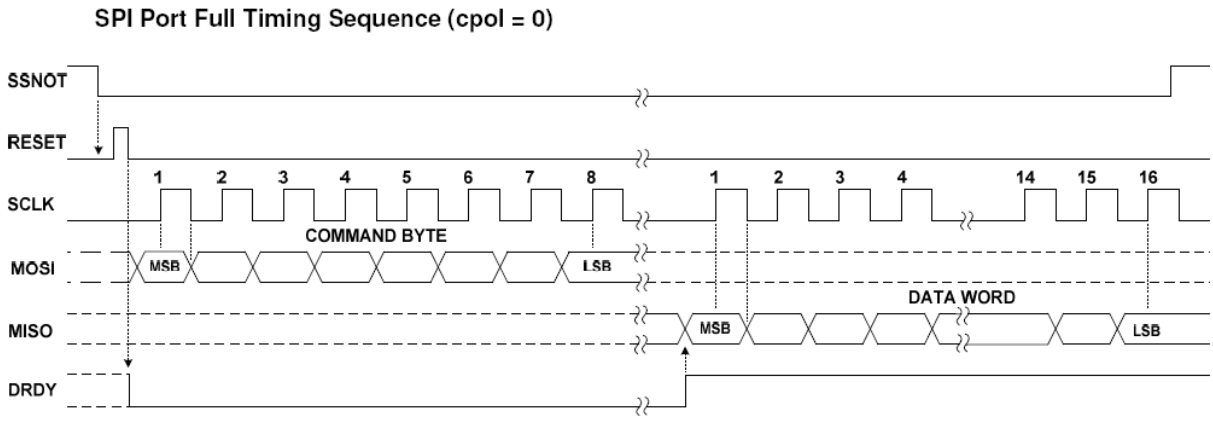

Figure 3: MicroMag3 SPI protocol [13]

The command sent to the MicroMag3 specifies both the axis to measure (X, Y, or Z) and the period division ratio. Magneto-inductive sensors measure the frequency of an LR relaxation oscillator whose inductance depends on the external magnetic field, taking advantage of the nonlinear permeability μ(H) of the core material [14]. The period division ratio determines how long the oscillator is run; measuring more cycles gives a more accurate field measurement, but takes a longer amount of time. A ratio of 1024 was selected, which guarantees that each axis can be measured in less than 15 ms. For non-zero magnetic fields, the actual delay is measured to be approximately 10 ms.

3.2 Accelerometer

VTI Technologies' SCA3000-E04 uses a capacitive sensor to measure the deflection of a proof mass due to acceleration [15]. It features an SPI interface, low power consumption, and high precision of 2 mg steps between ± 6g.

The SPI interface uses five wires: the XRESET line, which must be raised high at power-on (before the first measurement only); the CSB line, which must be pulled low to enable SPI communication; and MOSI, MISO, and SCLK lines as in section 3.1 [16].

3.3 Gyroscopes

Melexis's MLX90609 uses a capacitive sensor to detect secondary mode oscillations due to the Coriolis force from rotation [17]. The R2 version has a measurement range of ± 300 °/sec in increments of 0.3 °/sec, accessible simultaneously over both SPI and an analog voltage output. The SPI interface is the simplest of the sensors, with only four lines: SS, MOSI, MISO, and SCLK as in section 3.1 [18]. Since each gyroscope only measures rotation around one axis, however, three separate MLX90609 sensors were needed. Also, an interface circuit had to be constructed to allow these 5V sensors to share an SPI bus with the 3.3V sensors (see section 3.5).

3.4 SPI Time Division

In order to achieve the design goal of speeds on the order of 100 Hz, measurements and computation had to be appropriately arranged in time. Since each axis of the magnetometer takes approximately 10 ms to measure, only one axis was measured per SR-UKF cycle. The actual filter processing takes place while waiting for each magnetometer measurement to complete. Figure 4 shows the SPI inputs and outputs as measured by an oscilloscope.

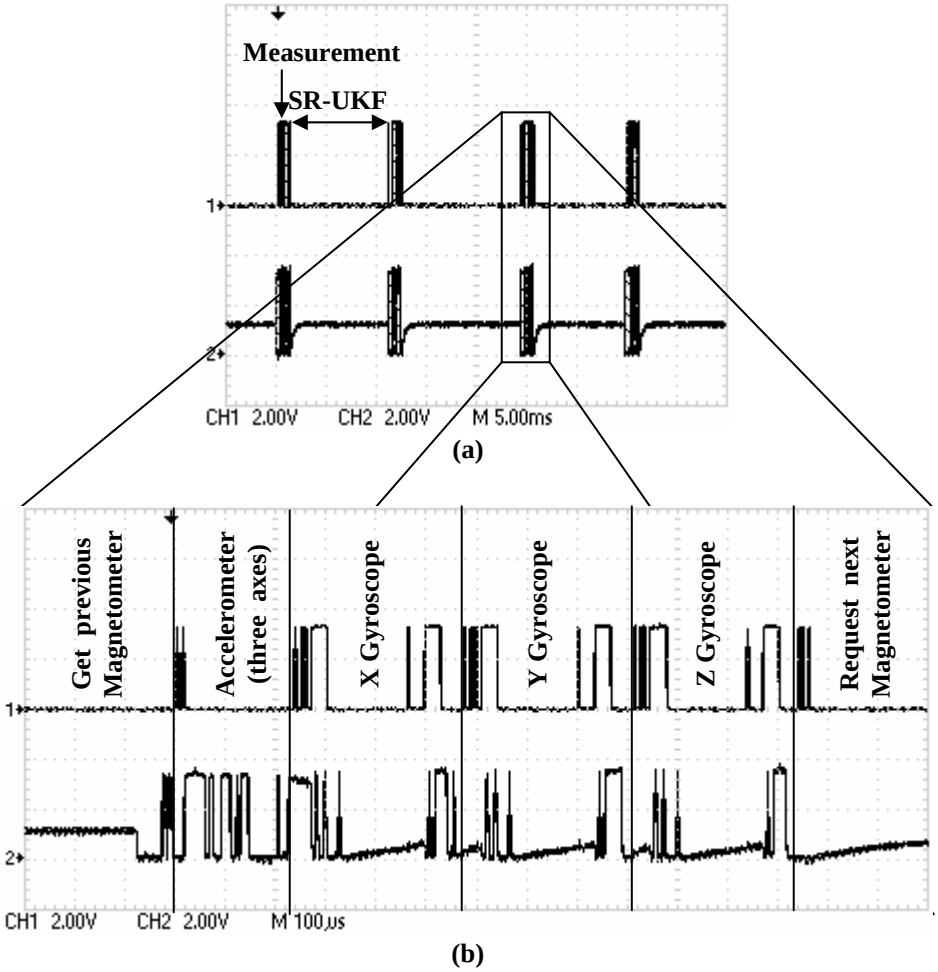

Figure 4: SPI MOSI (master output, top) and MISO (master input, bottom) during filter operation

3.5 Final circuit with UART

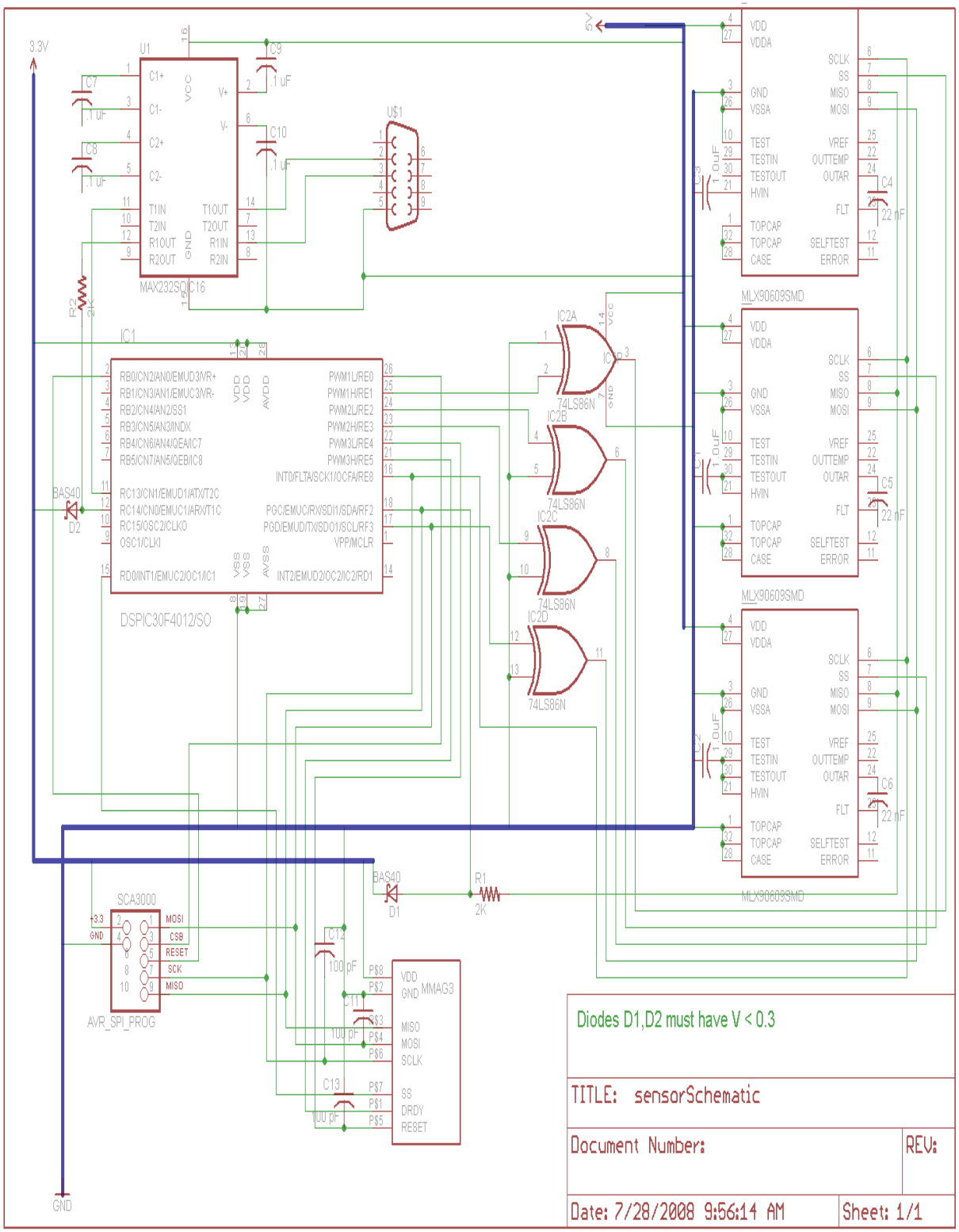

Figure 5: Full circuit

The final circuit is shown in Figure 5. The SPI lines for each sensor are controlled by the dedicated SPI ports on the dsPIC, and the other lines (SS, XRESET, etc.) are controlled by the general I/O pins. Some noteworthy elements of the circuit:

- The MLX90609 gyroscopes run on 5V, rather than 3.3V like the other sensors. The outputs from the microcontroller to the gyroscopes – the three SS lines and the MOSI line – are stepped up using XOR gates with one input grounded. These act level shifters, recognizing 3.3V as a logical high and thus mapping 0V → 0V and 3.3V → 5V. No level shifting is required for the SCLK line, since the gyroscopes' logical high threshold for the clock line is only 2.8V (rather than 3.5V as on the other lines).
- To prevent the gyroscopes from driving the output line (MISO) to 5V, potentially damaging the 3.3V sensors and microcontroller, a diode to 3.3V was added such that the excess voltage is dropped across a 2K resistor before reaching the 3.3V components.
- A universal asynchronous receiver/transmitter (UART) interface allows the microcontroller to communicate with a PC over RS-232 using a serial cable. The RS-232 standard defines voltages between -3V and -25V as logical high ("marking") and voltages between 3V and 25V as logical low ("spacing") [18]. Therefore, an additional chip (the MAX232ACSE) must be used to convert the 0-3.3V signal from the dsPIC's UART port into an RS-232-compatible signal; the MAX232 is able to generate voltages of ±10V with only a +5V source line, using dual charge-pump DC-DC converters [19]. Once the cable load for this application is attached, the actual signals transmitted are measured to be about ±7V (well within the specification). Although this is a 5V chip, it recognizes signals greater than 2V as logical high so no interface circuitry is needed for the transmission line. A diode/resistor pair is added to the receiver line as on the gyroscopes' MISO line to prevent damage to the dsPIC.

A photograph of the final circuit is shown in Figure 6.

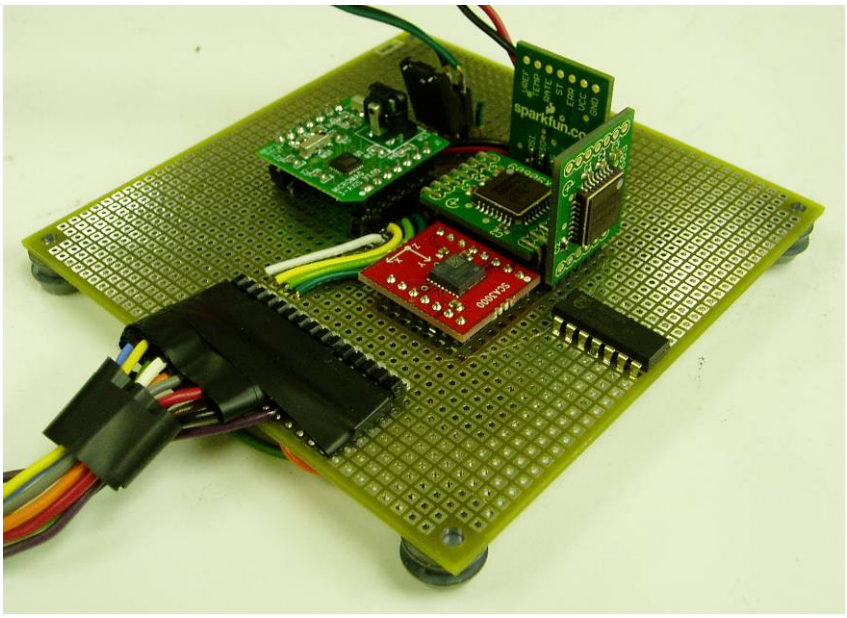

Figure 6: Photograph of Sensor Board

4. SRUKF IMPLEMENTATION

4.1 MATLAB Prototyping and GUI

In order to develop and test the SR-UKF, the filter was implemented in MATLAB. Sensor data were streamed from the dsPIC to the PC over a RS-232 serial cable, and the processing was performed in real time on the PC.

A number of visualizations were developed to aid in the debugging process. Figure 7 shows the four most commonly used windows. Three of the windows show the data being output by the dsPIC compared to the result of the measurement prediction $\hat{y}_k^-$ in the SR-UKF (see section 5.2 for analysis). The lower-right window shows a 3D view of the current attitude estimate, in which the solid blue, green, and red lines show the current orientations of the X, Y, and Z axes (respectively) and the dotted lines show the reference (unrotated) orientation.

In addition to the real-time GUI, the MATLAB program logs the input data, covariance matrices, Kalman gain matrices, and filter output for each step, allowing for offline analysis. See Appendix B for a complete listing of the MATLAB code, with annotations.

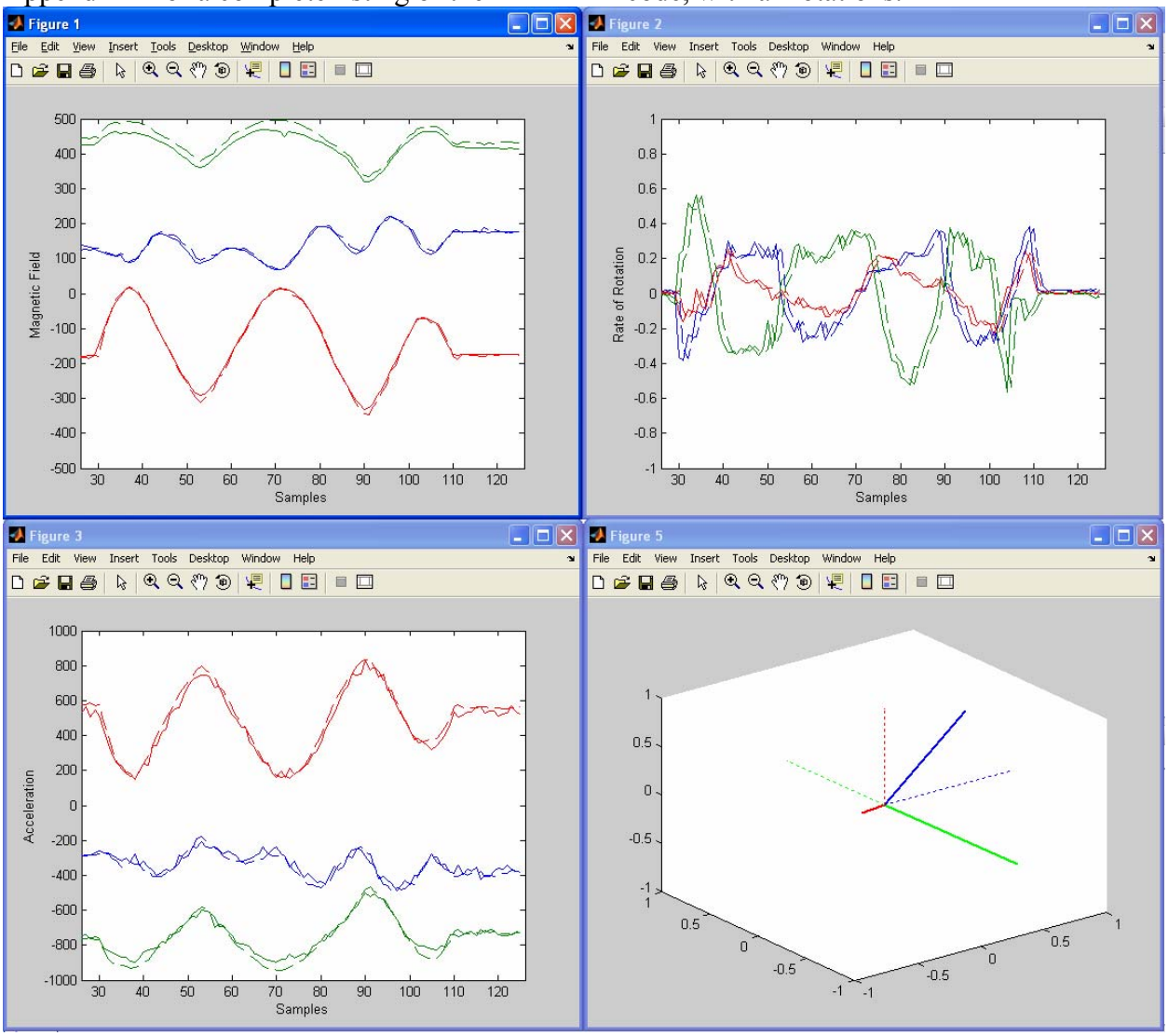

Figure 7: MATLAB GUI for SR-UKF Testing

4.2 dsPIC Implementation

An attempt to implement the filter on the dsPIC30F4012 was made using the Embedded MATLAB Coder (EMLC) from the MATLAB Real-Time Workshop. The EMLC allows for the automatic translation of MATLAB files (meeting certain prerequisites) into C files suitable for execution on an embedded system. In addition to a C file containing the converted MATLAB code and C versions of all necessary matrix functions, the EMLC generates header files that provide wrappers for the specific data types available on the dsPIC.

Although the SR-UKF code was converted correctly, it failed immediately upon execution due to a stack overflow. The available RAM for the dsPIC30F4012 is only 2KB (data space 0x0800 to data space 0x1000) [20]. The required matrices for the SR-UKF alone, however, consume over 3KB of memory (see Table 2) since every floating-point number uses 4 bytes.

Dynamic allocation (not implemented by the EMLC) would allow for a reduction in the amount of memory required, since not all the matrices are required simultaneously. Even with this approach, however, the memory consumed by the required matrices still exceeds 2KB, as shown in Table 3; this peak memory usage occurs just before the calculation of the Kalman gain matrix K. Note that these calculations are underestimates, since they do not take into account stack overhead from function calls or temporary variables.

Matrix Type	Bytes per Matrix	Total bytes
1x7: xMeanUT, yMeanUT, state, data	28	112
15x1: Wm, Wc	60	120
7x7: S, Sy, Q, R, Pxy, K, U	196	1372
15x7: xSigmaPts, xSigmaPtsUT, ySigmaPtsUT	420	1260
21x7: Input to QR function	588	588
	Total Memory Required:	**3452**

Table 2: SR-UKF Memory Requirements without dynamic allocation

Matrix Type	Bytes per Matrix	Total bytes
1x7: xMeanUT, yMeanUT, state, data	28	112
15x1: Wm, Wc	60	120
7x7: S, Sy, Q, R, Pxy	196	980
15x7: xSigmaPtsUT, ySigmaPtsUT	420	840
	Total Memory Required:	**2052**

Table 3: SR-UKF Memory Requirements with Dynamic Allocation

Since the full SR-UKF cannot be run on this dsPIC, a possible compromise is to run a simplified version of the filter. MATLAB tests were performed to determine the feasibility of running the SR-UKF using a constant Kalman gain matrix K and a constant square-root covariance S. Since most of the time- and memory-consuming operations of the filter involve updating these matrices, the simplified filter would be more appropriate for the dsPIC. This approach, however, led to instabilities in the filter and was not implemented. Figure 8 shows an example of these instabilities, in the predicted x measurement of the magnetic field (solid line).

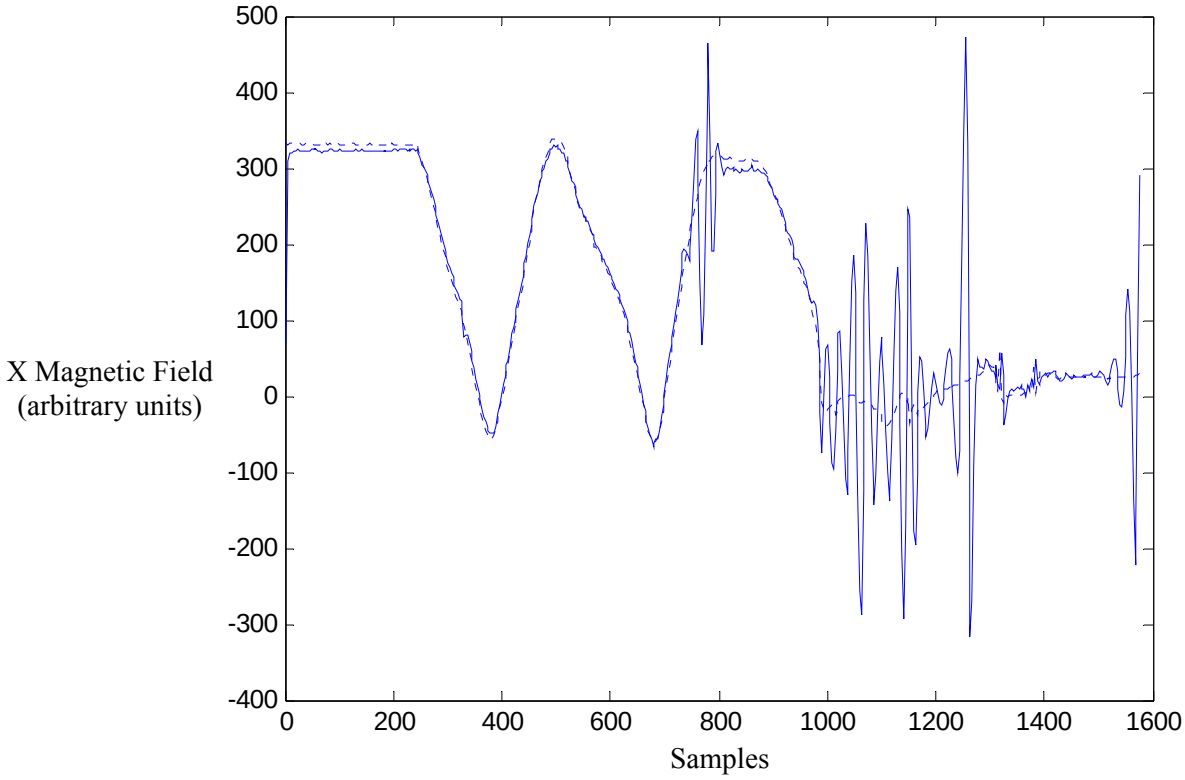

Figure 8: Actual (solid) vs. Predicted (dotted) Measurements for the Constant-Gain SR-UKF

5. RESULTS AND DISCUSSION

5.1 Low-Speed Sensor Tests

The following graphs show typical low-speed sensor measurements taken simultaneously. The sensor board was powered on with the y axis pointing straight up. A series of rotations were performed on the x-axis, then rotations were performed on the z-axis, and finally (after one-half x-axis rotation to reset to the initial position) rotations were performed on the y-axis.

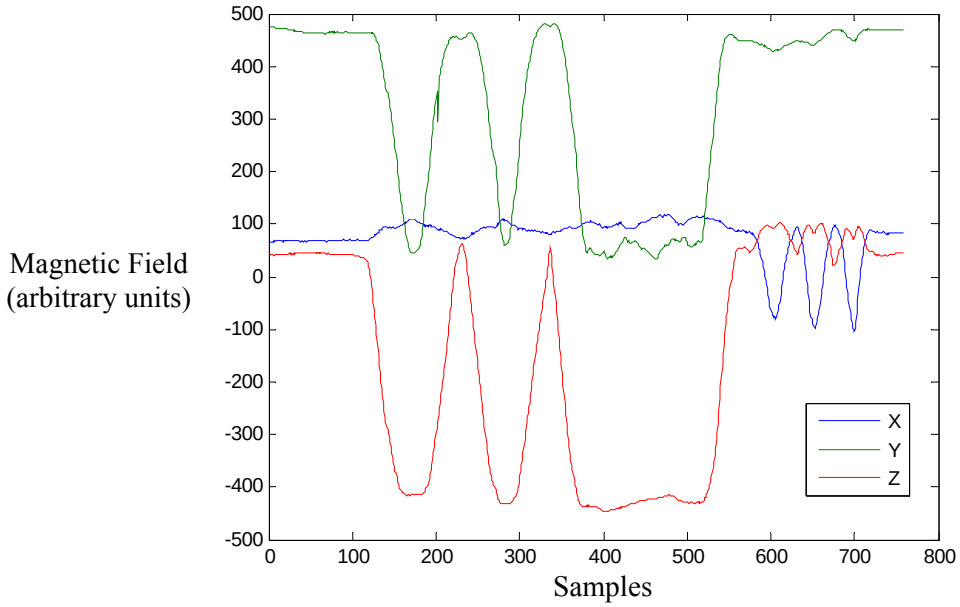

Figure 9: Example MicroMag3 Measurement

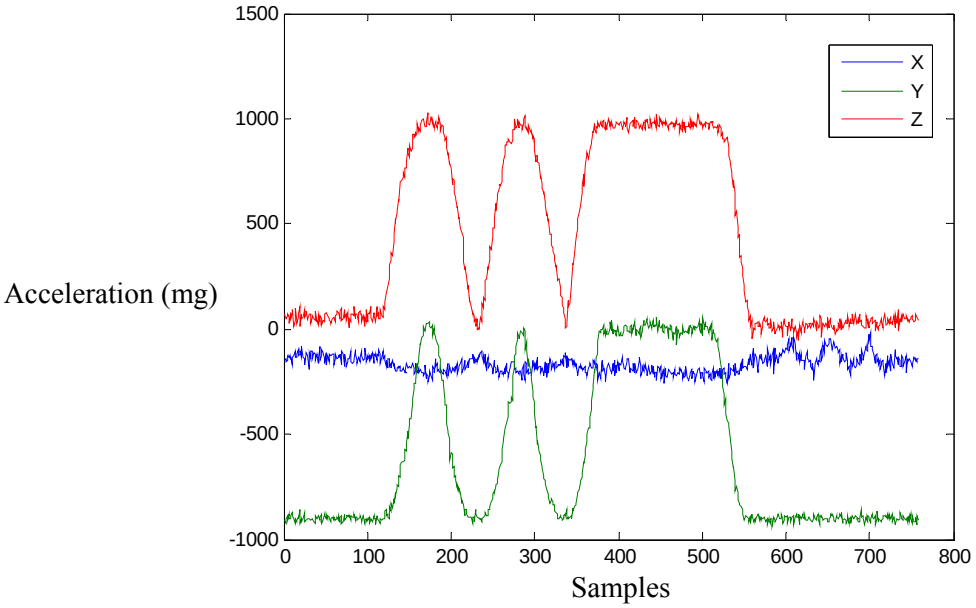

Figure 10: Example SCA3000 Measurement

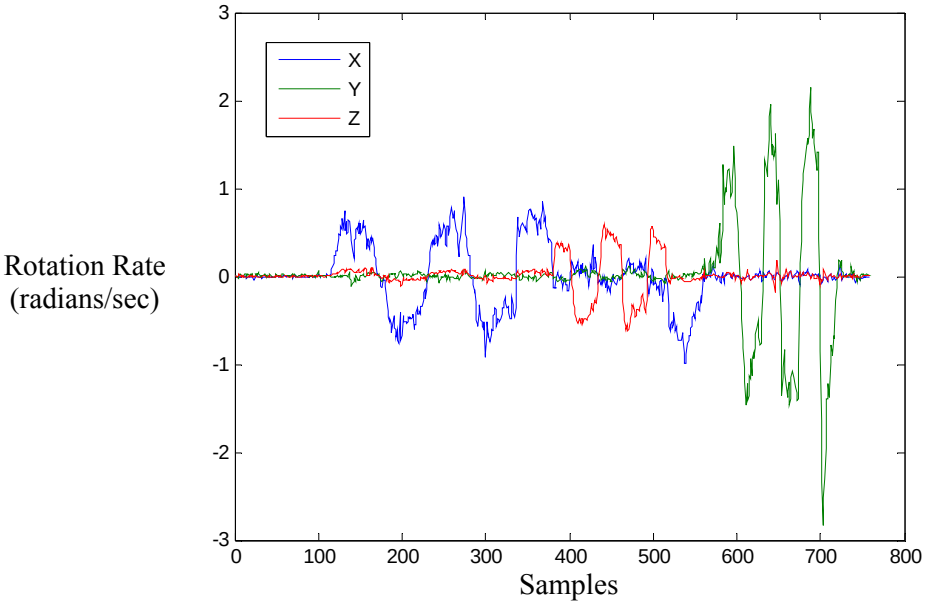

Figure 11: Example MLX90609 Measurement

Although the MicroMag3 takes longer to make a measurement compared to the SCA3000 accelerometer, these data reveal that it has a much higher signal-to-noise ratio. It does have very infrequent spikes (as can be seen on the Y-axis at around sample 200) but these can be easily filtered out.

The variation of the gyroscope measurements from pure sinusoids is not due to measurement errors by the MLX90609, but reflects physical variation in rotation speed (since the test rig was not powered, rotations had to be induced by hand).

5.2 MATLAB Tests

Comparing the actual sensor data and the result $\hat{y}_k^-$ of the measurement prediction step in the SR-UKF allows for verification of the filter in two ways. Showing that the estimated state can generate measurement predictions that closely match all of the sensor data *simultaneously* provides strong evidence that the model parameterization and measurement function are physically realistic. An incorrect measurement function for an SR-UKF with a single sensor might not be detected, since it still could be possible for the filter to match predicted and actual measurements by choosing a different (erroneous) state. As shown in Figure 12 and Figure 13, however, the MATLAB SR-UKF is able to choose a state that matches all six magnetic and acceleration sensors, verifying the correctness of the model (predicted measurements are indicated by solid lines, while actual measurements are shown as dots). See section 5.3 for quantitative analysis of the prediction matching.

Additionally, the time-dependent behavior of the prediction validates the time-update process model. If the SR-UKF were not modeling and correctly incorporating the rotation rate, the predicted measurements would be seen to lag behind the actual measurements since any changes would not be anticipated appropriately. See section 5.3 for quantitative analysis of the prediction delay.

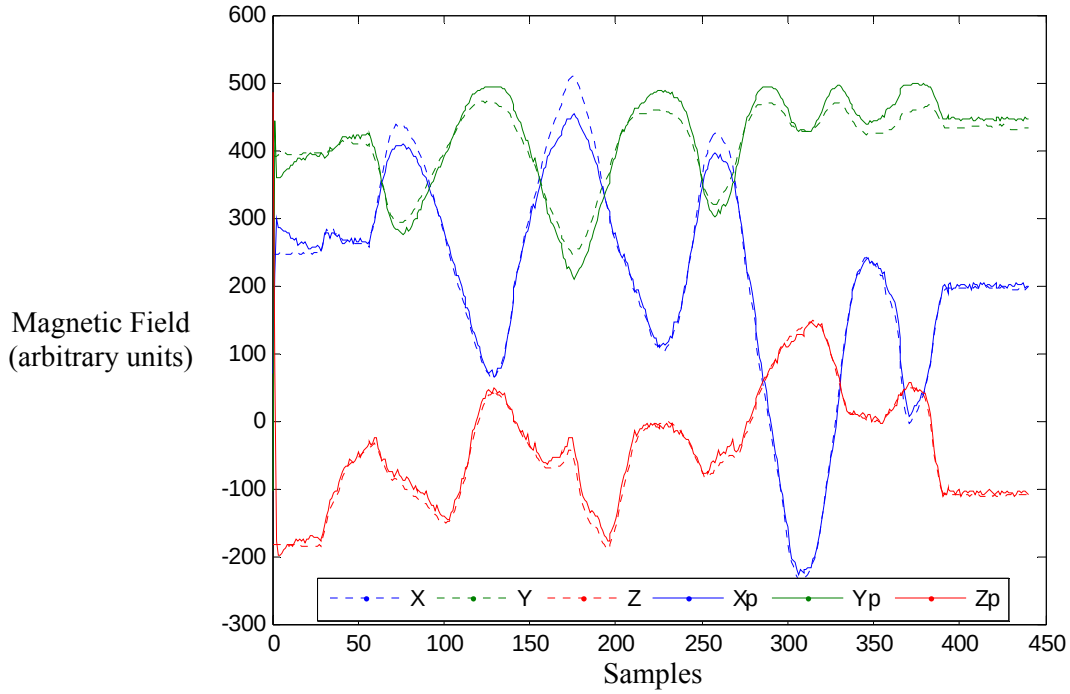

Figure 12: Actual vs. Predicted Measurements for MicroMag3

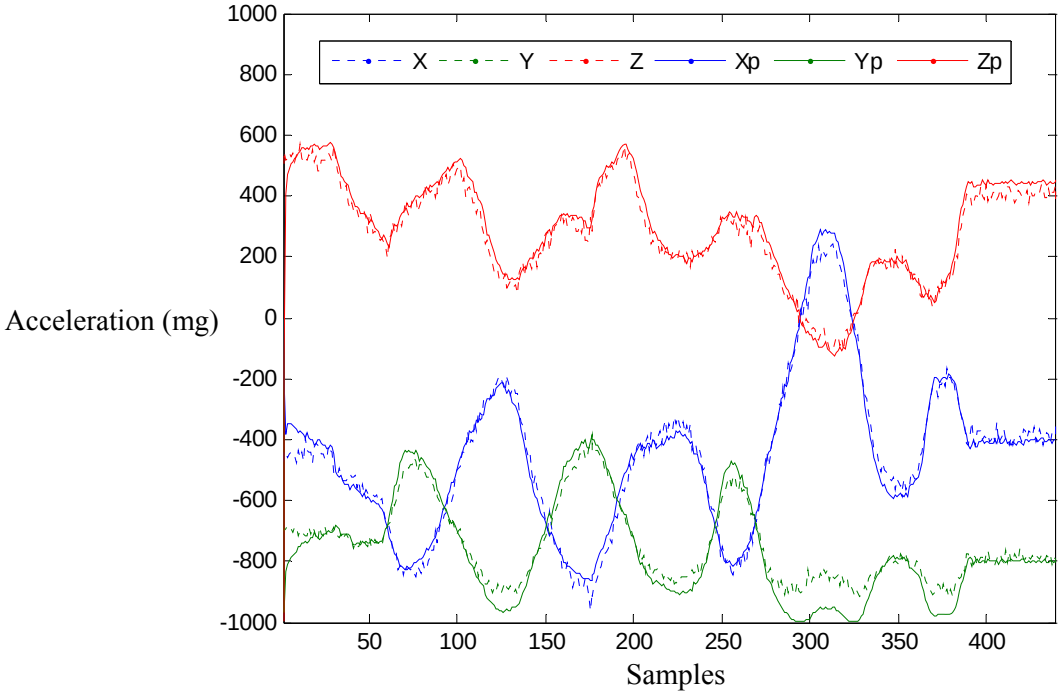

Figure 13: Actual vs. Predicted Measurements for SCA3000

5.3 High-Speed MATLAB Tests

As described in section 3.4, the measurement system was designed to read seven measurements – three gyroscope axes, three accelerometer axes, and one magnetometer axis – at approximately 100Hz. Running at this measurement rate, however, caused the noise on the SCA3000 accelerometer to increase dramatically. The characteristics of this transition were investigated by inserting a delay of varying length before requesting a sample from the accelerometer and then measuring the noise in all three accelerometer axes. The results are shown in Figure 14. Noise was quantified by calculating the root-mean-square error over a window of 50 samples between the measurements and their average values over the window. Note that the noise level drops off sharply at a delay of about 5 ms. Since this is unacceptably long, attempts were made to only query the axes on a rotating basis (much like the magnetometer), but the noise level did not appreciably decrease unless the SPI bus was completely silent for about 5 ms before requesting a sample. The data sheet claims that update rates of up to 93 Hz are possible, but does not indicate whether or not this specification still holds when the SCA3000 is sharing an SPI bus with other sensors [21]. Since this issue could not be resolved, and the noisy accelerometer measurements are not useful for attitude estimation, the high-speed MATLAB tests were conducted using only the magnetometer and gyroscopes.

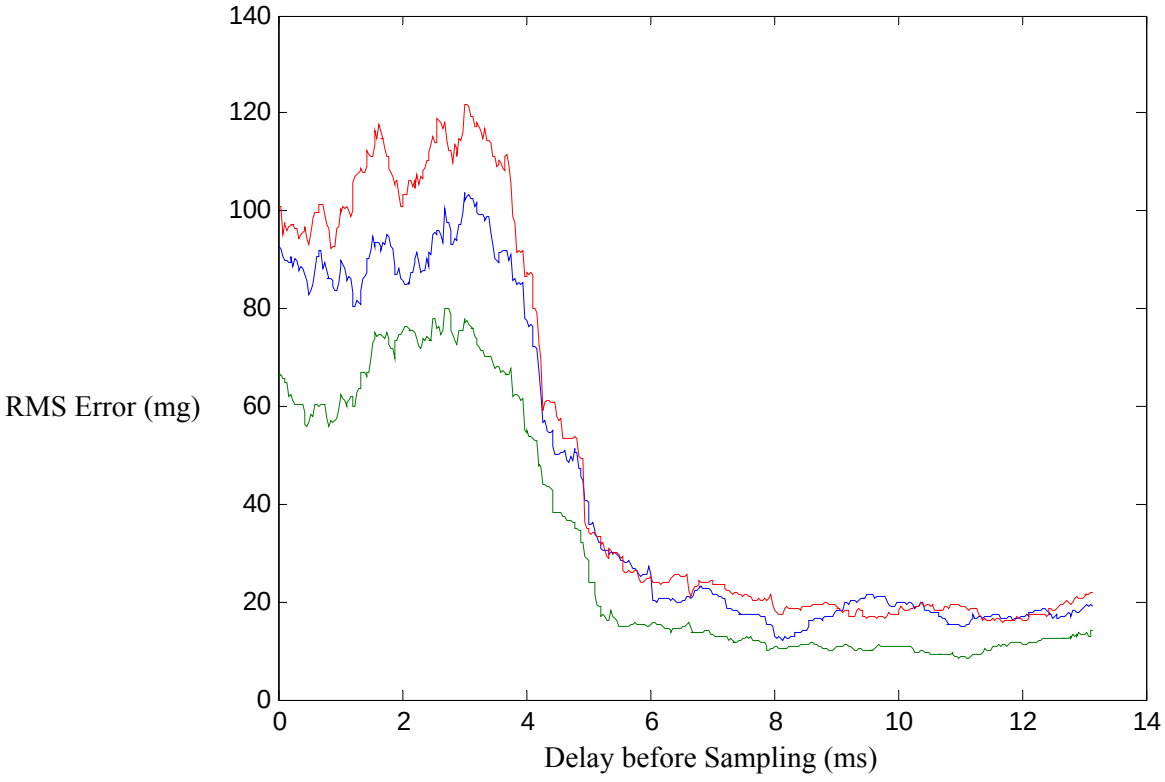

Figure 14: Noise level of SCA3000 Accelerometer vs. Delay before measurement

The MATLAB SR-UKF was able to run in real-time as the data was streamed from the dsPIC at speeds of approximately 90Hz, as long as the GUI was not displayed. Use of the MATLAB profiler showed that the longest single steps in the filter cycle were the time and measurement prediction functions, which involve repeated matrix multiplications.

The match between actual and predicted magnetometer measurements (discussed in section 5.2) was investigated analytically for the high-speed filter. The root-mean-square error between the two plots was less than 16.7 units in each axis, or less than 3.4% of the full magnetic magnitude of 500 units.

To demonstrate that there was no lag between the two plots, the RMS error was evaluated for various horizontal translations of the predicted measurements. Figure 15 shows that no translation improves the match between the actual and predicted measurements, so the estimated attitude does not lag behind the measured attitude. In contrast, Figure 16 shows that there is a definite lag of one cycle for the gyroscope measurements, since changes in rotational velocity are not being modeled. The difference between these plots demonstrates the benefit of explicitly modeling the rotation rates in the attitude state.

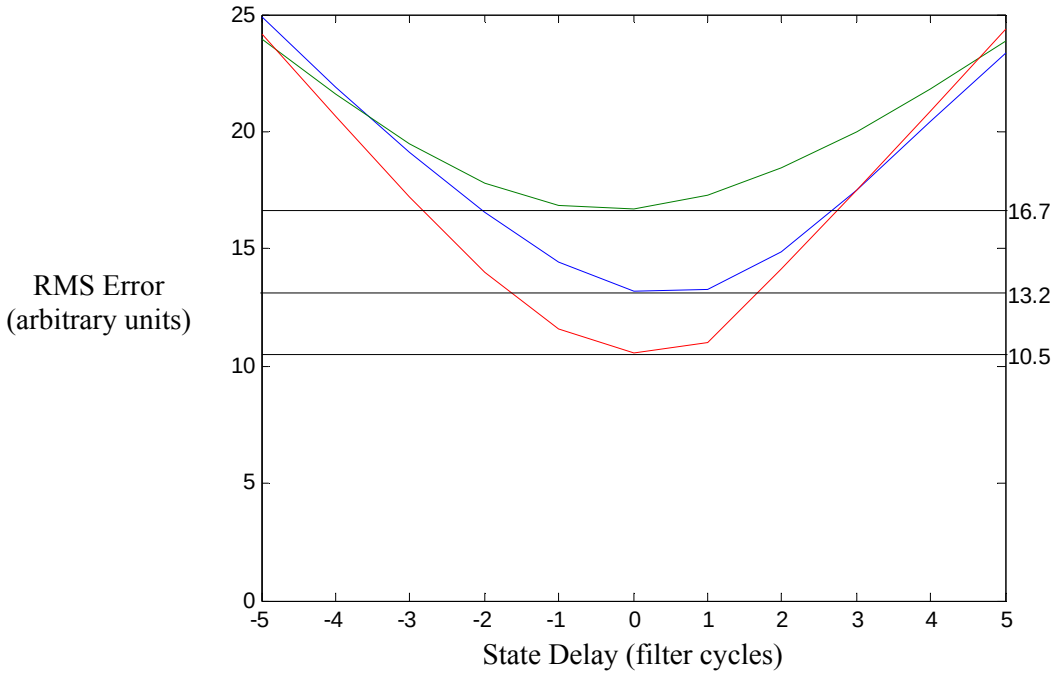

Figure 15: RMS Error between Actual and Predicted Magnetic Measurements vs. Lag Times

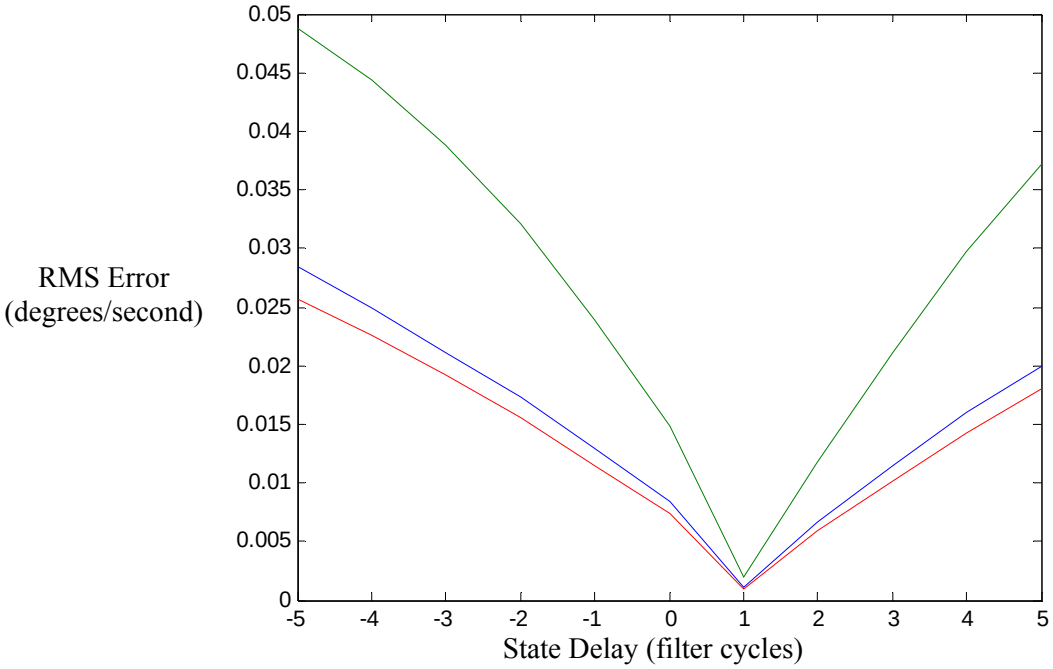

Figure 16: RMS Error between Actual and Predicted Gyroscope Measurements vs. Lag Times

6. FUTURE WORK

A dsPIC-based sensor board and MATLAB implementation of an SR-UKF were successfully completed, but several questions about the sensor system remain unanswered:

- Is the possible to run the filter on a more powerful microcontroller than the dsPIC30F4012? The space requirements for the filter are not unreasonable, but the processor speed necessary to complete a SR-UKF cycle in less than 10 ms has not yet been evaluated. A possible replacement microcontroller is the dsPIC30F6014A, which offers 8KB of RAM and has a similar architecture.
- Should an accelerometer be included in the final design? The SCA3000 can be exchanged with an accelerometer that does not lose reliability at high polling rates, but the usefulness of an acceleration measurement in the highly dynamic environment of the flyer has not been proven.
- How will the magnetometer measurements be affected by the flyer motors? The filter currently relies heavily on the magnetometer for absolute position measurements, but the introduction of magnetic devices nearby may impair its measurement ability. Sensor placement and electromagnetic shielding should be investigated to improve the signal-to-noise ratio.
- Can the performance of the SR-UKF be improved further by incorporating the flyer's control outputs into the process model? This would allow modeling of the rotation rates.

7. ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Mark Yim, for providing me with a challenging project and the resources I needed to complete it. I thank Bill Mather for acting as a sounding board for all of the problems I encountered and for always being available to offer advice. I am grateful to the National Science Foundation and Dr. Jan Van der Spiegel for making SUNFEST possible and giving me the opportunity to be involved in advanced research.

8. REFERENCES

[1] D. K. Wickenden, T. J. Kristenmacher, R. Osiander, S. A. Ecelberger, R.B. Givens, J C. Murphy. (1997). Development of Miniature Magnetometers. *John Hopkins APL Technical Digest* [Online]. 18(2), 271-278. Available: http://www.jhuapl.edu/techdigest/td1802/wickend.pdf

[2] M. J. Caruso, T. Bratland, C. H. Smith, R. Schneider. (1998, Dec.). A New Perspective on Magnetic Field Sensing. *Sensors* [Online]. Available: http://www.sensorsmag.com/articles/1298/mag1298/main.shtml

[3] N. Yazdi, F. Ayazi, K. Najafi. (1998, Aug.). Micromachined Inertial Sensors. *Proceedings of the IEEE* [Online]. 86(8), 1640-1659. Available: http://dx.doi.org/10.1109/5.704269

[4] E. J. Lefferts, F. L. Markley, M.D. Shuster. (1982, Jan.). Kalman Filtering for Spacecraft Attitude Estimation. *AIAA 20th Aerospace Sciences Meeting* [Online]. Available: http://www.aiaa.org/content.cfm?pageid=406&gTable=mtgpaper&gID=90156

[5] M. D. Shuster. (1993, Oct.-Dec.) A Survey of Attitude Representations. *Journal of Astronautical Sciences* [Online]. 41(4), 439-517. Available: http://home.comcast.net/~mdshuster/Pub_1993h_J_Repsurv_scan.pdf

[6] S. Hsieh, L. K. Wang, F. Hsaio, K. Huang, F. Tsai. (2004, March). Airborne Attitude/Ground Target Location Determinations Using Unscented Kalman Filter. *2004 IEEE Aerospace Conference Proceedings* [Online]. 3, 1561-1568. Available: http://dx.doi.org/10.1109/AERO.2004.1367931

[7] J. L. Crassidis, F. L. Markley. (1995). Attitude Estimation Using Modified Rodrigues Parameters. *Proceedings of the Flight Mechanics/Estimation Theory Symposium* [Online]. 419-433. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19960035754_1996055864.pdf

[8] Microchip Technology Inc. PIC24 MCU / dsPIC DSC Math Library [Online]. Available: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en022432

[9] J. L. Crassidis, F. L. Markley, Y. Cheng, "Survey of Nonlinear Attitude Estimation Methods," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 1, pp. 12-28, Jan.-Feb. 2007.

[10] J. L. Crassidis, F. L. Markley, "Unscented Filtering for Spacecraft Attitude Estimation," *Journal of Guidance, Control, and Dynamics*, vol. 26, no.4, pp. 536-542, July-Aug. 2003.

[11] R. van der Merwe and E. A. Wan. (2001, May). The Square-Root Unscented Kalman Filter for State and Parameter-Estimation. *International Conference on Acoustics, Speech, and Signal Processing* [Online]. Available: http://speech.bme.ogi.edu/publications/ps/merwe01a.pdf

[12] R. van der Merwe and E. Wan. (2004, June). Sigma-Point Kalman Filters for Integrated Navigation. *Proceedings of the 60th Annual Meeting of The Institute of Navigation (ION)* [Online]. Available: http://speech.bme.ogi.edu/publications/ps/merwe04a.pdf

[13] PNI Corporation. MicroMag3 Data Sheet [Online]. Available: http://www.pnicorp.com/productDetail?nodeId=cMM3

[14] Zero-offset magnetometer having coil and core sensor controlling period of an oscillator circuit, by T. Hawks. (August 24, 1993). *Patent 5 239 264* [Online]. Available: http://patft.uspto.gov/

[15] VTI Technologies. Why 3D Mems [Online]. Available: http://www.vti.fi/en/products/technology/3d-mems/why-3d-mems/
[16] VTI Technologies. SCA3000 Accelerometer Product Family Specification [Online]. Available: http://www.vti.fi/en/products/accelerometers/consumer_electronics/
[17] Melexis Microelectronic Systems. MLX90609 DataSheet [Online]. Available: http://www.melexis.com/Sensor_ICs_Inertia/General/MLX90609_582.aspx
[18] C. E. Strangio. (2006). The RS232 Standard [Online]. Available: http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html
[19] Maxim Integrated Products. +5V Powered, Multichannel RS-232 Drivers/Receivers [Online]. Available: http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1798
[20] Microchip Technology Inc. dsPIC30F4011/12 Data Sheet [Online]. Available: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010338
[21] VTI Technologies. SCA3000-E04 Data Sheet [Online]. Available: http://www.vti.fi/en/products/accelerometers/consumer_electronics/

Appendix A: dsPIC Source Code

```
#include "p30f4012.h"
/**************
SPI Interface for Sensor Board
Author: Chris Baldassano, chrisb@princeton.edu
Language Toolsuite: Microchip C30
Date: August 2008
Details: Reads data from MicroMag3 magnetometer, 3 MLX90609 gyroscopes,
        and SCA3000 accelerometer and outputs the results over UART.
        For use with MATLAB SR-UKF Filter
Config. bits: Primary Oscillator, FRC w/ PLL 16x

Port map:
        RB0 -> Accelerometer XReset
        RD0 -> Magnetometer SS
        RE0 -> Accelerometer SS
        RE1 -> Z Gyroscope SS
        RE2 -> Y Gyroscope SS
        RE3 -> X Gyroscope SS
        RE4 -> Magnetometer Reset
        RE5 -> Magnetometer DRDY
***************/


// Send a 16-bit integer over UART
void SendIntUART(int toSend);

int main()
{
        int magAxes[3];
        int gyroAxes[3];
        int accelAxes[3];
        int dummy;
        int i;
        long counter;
        char j;
        char magAxis;
        int sign;

        TRISE = 0b000100000;    //Set RE5 as input, all others as output
        PORTE = 0b000001111;   //Set RE0-3 high

        TRISB = 0b000000;               //Set all as output
        PORTB = 0b000000;               //Set all low

        TRISD = 0b00;                   //Set all as output
        PORTDbits.RD0 = 1;              //Set RD0 high

        //Nominally:
                //Int. Osc = 7.37 MHz
                //Fcy = 7.37 * 16 PLL / 4 = 29.48 MHz
        //Measured: Fcy = 31.3344 MHz

        /* UART Setup */
        //BRG = Fcy / (16*Baud) - 1
        U1BRG = 33;                                     //57600 Baud
```

```
U1MODEbits.PDSEL = 0b00;        //8-bit data, no parity bit
U1MODEbits.STSEL = 0b0;                 //1 stop bit
U1MODEbits.ALTIO = 0b1;                 //Use alternate ports (RC13-14)
U1MODEbits.UARTEN = 0b1;        //Enable UART
U1STAbits.UTXEN = 0b1;                  //Enable transmission

/* SPI Setup */
SPI1CONbits.CKE = 0b1;          //Change MOSI on falling CLK edge
SPI1CONbits.MSTEN = 0b1;        //Master mode
//SPICLK = Fcy / (Primary*Secondary)
SPI1CONbits.SPRE = 0b000;       //Secondary = 8
SPI1CONbits.PPRE = 0b01;        //Primary = 16
SPI1CONbits.MODE16 = 0b0;       //8-bit mode
SPI1STATbits.SPIROV = 0b0;      //Reset overflow flag
SPI1STATbits.SPIEN = 0b1;       //Enable SPI


PORTBbits.RB0 = 1;                      //Start up accelerometer
// Loops measured to take .25 uS per count
for (counter = 0; counter < 35*4000; counter++);        //wait 35ms

//Wait for UART character to begin loop
while (!U1STAbits.URXDA);
dummy = U1RXREG;

//Request first magnetometer reading
PORTDbits.RD0 = 0;      //Magnetometer SS
PORTEbits.RE4 = 1;      //Pulse Reset
for(i=0;i<20;i++);              //5 uS pause
PORTEbits.RE4 = 0;      //Clear Reset
for(i=0;i<20;i++);              //5 uS pause

//Request x measurement
SPI1BUF = 0b01010001;
while (!SPI1STATbits.SPIRBF);
dummy = SPI1BUF;
PORTDbits.RD0 = 1;

//Only one magnetometer axis is read per cycle, to increase speed
for (magAxis = 0; magAxis < 3; magAxis = (magAxis + 1) % 3)
{
        //Wait for magnetometer reading to complete
        while (!PORTEbits.RE5);

        //Make accelerometer measurement
        for(i=0;i<20;i++);              //5 uS pause
        PORTEbits.RE0 = 0;
        for(i=0;i<20;i++);              //5 uS pause

        //Request all three axes
        SPI1BUF = 0b00100100;
        while (!SPI1STATbits.SPIRBF);
        dummy = SPI1BUF;

        //Receive and process all three axes
        for (j = 0; j < 3; j++)
```

```
{
        SPI1BUF = 0b00000000; //dummy
        while (!SPI1STATbits.SPIRBF);
        accelAxes[j] = SPI1BUF;
        accelAxes[j] <<= 8;

        SPI1BUF = 0b00000000; //dummy
        while (!SPI1STATbits.SPIRBF);
        accelAxes[j] |= SPI1BUF;

        sign = accelAxes[j] & 0b1000000000000000;
        accelAxes[j] >>= 3;
        accelAxes[j] = (accelAxes[j] & 0b0000111111111111);
        if (sign != 0)
                accelAxes[j] = accelAxes[j] - 4096; //4096 = 2^12

}
PORTEbits.RE0 = 1;


//Receive magnetometer measurement
PORTDbits.RD0 = 0;
for(i=0;i<20;i++);              //5 uS pause
SPI1BUF = 0b00000000; //dummy
while (!SPI1STATbits.SPIRBF);
magAxes[magAxis] = SPI1BUF;
magAxes[magAxis] <<= 8;
SPI1BUF = 0b00000000; //dummy
while (!SPI1STATbits.SPIRBF);
magAxes[magAxis] |= SPI1BUF;
PORTDbits.RD0 = 1;

//Make gyroscope measurements
for (j = 0; j < 3; j++)
{
        if (j==0)
                PORTEbits.RE3 = 0;      //x-axis
        else if (j==1)
                PORTEbits.RE2 = 0;      //y-axis
        else
                PORTEbits.RE1 = 0;      //z-axis

        for(i=0;i<50;i++);              //12.5 uS pause

        //Start ADC
        SPI1BUF = 0b10010100;
        while(!SPI1STATbits.SPIRBF);
        dummy = SPI1BUF;

        //Receive 16 bits (not used)
        SPI1BUF = 0b11111111; //dummy
        while(!SPI1STATbits.SPIRBF);
        dummy = SPI1BUF;
        SPI1BUF = 0b00000000; //dummy
        while(!SPI1STATbits.SPIRBF);
        dummy = SPI1BUF;
```

```
                //wait for measurement
                for(i=0;i<500;i++);                     //125 uS pause

                //Request measurement
                SPI1BUF = 0b10000000;
                while(!SPI1STATbits.SPIRBF);
                dummy = SPI1BUF;
                //Receive measurement
                SPI1BUF = 0b11111111; //dummy
                while (!SPI1STATbits.SPIRBF);
                gyroAxes[j] = SPI1BUF;
                gyroAxes[j] <<= 8;
                SPI1BUF = 0b00000000; //dummy
                while (!SPI1STATbits.SPIRBF);
                gyroAxes[j] |= SPI1BUF;

                PORTE = 0b000001111;

                //Isolate data bits
                gyroAxes[j] >>= 1;
                gyroAxes[j] &= 0b0000011111111111;
        }

        //Request next magnetometer measurement
        PORTDbits.RD0 = 0;      //Magnetometer SS
        PORTEbits.RE4 = 1;      //Pulse Reset
        for(i=0;i<20;i++);      //5 uS pause
        PORTEbits.RE4 = 0;      //Clear Reset
        for(i=0;i<20;i++);      //5 uS pause

        if (magAxis == 2)                       //x-axis
                SPI1BUF = 0b01010001;
        else if (magAxis == 0)      //y-axis
                SPI1BUF = 0b01010010;
        else if (magAxis == 1)      //z-axis
                SPI1BUF = 0b01010011;
        while (!SPI1STATbits.SPIRBF);
        dummy = SPI1BUF;
        PORTDbits.RD0 = 1;


        //Send measurements over UART while waiting for magnetometer
        for (j = 0; j < 3; j++)
        {
                SendIntUART(accelAxes[j]);
        }
        for (j = 0; j < 3; j++)
        {
                SendIntUART(gyroAxes[j]);
        }
        SendIntUART(magAxes[magAxis]);

}

return 0;
```

```
}

void SendIntUART(int toSend)
{
	char UARTchar;
	UARTchar = toSend >> 8;//MSB
	U1TXREG = UARTchar;
	while(U1STAbits.UTXBF==1);
	UARTchar = (toSend << 8) >> 8;    //LSB
	U1TXREG = UARTchar;
	while(U1STAbits.UTXBF==1);
}
```

Appendix B: MATLAB Source Code

```
function x = Embedded_SRUKF_full()
% Square-Root Unscented Kalman Filter for Sensor Board
%   Author: Chris Baldassano
%   MATLAB version: 7.5.0 (R2007b)
%   Based on: "THE SQUARE-ROOT UNSCENTED KALMAN FILTER
%       FOR STATE AND PARAMETER-ESTIMATION" by Rudolph van der Merwe and
%       Eric A. Wan
%   Details: Designed for attitude estimation system with MicroMag3
%       magnetometer, SCA3000 accelerometer, and 3 MLX90609 gyroscopes

% Reset logs
clear global data;
clear global tocLog;
clear global SList;
clear global filtered;
clear global yMeanList;
clear global KList;
global filtered;
global SList;
global KList;
global yMeanList;
global data;
global tocLog;

% Setup serial port
delete(instrfindall);
s = serial('COM1','BAUD',57600);
s.ByteOrder = 'bigEndian';
fopen(s);

% Define initial vectors and matrices, and weights for mean/covariance
stateDim = 7;
measDim = 7;
x = [0;0;0;1;0;0;0];            %initial state
S = chol(.5^2*eye(stateDim))';  %initial state covariance
Q=.05^2*eye(stateDim);          %process covariance
Q(1,1) = .1^2;
Q(2,2) = .1^2;
Q(3,3) = .1^2;
R=200^2*eye(measDim);           %measurement covariance
R(4,4) = .01^2;
R(5,5) = .01^2;
R(6,6) = .01^2;
R(7,7) = 5^2;
sqQ = chol(Q);
sqR = chol(R);
alpha=.5;
beta=2;
Wm=[1-alpha^(2) (alpha^(2))/(2*stateDim)+zeros(1,2*stateDim)];
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);
eta = sqrt(stateDim*alpha^2);

magAxis = 2;
tic;    %start timing
```

```
fwrite(s,'M','char', 'async');  %request measurement
i = 0;
while(1)
    i = i+1;
    while (s.BytesAvailable < 14)   %wait for measurement
    end
    tau = toc;  %compute time elapsed
    tocLog(i) = tau;
    tic;
    currAxes = fread(s,7,'short');

    %data(1:3) -> acceleration
    %data(4:6) -> rotation
    %data(7) -> current axis of magnetic field
    data(i,1) = -2*currAxes(3); %X inverted
    data(i,2) = 2*currAxes(2);
    data(i,3) = -2*currAxes(1); %Z inverted
    data(i,4:6) = ((((25/12)*currAxes(4:6)+400)-2500)*300/2000)*pi/180;
    data(i,7) = currAxes(7);

    %gyroscope calibration (drifts slowly over time)
    data(i,4) = data(i,4) + .01;
    data(i,5) = data(i,5) - .06;
    data(i,6) = data(i,6) + 0.0;

    magAxis = mod(magAxis+1,3); %get current magnetic axis

    %Calculate sigma points
    A = eta*S;
    xSigmaPts = [x x(:,ones(1,stateDim))+A x(:,ones(1,stateDim))-A];

    %Perform state update using unscented transform
    numpts=2*stateDim+1;
    xMeanUT=zeros(stateDim,1);
    xSigmaPtsUT=zeros(stateDim,numpts);
    for k=1:numpts
        xSigmaPtsUT(:,k)=processModel(xSigmaPts(:,k),tau);
        xMeanUT=xMeanUT+Wm(k)*xSigmaPtsUT(:,k);
    end
    [Temp S] = qr([sqrt(Wc(2))*(xSigmaPtsUT(:,2:numpts)-
xMeanUT(:,ones(1,numpts-1))) sqQ]',0);
    if (Wc(1) < 0)
        S = chol(S'*S-(((-1*Wc(1))^(1/4))*(xSigmaPtsUT(:,1)-xMeanUT(:)))*(((-
1*Wc(1))^(1/4))*(xSigmaPtsUT(:,1)-xMeanUT(:)))');
    else
        S = chol(S'*S+((Wc(1)^(1/4))*(xSigmaPtsUT(:,1)-
xMeanUT(:)))*((Wc(1)^(1/4))*(xSigmaPtsUT(:,1)-xMeanUT(:)))');
    end

    %Predict current measurements using unscented transform
    yMeanUT=zeros(measDim,1);
    ySigmaPtsUT=zeros(measDim,numpts);
    for k=1:numpts
        ySigmaPtsUT(:,k)=measurementModel(xSigmaPtsUT(:,k),magAxis);
        yMeanUT=yMeanUT+Wm(k)*ySigmaPtsUT(:,k);
    end
```

```
    [Temp Sy] = qr([sqrt(Wc(2))*(ySigmaPtsUT(:,2:numpts)-
yMeanUT(:,ones(1,numpts-1))) sqR]',0);
    if (Wc(1) < 0)
        Sy = chol(Sy'*Sy - (((-1*Wc(1))^(1/4))*(ySigmaPtsUT(:,1)-
yMeanUT(:)))*(((-1*Wc(1))^(1/4))*(ySigmaPtsUT(:,1)-yMeanUT(:)))');
    else
        Sy = chol(Sy'*Sy + ((Wc(1)^(1/4))*(ySigmaPtsUT(:,1)-
yMeanUT(:)))*((Wc(1)^(1/4))*(ySigmaPtsUT(:,1)-yMeanUT(:)))');
    end

    Sy = Sy';   %Switch to lower triangular matrix

    %Calculate Kalman gain matrix
    Pxy = (xSigmaPtsUT(:,1:numpts)-
xMeanUT(:,ones(1,numpts)))*diag(Wc)*(ySigmaPtsUT(:,1:numpts)-
yMeanUT(:,ones(1,numpts)))';
    K = (Pxy/(Sy)')/Sy;

    %Update state estimate
    x = xMeanUT + K*(data(i,:)' - yMeanUT);
    x = normalizeQ(x);

    %Update state covariance
    upMat = K*Sy;
    for k=1:measDim
        S = chol(S'*S - upMat(:,k)*upMat(:,k)');
    end
    S = S';     %Switch to lower triangular matrix

    %Log data for analysis
    SList{i} = S;
    KList{i} = K;
    filtered(i,:) = x;
    yMeanList(i,:) = yMeanUT;

    %Plot data for GUI
    start = i - 99;
    if (start < 1)
        start = 1;
    end
    start = start - mod(start,3) + 1;
    if (mod(i,10) == 0)
        figure(1);
        plot(start:i,data(start:i,1:3),'-');
        hold on;
        plot(start:i,yMeanList(start:i,1:3),'--');
        axis([start i+1 -1000 1000]);
        hold off;
        xlabel('Samples')
        ylabel('Acceleration')

        figure(2);
        plot(start:i,data(start:i,4:6),'-');
        hold on;
        plot(start:i,yMeanList(start:i,4:6),'--');
```

```
        axis([start i+1 -1 1]);
        hold off;
        xlabel('Samples')
        ylabel('Rate of Rotation')

        figure(3);
        plot(start:3:i,data(start:3:i,7),'-
',start+1:3:i,data(start+1:3:i,7),'-',start+2:3:i,data(start+2:3:i,7),'-');
        hold on;
        plot(start:3:i,yMeanList(start:3:i,7),'--
',start+1:3:i,yMeanList(start+1:3:i,7),'--
',start+2:3:i,yMeanList(start+2:3:i,7),'--');
        axis([start i+1 -500 500]);
        hold off;
        xlabel('Samples')
        ylabel('Magnetic Field')

        figure(5);
        clf;
        axis([-1 1 -1 1 -1 1]);
        disp3d(x);
    end

end

function x = normalizeQ(nonUnit)
%Maintain quaternion normalization
mag = sqrt(nonUnit(4)^2+nonUnit(5)^2+nonUnit(6)^2+nonUnit(7)^2);
x = [0;0;0;0;0;0;0];
x(1:3) = nonUnit(1:3);
x(4:7) = nonUnit(4:7)/mag;

function xp = processModel(x,tau)
%Time-update function
xp = (eye(7) + (tau/2)*...
   [0 0 0 0 0 0 0;...
   0 0 0 0 0 0 0;...
   0 0 0 0 0 0 0;...
   0 0 0 0 -x(1) -x(2) -x(3);...
   0 0 0 x(1) 0 x(3) -x(2);...
   0 0 0 x(2) -x(3) 0 x(1);...
   0 0 0 x(3) x(2) -x(1) 0])*x;

function yp = measurementModel(x,magAxis)
%Measurement prediction functions
yp = [0;0;0;0;0;0;0];
magneticMagnitude = 500;
gravMagnitude = 1000;
currR = RotationMat(x);
yp(1:3) = currR*gravMagnitude*[0 0 -1]';
yp(4) = x(1);
yp(5) = x(2);
yp(6) = x(3);
threeMag = currR*magneticMagnitude*[.1371 -.2059 .9689]';
yp(7) = threeMag(magAxis+1);
```

```
function disp3d(x)
%Show 3D view of attitude estimate
currR = inv(RotationMat(x));
t = currR*[1 0 0]';
line([0 t(1)],[0 t(2)],[0 t(3)], 'Color', 'b','LineWidth',2);
t = currR*[0 1 0]';
line([0 t(1)],[0 t(2)],[0 t(3)], 'Color', 'g','LineWidth',2);
t = currR*[0 0 1]';
line([0 t(1)],[0 t(2)],[0 t(3)], 'Color', 'r','LineWidth',2);

line([0 1],[0 0],[0 0], 'Color', 'b','LineWidth',1,'LineStyle', ':');
line([0 0],[0 1],[0 0], 'Color', 'g','LineWidth',1,'LineStyle', ':');
line([0 0],[0 0],[0 1], 'Color', 'r','LineWidth',1,'LineStyle', ':');


function R = RotationMat(x)
%Calculate rotation between reference frame and state estimate
R = ([x(4)^2+x(5)^2-x(6)^2-x(7)^2,2*(x(5)*x(6)+x(4)*x(7)),2*(x(5)*x(7)-
x(4)*x(6));2*(x(5)*x(6)-x(4)*x(7)),x(4)^2-x(5)^2+x(6)^2-
x(7)^2,2*(x(6)*x(7)+x(4)*x(5));2*(x(5)*x(7)+x(4)*x(6)),2*(x(6)*x(7)-
x(4)*x(5)),x(4)^2-x(5)^2-x(6)^2+x(7)^2]);
```