UNIVERSITY of CALIFORNIA
SANTA CRUZ

ATTITUDE ESTIMATION FOR A LOW-COST UAV

A thesis submitted in partial satisfaction of the
requirements for the degree of

BACHELOR OF SCIENCE

in

PHYSICS

by

Gregory M. Horn

19 March 2009

The thesis of Gregory M. Horn is approved by:

Professor Gabriel Elkaim
Technical Advisor

Professor David P. Belanger
Thesis Advisor

Professor David P. Belanger
Chair, Department of Physics

Abstract

Attitude Estimation for a Low-Cost UAV

by

Gregory M. Horn

Unmanned Aerial Vehicles (UAVs) and Micro Aerial Vehicles (MAVs) are a rapidly growing area of research and development. One of the biggest technical barriers is attitude estimation, which is typically done using microelectromechanical systems (MEMS) inertial sensors and an embedded Kalman filter. This requires either an expensive commercial module or significant knowledge of electronics, attitude dynamics, and Kalman filtering. This thesis presents a sophisticated quaternion-based Kalman filter which is capable of running at 100 Hz in floating point on a 16-bit microcontroller, without sacrificing performance. The whole attitude estimation system is reproducible for around $500 and will be made open source.

This thesis presents the complementary sensor fusion problem and proposes the extended Kalman filter as a solution. The necessary mathematical background in spatial rotations and attitude dynamics is developed and the implementation equations for an extended Kalman filter are derived using the quaternion attitude representation. This algorithm is verified in simulation, resulting in roll, pitch, and yaw RMS errors of 1.03, 0.901, and 2.47 degrees respectively over the simulation of a 10-minute flight.

A custom autopilot board is designed and fabricated, and the attitude estimation algorithm is embedded and shown to be capable of running at 100 Hz in floating point on the 16-bit dsPIC33F (which costs $10 and has no floating point unit). The autopilot board is flown on a remote-control model aircraft under human control and a data set of flight sensor readings is recorded. Although there is no truth data to compare to the attitude estimation output, results indicate that the attitude estimation algorithm works reliably on real flight data.

Autonomous control algorithms have been developed and validated in simulation, and in future research an aircraft will be flown under full autonomous control. When the system is proven, all algorithms, software, and hardware designs will be published to the open source community in order to promote low-cost

UAV research. Any university lab with reasonable technical experience will be able to start a UAV program based on our system for under $1000 including the aircraft.

Contents

List of Figures

List of Tables

For my family. Without your love and support I could not be where I am today.

Acknowledgements

I would like to thank my thesis advisor, Gabriel Elkaim. Gabe introduced me to most of the concepts contained in this thesis. He provided guidance, and allowed me the freedom to pursue the project with my own style. Mariano Lizarraga has put more time into this project than anyone else. He has worked tirelessly, contributing to every facet of the system. Without his work there would be no UAV.

I would also like to thank Bruce Schumm, Ned Spencer, Max Wilder, and all of the other wonderful people at the Santa Cruz Institude for Particle Physics (SCIPP). They introduced me to undergraduate research and taught me the skills in electronics and prototyping which I utilized throughout this project.

1 Introduction

The development and operation of Unmanned Aerial Vehicles (UAVs) have been almost exclusively limited to military (see Figure 1.1) and government (e.g., NASA) applications because of prohibitive cost. In the future, civilian UAVs and and Micro Aerial Vehicles (MAVs) will contribute to society in many ways. Groups are looking into their use for search and rescue, fire fighting, flight safety research, aerial photography, recreation, and more, but before UAVs become commonplace their cost must be reduced significantly.

Figure 1.1: An Air Force RQ-4 Global Hawk UAV

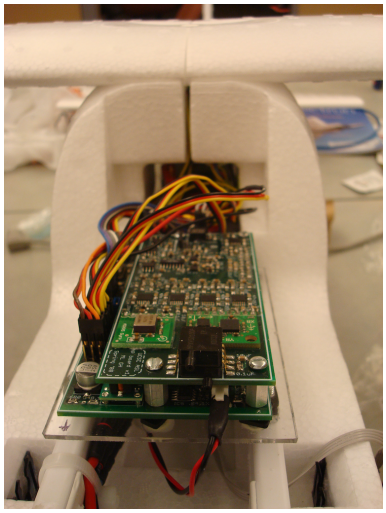
The most critical component in an autopilot is the attitude estimator, which takes sensor readings and extrapolates an aircraft's attitude (e.g., the Euler angles roll, pitch, and yaw). Until recently this required expensive navigation-grade sensors, but advances in commercial microelectromechanical systems (MEMS) sensor technology are providing a low-cost alternative. These MEMS sensors perform poorly compared to their their navigation-grade counterparts but with sophisticated filtering they can be successfully utilized in

an attitude estimator for a small UAV.

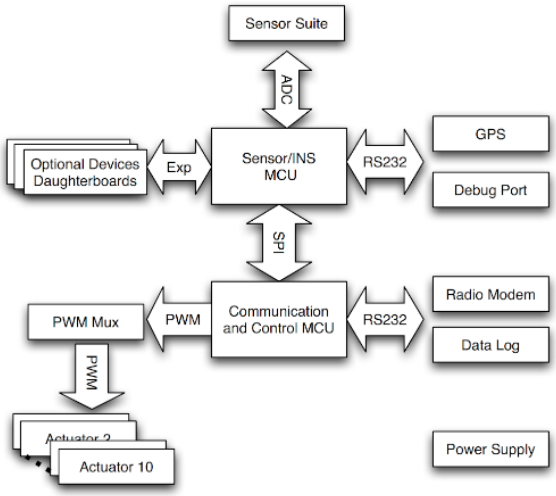
The sensor filtering techniques typically implemented require significant processing power. A compromise must be made between a bulky, power-hungry embedded system like a PC/104, and a small low-power microcontroller which sacrifices attitude estimation performance. This thesis presents a sophisticated attitude estimation algorithm which is capable of running on a 16-bit microcontroller at 100 Hz without sacrificing performance.

1.1 SLUGS

The estimation algorithm was developed for the Santa Cruz Low-cost UAV GNC System (SLUGS) (see Figure 1.2(a)). The goal of the SLUGS project is to provide a versatile, robust autopilot board suitable for research in small UAVs. The sensors utilized by SLUGS are a Global Positioning System (GPS) module, a MEMS barometer, and three axes each of magnetic field sensors, MEMS accelerometers, and MEMS angular rate sensors (rate gyros). SLUGS utilizes two processors: a sensor microcontroller which takes sensors readings and performs attitude estimation, and a control microcontroller which takes the estimated attitude and executes control laws (see Figures 1.2(b) and A.1).



(a) SLUGS mounted in aircraft



(b) SLUGSv1 block diagram

Figure 1.2: The SLUGS autopilot

Both microcontrollers are easily programmable in Simulink (see Figure 1.3) which dramatically reduces development overhead by allowing rapid implementation and reconfiguration of control and estimation architectures. SLUGS will soon be made open source, making UAV research accessible to a broader scientific community as well as the hobbyist community.

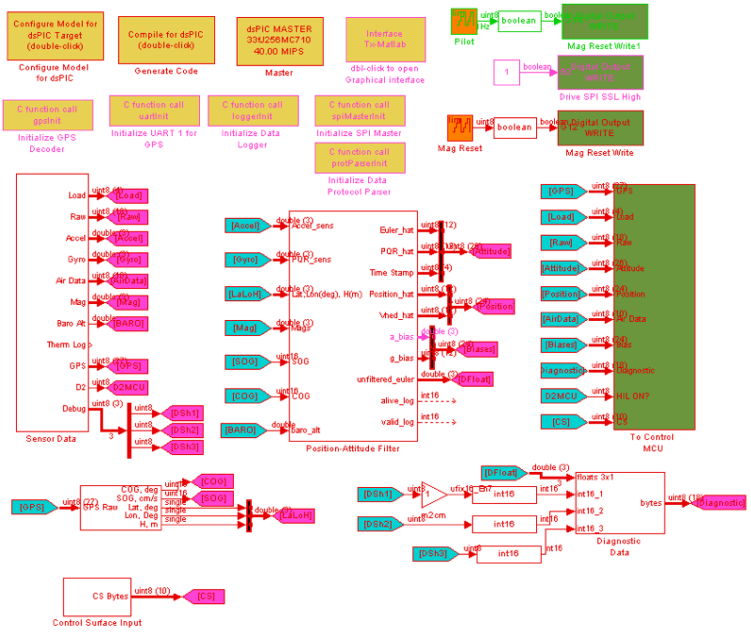


Figure 1.3: Programming the microcontroller in Simulink

The key to making SLUGS successful is having reliable attitude estimation. In Chapter 2 I describe the issues associated with utilizing inexpensive MEMS sensors, and present the Kalman filter as a robust sensor fusion technique. In Chapters 3 and 4 I derive the static and dynamic equations which describe attitude and spacial rotations. In Chapter 5 I implement these equations in a Kalman filter, and in Chapter 6 I present the simulation, benchtop, and flight results of my algorithm.

2 Kalman Filtering

2.1 Statement of the Problem

In two dimensions a frame rotation is performed using the rotation matrix

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}, \tag{2.1}$$

where (u, v) is a vector expressed in frame (x, y), (u', v') is the same vector expressed in frame (x', y'), and α is the angle between the two frames (see Figure 2.1). The angle α contains the information needed to fully specify the attitude of frame (x', y') with respect to (x, y). It should be clear that a counter-clockwise rotation of frame (x', y') with respect to (x, y) is equivalent to a clockwise rotation of vector (u', v') within frame (x', y').

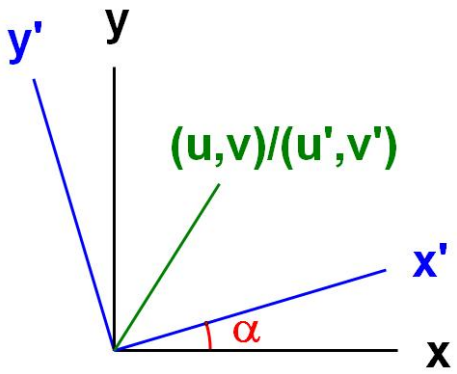


Figure 2.1: Two dimensional vector rotation

2.1.1 Filtering Accelerometers

Because accelerometers sense the acceleration due to gravity which always points down, one might be tempted to write

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} 0 \\ g \end{pmatrix} \tag{2.2}$$

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} g\sin(\alpha) \\ g\cos(\alpha) \end{pmatrix}, \tag{2.3}$$

where $\vec{z}$ is the accelerometer measurement (in frame (x', y')). One could then solve equation (2.3) for α, yielding

$$\begin{pmatrix} \alpha \\ \alpha \end{pmatrix} = \begin{pmatrix} \arcsin\left(\frac{z_1}{g}\right) \\ \arccos\left(\frac{z_1}{g}\right) \end{pmatrix}. \tag{2.4}$$

However, accelerometers measure inertial acceleration in addition to gravity. The actual measured signal would therefore be

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} A_x \\ A_y + g \end{pmatrix}, \tag{2.5}$$

where $[A_x \ A_y]^T$ is the sensor's inertial acceleration in frame (x, y). Whenever the sensor experienced an inertial acceleration, equation (2.4) would be invalid and the computed result for α would be incorrect.

If the expected acceleration in this case $[A_x \ A_y]^T$ was expected to be random and Gaussian, it could be treated as noise and filtered out. Figure 2.2 shows a simulation of a mass on a spring with dampening. The available position measurements are noisy so a low-pass filter is applied. Since a low-pass filter is a weighted average of *past* data, its output tends to lag behind the true state. This phase lag is unacceptable in an autopilot system so another solution must be found.

2.1.2 Integrating Rate Gyros

Another tempting way of tracking α is by integrating a rate gyro output z_ω. One might write

$$z_\omega(t) = \dot{\alpha}(t) \tag{2.6}$$

$$\int_0^t z_\omega(\tau)d\tau = \alpha(t). \tag{2.7}$$

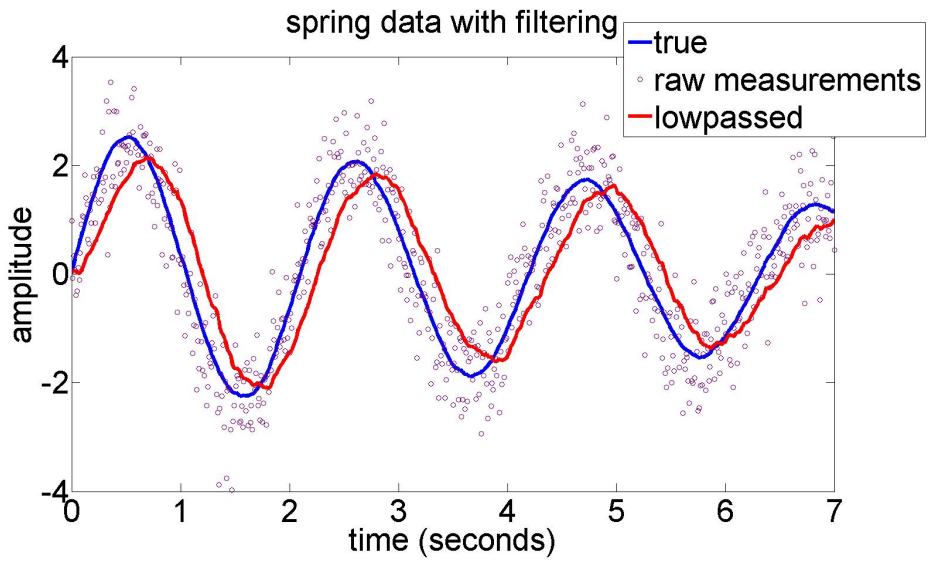


Figure 2.2: Simulation of a spring with dampening. The raw measurements are very noisy so a low-pass filter is applied for smoothing, which causes a phase delay.

However, the gyro output is offset by a small drifting bias, so the true sensor equations are

$$z_\omega(t) = \dot{\alpha}(t) + b(t) \tag{2.8}$$

$$\int_0^t z_\omega(\tau)d\tau = \alpha(t) + \int_0^t b(\tau)d\tau. \tag{2.9}$$

Figure 2.3 shows the same simulation as before, a mass on a spring, but this time a velocity sensor (with an exaggerated bias) is integrated to extract position. Because of the bias an error is integrated over time.

Integrating rate gyros results in an accumulated error, but there is no phase delay. Filtering accelerometers results in no accumulated error but a significant phase delay. The rest of the chapter will discuss how two complementary sensors such as these can be incorporated using a dynamically-weighted recursive least-squares algorithm called a Kalman filter.

2.2 The Observer

An ideal discrete dynamic system is modeled by the difference equation

$$\vec{x}_k = f_{k-1}(\vec{x}_{k-1}), \tag{2.10}$$

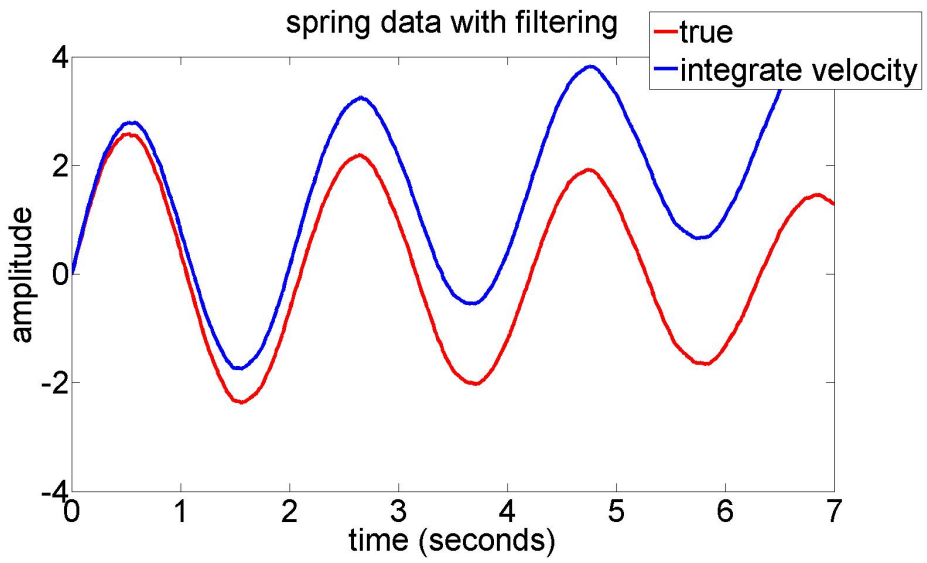


Figure 2.3: Simulation of a spring with dampening. A non-ideal velocity sensor is integrated to extract position. Bias drift on the velocity sensor is also integrated, causing an increasing error on the position estimate.

where $\vec{x}$ is known as the state of the system (usually just called the "state"). In a physical system with disturbances it is more accurate to include a system noise term $\vec{w}$

$$\vec{x}_k = f_{k-1}(\vec{x}_{k-1}) + \vec{w}_{k-1} \tag{2.11}$$

$$\vec{w}_k \sim \mathcal{N}(0, Q_k). \tag{2.12}$$

The notation $\vec{w}_k \sim \mathcal{N}(0, Q_k)$ specifies that $\vec{w}_k$ is randomly distributed with a Gaussian probability distribution of covariance Q_k.

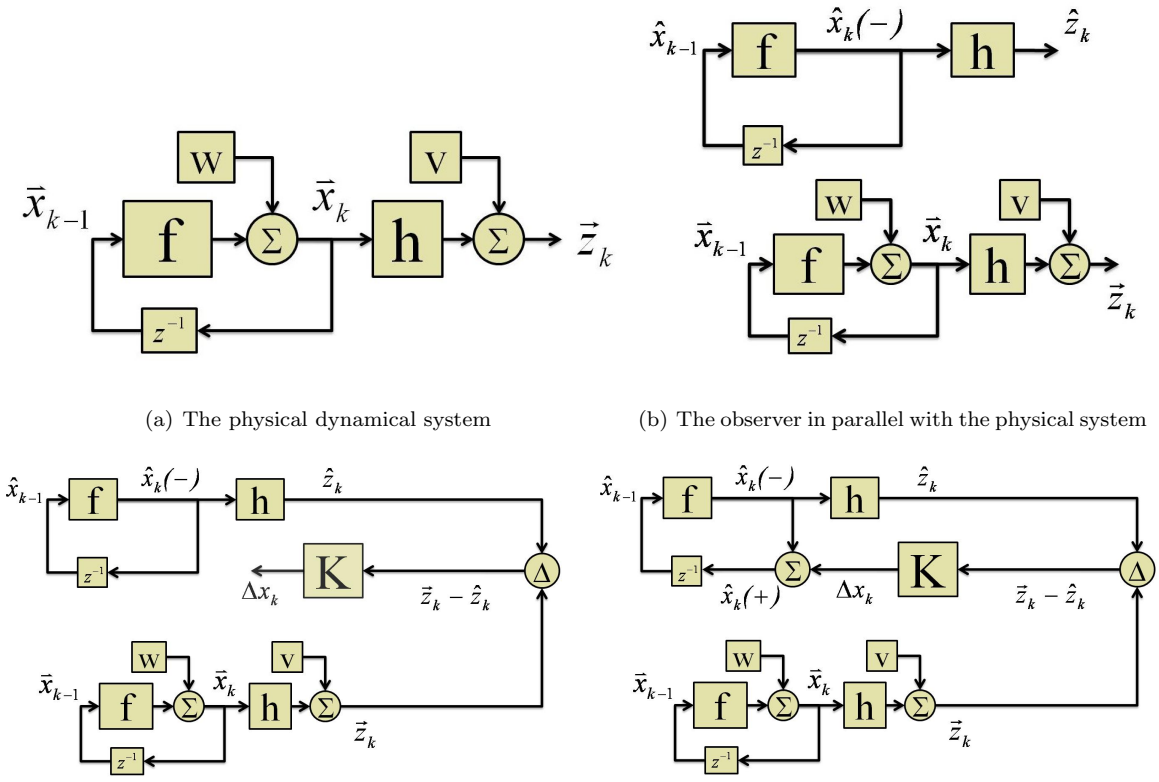
In general, one does not have direct access to the value of $\vec{x}_k$, but must measure it with sensors which are inherently noisy. The measurement equation is written

$$\vec{z}_k = h_k(\vec{x}_k) + \vec{v}_k \tag{2.13}$$

$$\vec{v}_k \sim \mathcal{N}(0, R_k). \tag{2.14}$$

The complete system is represented in block form in Figure 2.4(a). Note that the unit delay operator z^{-1} is standard notation, and should not confused with the sensor measurement.

One way of estimating the state $\vec{x}_k$ is by forming a virtual system called an observer. The observer



(a) The physical dynamical system

(b) The observer in parallel with the physical system

(c) The measurements differ, and a state error is calculated

(d) The state error is used to correct the state estimate

Figure 2.4: Formation of the observer

has virtual state $\hat{x}_k$, virtual measurements $\hat{z}_k$, and obeys the same equations as the real system:

$$\hat{x}_k = f_{k-1}(\hat{x}_{k-1}) \tag{2.15}$$

$$\hat{z}_k = h_k(\hat{x}_k). \tag{2.16}$$

The observer is usually run on a computer in real time, in parallel with the real system (see Figure 2.4(b)).

Because of the system noise $\vec{w}$, the *a priori* propagated state $\hat{x}_k(-)$ differs from the true state $\vec{x}_k$, and virtual measurement $\hat{z}_k$ does not agree with the actual measurement $\vec{z}_k$. Feedback gain K_k is formed to estimate the state error $\Delta\hat{x}_k = \vec{x}_k - \hat{x}_k$ based on the measurement error $\vec{z}_k - \hat{z}_k$ (see Figure 2.4(c)). Finally, this error is used to correct the observer's state, forming the *a posteriori* corrected state $\hat{x}_k(+)$ (see Figure 2.4(d)).

2.3 The Kalman Filter

The expectation value of the state error $\Delta\hat{x}_k$ is known as the covariance of the state, written as

$$P_k = \mathcal{E}([\vec{x}_k - \hat{x}_k][\vec{x}_k - \hat{x}_k]^T). \tag{2.17}$$

By analyzing the system dynamics $f(\vec{x})$, the expected system and sensor noise covariances Q_k and R_k, and the measurement function $h(\vec{x})$, an optimal feedback gain K_k can be derived to minimize P_k. For linear systems this is known as the Kalman filter and was first presented in 1960 by Kalman[1]. The implementation equations for a linear system, adapted from [2] and [3], are summarized in Table 2.1.

System dynamics	$\vec{x}_k = \Phi_{k-1}\vec{x}_{k-1} + \vec{w}_{k-1}$ $\vec{w}_k \sim \mathcal{N}(0, Q_k)$
Measurement	$\vec{z}_k = H_k\vec{x}_k + \vec{v}_k$ $\vec{v}_k \sim \mathcal{N}(0, R_k)$
Covariance	$P_k = \mathcal{E}([\vec{x}_k - \hat{x}_k][\vec{x}_k - \hat{x}_k]^T)$
State propagation	$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+)$
Predicted measurement	$\hat{z}_k = H_k\hat{x}_k(-)$
State covariance propagation	$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1}$
Feedback gain	$K_k = P_k(-)H_k^T(H_kP_k(-)H_k^T + R_k)^{-1}$
State update	$\hat{x}_k(+) = \hat{x}_k(-) + K_k(z_k - \hat{z}_k)$
State covariance update	$P_k(+) = (I - K_kH_k)P_k(-)$

Table 2.1: Discrete Kalman filter implementation equations

The state estimate and covariance are propagated at every time step, but in many systems measurements are only sporadically available. In this case the state covariance grows for many time steps as the estimated state diverges from the true state. When a measurement becomes available, the feedback gain K is calculated, the state is corrected, and the state covariance is updated (reduced) accordingly.

In Figure 2.5 a Kalman filter is applied to the previous dampened spring example. The 1-σ bound

on the position covariance is overlaid in green. As the filter converges the covariance tightens accordingly. The dramatic improvement in goodness of fit over the previously presented techniques (low-passing and integrating velocity) highlights the usefulness of the Kalman filter.

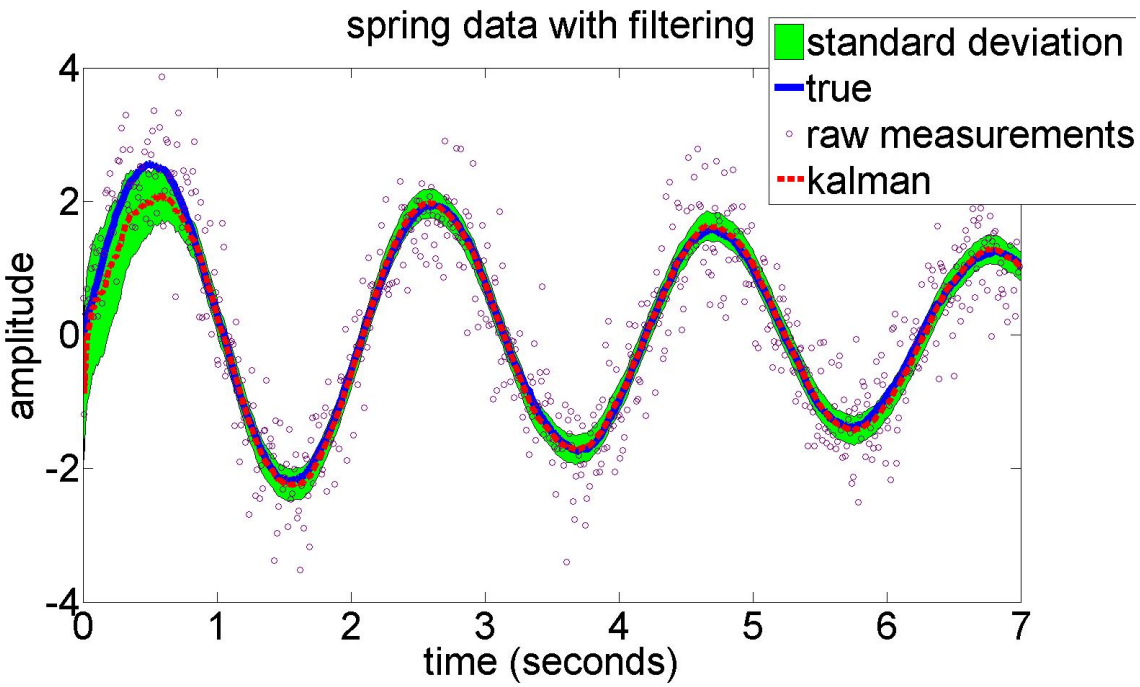


Figure 2.5: Kalman filter applied to spring with dampening and noisy measurements. The 1-σ bound on the position covariance shrinks as the filter converges.

2.3.1 The Extended Kalman Filter

The Kalman filter has been applied to nonlinear systems in many different ways. One of the most straightforward techniques is to linearize the system dynamics and the measurement function around the expected state $\vec{x}_k(-)$, and then apply the Kalman filter as normal. This is knows as extended Kalman filtering and the implementation equations, adapted from [2], are presented in Table 2.2.

With the extended Kalman filter we now have the necessary machinery to dynamically estimate attitude and drifting gyro biases, as long as they are modeled as states in our dynamical system. Next the three-dimensional attitude dynamic and measurement equations will be derived.

System dynamics	$\vec{x}_k = f_{k-1}(\vec{x}_{k-1}) + \vec{w}_{k-1}$ $\vec{w}_k \sim N(0, Q_k)$
Measurement	$\vec{z}_k = h_k(\vec{x}_k) + \vec{v}_k$ $\vec{v}_k \sim N(0, R_k)$
Covariance	$P_k = \mathcal{E}([\vec{x}_k - \hat{x}_k][\vec{x}_k - \hat{x}_k]^T)$
State propagation	$\hat{x}_k(-) = f_{k-1}(\hat{x}_{k-1}(+))$
Dynamics linearization	$\Phi_{k-1} = \left.\dfrac{\partial f_{k-1}}{\partial x}\right\|_{x=\hat{x}_{k-1}(+)}$
Predicted measurement	$\hat{z}_k = h_k(\hat{x}_k(-))$
Measurement linearization	$H_k = \left.\dfrac{\partial h_k}{\partial x}\right\|_{x=\hat{x}_k(-)}$
State covariance propagation	$P_k(-) = \Phi_{k-1} P_{k-1}(+) \Phi_{k-1}^T + Q_{k-1}$
Feedback gain	$K_k = P_k(-) H_k^T (H_k P_k(-) H_k^T + R_k)^{-1}$
State update	$\hat{x}_k(+) = \hat{x}_k(-) + K_k(z_k - \hat{z}_k)$
State covariance update	$P_k(+) = (I - K_k H_k) P_k(-)$

Table 2.2: Extended Kalman filter implementation equations

3 Attitude and Spatial Rotations

3.1 Introduction

A rigid body's attitude is its orientation relative to some reference frame such as level Earth. There are several equivalent mathematical representations of attitude and the relevant ones are discussed in this chapter.

In this thesis the "Earth frame" or "local Earth frame" refers to the north-east-down (NED) frame, where $\vec{x}$ points due north, $\vec{y}$ points due east, and $\vec{z}$ points to the center of the Earth (which is assumed here to be spherical and of radially symmetric density). The body frame refers to the frame of the aircraft where $\vec{x}$ points out the nose, $\vec{y}$ points out of the right wing, and $\vec{z}$ points down from the belly (see Figure 3.1).

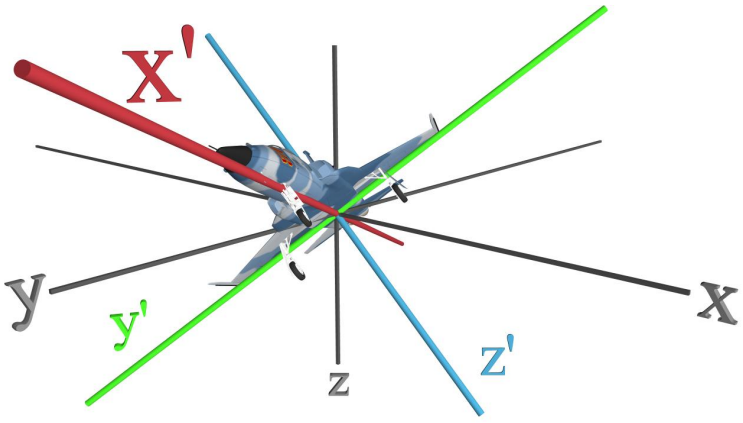


Figure 3.1: Earth and body reference frames. The Earth (NED) axes are (x, y, z) and the body axes are (x', y', z').

The NED frame's simplicity and the fact that gravity always points down makes this coordinate system useful, but caution must be exercised when integrating dynamics over long distances. A particle

could travel a unit distance north, west, south, and then east, and not arrive back where it started (see Figure 3.2). This is not a concern for us because this attitude estimator is being developed for a UAV with relatively short range.

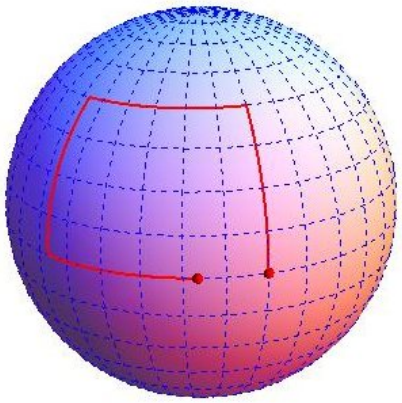


Figure 3.2: Nonlinearity in the NED frame. A particle travels a unit distance north, west, south, and then east, but does not arrive where it started.

3.2 Euler Angles

The aerospace Euler angles are probably the most easily accessible representation of attitude. They are known as yaw/heading, pitch, and roll, or ψ, θ, and ϕ (see Figure 3.3). The Euler angles are useful because the individual angles have intuitive significance to pilots. For example, navigation is done by tracking a heading ψ, and climbing/descending can be done with a combination of throttle and θ control.

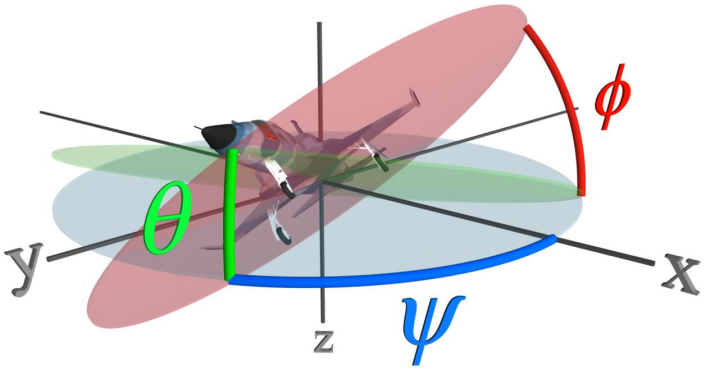


Figure 3.3: The Euler angles

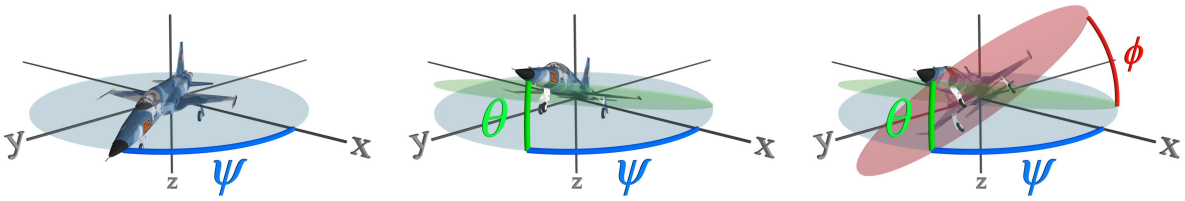
The three dimensional generalization of the rotation

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \tag{3.1}$$

is

$$\begin{pmatrix} u_1 \\ v_1 \\ w_1 \end{pmatrix} = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix}, \tag{3.2}$$

where (u_e, v_e, w_e) is a vector in the Earth frame and (u_1, v_1, w_1) is that vector in an intermediate frame (x_1, y_1, z_1). This represents a rotation of ψ about the original z-axis (see Figure 3.4(a)).



(a) ψ (b) θ (c) ϕ

Figure 3.4: The 3-2-1 Euler angle rotation sequence

The next rotation (θ) is taken about the intermediate y_1-axis, which is out the aircraft's right wing after the ψ rotation and before the ϕ rotation (see Figure 3.4(b)). The rotation matrix is

$$\begin{pmatrix} u_2 \\ v_2 \\ w_2 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ w_1 \end{pmatrix}, \tag{3.3}$$

where (u_2, v_2, w_2) is the vector in a second intermediate frame (x_2, y_2, z_2). The final rotation (ϕ) is taken about the intermediate x_2 axis, which points out the aircraft's nose (see Figure 3.4(c)). The rotation matrix is

$$\begin{pmatrix} u_b \\ v_b \\ w_b \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix} \begin{pmatrix} u_2 \\ v_2 \\ w_2 \end{pmatrix}, \tag{3.4}$$

where (u_b, v_b, w_b) is the vector in the final body frame (x_b, y_b, z_b). Combining equations (3.2), (3.3), and (3.4) yields

$$\begin{pmatrix} u_b \\ v_b \\ w_b \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix} \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix}, \quad (3.5)$$

which can be multiplied out yielding

$$\begin{pmatrix} u_b \\ v_b \\ w_b \end{pmatrix} = \begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \cos(\psi)\sin(\theta)\sin(\phi) - \cos(\phi)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\theta)\sin(\phi)\sin(\psi) & \cos(\theta)\sin(\phi) \\ \cos(\phi)\cos(\psi)\sin(\theta) + \sin(\phi)\sin(\psi) & -\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\sin(\psi) & \cos(\theta)\cos(\phi) \end{pmatrix} \begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix}. \quad (3.6)$$

The matrix in equation (3.6) rotates vectors from the NED frame to the body frame, and is expressed compactly as

$$\begin{pmatrix} u_b \\ v_b \\ w_b \end{pmatrix} = T(\phi, \theta, \psi) \begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix}. \quad (3.7)$$

There are singularities in equation (3.6) when the aircraft's nose is pointed straight up or down (i.e., in the negative or positive z direction). This is shown by letting θ approach $\frac{\pi}{2}$ in (3.6) and then applying trigonometric identities, yielding

$$\lim_{\theta \to \frac{\pi}{2}} T(\phi, \theta, \psi) = \begin{pmatrix} 0 & 0 & -1 \\ \sin(\phi - \psi) & cos(\phi - \psi) & 0 \\ \cos(\phi - \psi) & -sin(\phi - \psi) & 0 \end{pmatrix}. \quad (3.8)$$

Therefore, when the aircraft pitches straight up, the heading (ψ) and roll (ϕ) become indistinguishable. In the real world an airplane could never have a pitch of exactly $\theta = \frac{\pi}{2}$, but in a real flight control system the computer would be dividing very small numbers for θ close to $\frac{\pi}{2}$, and very small errors would rapidly grow into significant ones.

The advantage of using Euler angles for attitude representation is that they are intuitive and simple to visualize. Drawbacks include the singularities and the relative computational complexity of computing the

derivatives of equation (3.6) (which are transcendental) for implementation in the extended Kalman filter. The system dynamics (which are derived in Chapter 4) are also relatively computationally complex.

3.3 Direction Cosine Matrix

The Direction Cosine Matrix (DCM) is a 3×3 matrix which rotates a vector from one reference frame into another, and is fundamentally defined as

$$\vec{v}_b = R\vec{v}_e = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \vec{v}_e. \tag{3.9}$$

The components $r_{i,j}$ have already been derived as a function of Euler angles in equation (3.6).

For a matrix to be a rotation matrix it must rotate vectors while preserving their magnitude, which is equivalent to saying that for rotation matrix R

$$|\vec{v}_b|^2 = |\vec{v}_e|^2 \tag{3.10}$$

$$|R\vec{v}_e|^2 = |\vec{v}_e|^2$$

$$(R\vec{v}_e)^T(R\vec{v}_e) = \vec{v}_e^T\vec{v}_e$$

$$\vec{v}_e^T R^T R\vec{v}_e = \vec{v}_e^T\vec{v}_e$$

$$\vec{v}_e^T(R^T R)\vec{v}_e = \vec{v}_e^T\vec{v}_e$$

$$R^T R = I. \tag{3.11}$$

From equation (3.11) the inverse of rotation matrix R is its transpose so it is orthonormal.

While a DCM can itself be used to fully represent an aircraft's attitude, I personally have found it to be most useful as a tool within a different representation of attitude, i.e., it is simpler to use the vector $[\phi\ \theta\ \psi]^T$ to represent attitude, and then compute the DCM using equation (3.6) when it is necessary to rotate a vector from one frame to another.

3.4 Axis and Angle of Rotation

Euler's rotation theorem states that any sequence of rotations (such as the 3-2-1 Euler angle sequence in equation (3.5)) can be expressed as a single rotation of angle α about a certain axis $\vec{\sigma}$ (see Figure 3.5)[4].

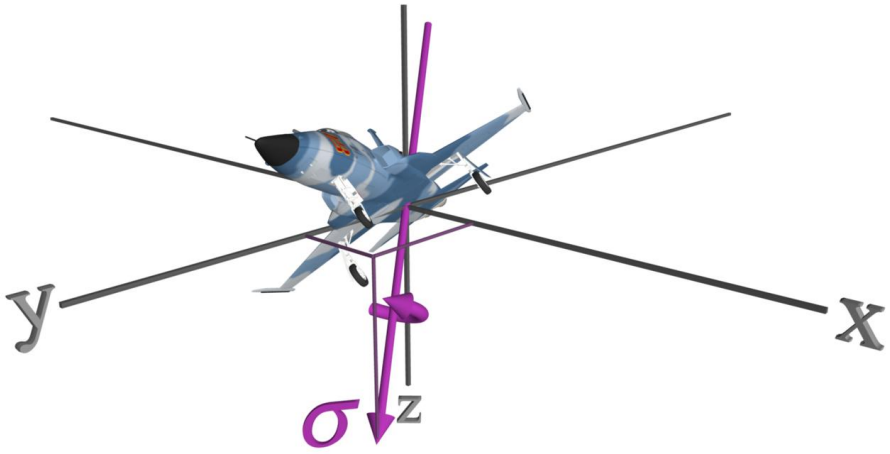


Figure 3.5: Generalized rotation axis. Any rotation sequence can be expressed as a single rotation of angle α about the Euler axis $\vec{\sigma}$.

To derive the general rotation axis, first consider the special case of the rotation about the z-axis

$$\begin{pmatrix} u_b \\ v_b \\ w_b \end{pmatrix} = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix} = \begin{pmatrix} u_e \cos(\psi) + v_e \sin(\psi) \\ -u_e \sin(\psi) + v_e \cos(\psi) \\ w_e \end{pmatrix}. \tag{3.12}$$

It is clear that the x and y components of vector $[u_e\ v_e\ w_e]^T$ are rotated while the z component is unaffected. Therefore the only vector that would be unaffected by a z-axis rotation is a vector with only a z component. It is a general result that a vector parallel to a given axis of rotation is invariant under that rotation. Another way of stating this is that the axis of rotation of a matrix T is the eigenvector of T associated with eigenvalue $\lambda = 1$.[1]

The full derivation of the angle of rotation about the generalized axis is borrowed from [5]. The

[1]The two requirements for a matrix to be a rotation matrix are that it orthogonal and that it has determinant equal to 1. From this it can be shown that one of its eigenvalues must be 1. For a complete discussion see Chapter 3 of [5].

general rotation matrix T is factored into

$$T = R^T A R = R^T \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} R. \tag{3.13}$$

Reading the rotations in equation (3.13) from right to left, matrix R first rotates the frame so that the new z-axis is parallel to the axis of rotation $\vec{\sigma}$. Matrix A then rotates the frame an angle of α about the new z-axis (which is $\vec{\sigma}$). Last, R^T undoes the original R rotation, restoring the original coordinate system plus the rotation of α about $\vec{\sigma}$.

Since this method always works, every rotation matrix can be factored into this form. Luckily it is unnecessary to ever perform this factorization. Taking the trace of matrix T, remembering that rotation matrices' inverses are their transposes, and using the identity $trace(XY) = trace(YX)$, yields

$$trace(T) = trace(R^T A R) \tag{3.14}$$

$$trace(T) = trace(R^T (AR))$$

$$trace(T) = trace((AR)R^T)$$

$$trace(T) = trace(A(RR^T))$$

$$trace(T) = trace(A(I))$$

$$trace(T) = trace(A)$$

$$trace(T) = trace(\begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix})$$

$$trace(T) = 1 + 2\cos(\alpha)$$

$$\alpha = \arccos\left(\frac{trace(T) - 1}{2}\right). \tag{3.15}$$

We now have a general formula for the angle of rotation about the general axis. In the next section we will see how this general axis and angle can be elegantly utilized for attitude representation and spacial rotation.

3.5 Quaternions

Quaternions are hyper-complex numbers of rank 4. A quaternion q can be expressed as

$$q = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \tag{3.16}$$

where

$$q_0, q_1, q_2, q_3 \in \mathbb{R},$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1,$$

and

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$$

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}.$$

A quaternion's conjugate is defined as

$$q^* \equiv q_0 - \mathbf{i}q_1 - \mathbf{j}q_2 - \mathbf{k}q_3. \tag{3.17}$$

Multiplication between two quaternions q and p is notated $q \otimes p$ and is performed as expected.

Keeping in mind that $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are non-commutative, distributing out terms and rearranging yields

$$
\begin{aligned}
q \otimes p &= (q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3)(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) \\
q \otimes p &= q_0(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) + \mathbf{i}q_1(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) \\
&\quad + \mathbf{j}q_2(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) + \mathbf{k}q_3(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) \\
q \otimes p &= q_0(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) + q_1(\mathbf{i}p_0 + \mathbf{i}\mathbf{i}p_1 + \mathbf{i}\mathbf{j}p_2 + \mathbf{i}\mathbf{k}p_3) \\
&\quad + q_2(\mathbf{j}p_0 + \mathbf{j}\mathbf{i}p_1 + \mathbf{j}\mathbf{j}p_2 + \mathbf{j}\mathbf{k}p_3) + q_3(\mathbf{k}p_0 + \mathbf{k}\mathbf{i}p_1 + \mathbf{k}\mathbf{j}p_2 + \mathbf{k}\mathbf{k}p_3) \\
q \otimes p &= q_0(p_0 + \mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) + q_1(\mathbf{i}p_0 - p_1 + \mathbf{k}p_2 - \mathbf{j}p_3) \\
&\quad + q_2(\mathbf{j}p_0 - \mathbf{k}p_1 - p_2 + \mathbf{i}p_3) + q_3(\mathbf{k}p_0 + \mathbf{j}p_1 - \mathbf{i}p_2 - p_3) \\
q \otimes p &= q_0p_0 - q_1p_1 - q_2p_2 - q_3p_3 + q_0(\mathbf{i}p_1 + \mathbf{j}p_2 + \mathbf{k}p_3) + p_0(\mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3) \\
&\quad + \mathbf{i}(q_2p_3 - q_3p_2) + \mathbf{j}(-q_1p_3 + q_3p_1) + \mathbf{k}(q_1p_2 - q_2p_1).
\end{aligned}
\tag{3.18}
$$

Treating a quaternion q as a having scalar and vector parts

$$
q = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix}, \tag{3.19}
$$

lets equation (3.18) be rewritten as

$$
q \otimes p = q_0p_0 - \vec{q} \cdot \vec{p} + q_0\vec{p} + p_0\vec{q} + \vec{q} \times \vec{p}, \tag{3.20}
$$

which is an efficient formula for quaternion multiplication. It is also clear from the cross-product term in equation (3.20) that quaternions do not commute under multiplication, i.e., $q \otimes p \neq p \otimes q$.

A quaternion's norm is notated $|q|$ and defined as

$$
|q| \equiv \sqrt{q^* \otimes q}\,, \tag{3.21}
$$

which can be simplified using equation (3.20):

$$\begin{aligned}
|q| &= \sqrt{q^* \otimes q} \\
&= \sqrt{\begin{pmatrix} q_0 \\ -\vec{q} \end{pmatrix} \otimes \begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix}} \\
&= \sqrt{q_0^2 - (-\vec{q}) \cdot \vec{q} + q_0\vec{q} + q_0(-\vec{q}) + (-\vec{q}) \times \vec{q}} \\
&= \sqrt{q_0^2 + \vec{q} \cdot \vec{q} + (q_0\vec{q} - q_0\vec{q}) - \vec{q} \times \vec{q}} \\
&= \sqrt{q_0^2 + \vec{q} \cdot \vec{q}} \\
|q| &= \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}\ . \qquad (3.22)
\end{aligned}$$

It is extremely useful to require that a quaternion be normalized,

$$\sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \equiv 1, \qquad (3.23)$$

and for the rest of the paper we will assume this is the case. (In fact, it is a convention that all rotation quaternions are normalized.) One immediate benefit of this is that a quaternion's conjugate is its inverse:

$$q^* \otimes q = q \otimes q^* = q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \qquad (3.24)$$

It is now stated (and will shortly be verified) that the quaternion rotation operator is

$$\begin{pmatrix} 0 \\ \vec{v}_b \end{pmatrix} = q^* \otimes \begin{pmatrix} 0 \\ \vec{v}_e \end{pmatrix} \otimes q, \qquad (3.25)$$

which through tedious algebra can be simplified to

$$\begin{pmatrix} 0 \\ \vec{v_b} \end{pmatrix} = \begin{pmatrix} 0 \\ (2q_0^2+2q_1^2-1)v_{e_1}+(2q_1q_2+2q_0q_3)v_{e_2}+(2q_1q_3-2q_0q_2)v_{e_3} \\ (2q_1q_2-2q_0q_3)v_{e_1}+(2q_0^2+2q_2^2-1)v_{e_2}+(2q_2q_3+2q_0q_1)v_{e_3} \\ (2q_1q_3+2q_0q_2)v_{e_1}+(2q_2q_3-2q_0q_1)v_{e_2}+(2q_0^2+2q_3^2-1)v_{e_3} \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ \vec{v_b} \end{pmatrix} = \begin{pmatrix} 0 \\ \begin{pmatrix} 2q_0^2+2q_1^2-1 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & 2q_0^2+2q_2^2-1 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 2q_0^2+2q_3^2-1 \end{pmatrix} \vec{v_e} \end{pmatrix},$$

which implies that

$$\vec{v_b} = \begin{pmatrix} 2q_0^2+2q_1^2-1 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & 2q_0^2+2q_2^2-1 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 2q_0^2+2q_3^2-1 \end{pmatrix} \vec{v_e}. \tag{3.26}$$

We now have the DCM as a function of q:

$$T(q) = \begin{pmatrix} 2q_0^2+2q_1^2-1 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & 2q_0^2+2q_2^2-1 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 2q_0^2+2q_3^2-1 \end{pmatrix}. \tag{3.27}$$

The eigenvalues of $T(q)$ are

$$\lambda = \begin{pmatrix} q_0^2+q_1^2+q_2^2+q_3^2 \\ q_0^2-q_1^2-q_2^2-q_3^2+2q_0\mathbf{i}\sqrt{q_1^2+q_2^2+q_3^2} \\ q_0^2-q_1^2-q_2^2-q_3^2-2q_0\mathbf{i}\sqrt{q_1^2+q_2^2+q_3^2} \end{pmatrix}, \tag{3.28}$$

so with the normalization condition in equation (3.23), the first eigenvalue of $T(q)$ is 1, which is a requirement for $T(q)$ to be the DCM. The other requirement is that $T(q)$ is orthogonal, which is easily verified by plugging in $T(q)^T T(q)$, yielding the identity matrix (the tedious algebra is omitted).

Now that it is confirmed that $T(q)$ is a rotation matrix, the general angle of rotation α is derived

by plugging equation (3.27) into (3.15) yielding

$$\alpha = \arccos\left(\frac{trace\,(T(q)) - 1}{2}\right)$$
$$\alpha = \arccos\left(\frac{(2q_0^2 + 2q_1^2 - 1) + (2q_0^2 + 2q_2^2 - 1) + (2q_0^2 + 2q_3^2 - 1) - 1}{2}\right)$$
$$\alpha = \arccos\left(2q_0^2 + (q_0^2 + q_1^2 + q_2^2 + q_3^2) - 2\right)$$
$$\alpha = \arccos(2q_0^2 - 1)$$
$$q_0^2 = \frac{\cos(\alpha) + 1}{2}$$
$$q_0 = \cos\left(\frac{\alpha}{2}\right) \tag{3.29}$$
$$\alpha = 2\arccos(q_0). \tag{3.30}$$

The general axis of rotation $\vec{\sigma}$ is derived by taking the eigenvector associated with eigenvalue $\lambda = 1$. The eigenvector turns out to be $[q_1\ q_2\ q_3]^T$ so $\vec{q}$ is now known up to a scaling constant, represented as

$$\vec{q} = c\frac{\vec{\sigma}}{|\sigma|}. \tag{3.31}$$

Combining equations (3.29) and (3.31) yields

$$q = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ c\frac{\sigma_x}{|\vec{\sigma}|} \\ c\frac{\sigma_y}{|\vec{\sigma}|} \\ c\frac{\sigma_z}{|\vec{\sigma}|} \end{pmatrix}, \tag{3.32}$$

and the normalization condition $|q| = 1$ is used on equation (3.32) to solve for the scaling constant c:

$$1 = |q|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2$$
$$1 = \cos\left(\frac{\alpha}{2}\right)^2 + \left(c\frac{\sigma_x}{|\vec{\sigma}|}\right)^2 + \left(c\frac{\sigma_y}{|\vec{\sigma}|}\right)^2 + \left(c\frac{\sigma_z}{|\vec{\sigma}|}\right)^2$$
$$1 = \cos\left(\frac{\alpha}{2}\right)^2 + c^2\left(\frac{\sigma_x^2 + \sigma_y^2 + \sigma_z^2}{|\vec{\sigma}|^2}\right) = \cos\left(\frac{\alpha}{2}\right)^2 + c^2\left(\frac{|\vec{\sigma}|^2}{|\vec{\sigma}|^2}\right)$$
$$1 = \cos\left(\frac{\alpha}{2}\right)^2 + c^2$$
$$c^2 = 1 - \cos\left(\frac{\alpha}{2}\right)^2$$
$$c = \sin\left(\frac{\alpha}{2}\right) \tag{3.33}$$

The attitude is therefore fully represented by the attitude quaternion

$$q = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right) \frac{\sigma_x}{|\vec{\sigma}|} \\ \sin\left(\frac{\alpha}{2}\right) \frac{\sigma_y}{|\vec{\sigma}|} \\ \sin\left(\frac{\alpha}{2}\right) \frac{\sigma_z}{|\vec{\sigma}|} \end{pmatrix}, \tag{3.34}$$

where $\vec{\sigma}$ and α are the general axis and angle of rotation.

If q and p are both quaternions, applying two subsequent rotations to a frame is written

$$p^* \otimes \left[q^* \otimes \begin{pmatrix} 0 \\ \vec{v_e} \end{pmatrix} \otimes q \right] \otimes p = (p^* \otimes q^*) \otimes \begin{pmatrix} 0 \\ \vec{v_e} \end{pmatrix} \otimes (q \otimes p) \tag{3.35}$$

$$= (q \otimes p)^* \otimes \begin{pmatrix} 0 \\ \vec{v_e} \end{pmatrix} \otimes (q \otimes p) \tag{3.36}$$

so it is clear that quaternion right-multiplication is equivalent to subsequent rotations. This can be written as

$$r = q \otimes p, \tag{3.37}$$

where rotating by q and then by p is equivalent to rotating by r. Equivalent forms of this are

$$q = r \otimes p^* \tag{3.38}$$

$$p = q^* \otimes r. \tag{3.39}$$

3.6 The Error Quaternion

The following rotation sequence consists of a large rotation represented by the quaternion q, followed by a small rotation represented by the quaternion p:

$$\vec{v}_b = \begin{pmatrix} (2p_0^2-1)+2p_1^2 & 2p_1p_2+2p_0p_3 & 2p_1p_3-2p_0p_2 \\ 2p_1p_2-2p_0p_3 & (2p_0^2-1)+2p_2^2 & 2p_2p_3+2p_0p_1 \\ 2p_1p_3+2p_0p_2 & 2p_2p_3-2p_0p_1 & (2p_0^2-1)+2p_3^2 \end{pmatrix} \cdot \begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \vec{v}_e. \quad (3.40)$$

Since the quaternion

$$p = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\frac{\sigma_x}{|\vec{\sigma}|} \\ \sin\left(\frac{\alpha}{2}\right)\frac{\sigma_y}{|\vec{\sigma}|} \\ \sin\left(\frac{\alpha}{2}\right)\frac{\sigma_z}{|\vec{\sigma}|} \end{pmatrix} \quad (3.41)$$

represents a rotation of small α, it can be linearized as

$$p = \begin{pmatrix} 1 \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \quad (3.42)$$

and the DCM becomes

$$T = \begin{pmatrix} 1+\epsilon_1^2 & 2\epsilon_1\epsilon_2+2\epsilon_3 & 2\epsilon_1\epsilon_3-2\epsilon_2 \\ 2\epsilon_1\epsilon_2-2\epsilon_3 & 1+2\epsilon_2^2 & 2\epsilon_2\epsilon_3+2\epsilon_1 \\ 2\epsilon_1\epsilon_3+2\epsilon_2 & 2\epsilon_2\epsilon_3-2\epsilon_1 & 1+2\epsilon_3^2 \end{pmatrix} \begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \tag{3.43}$$

$$\simeq \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} \begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \tag{3.44}$$

$$\simeq E \cdot T_0. \tag{3.45}$$

Another way of writing this is

$$q = \hat{q} \otimes \begin{pmatrix} 1 \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix}, \tag{3.46}$$

and we will see later how this can be extremely useful.

4 Attitude Dynamics

In order to understand the mechanics of spacial rotations it is necessary to derive the differential equations which relate attitude to a body frames's rate of rotation $\vec{\omega}$ (which in this thesis is always represented in the body frame). Using Euler angles for example, the equation required is

$$\frac{d}{dt}\begin{pmatrix}\phi\\ \theta\\ \psi\end{pmatrix} = F\begin{pmatrix}\omega_x\\ \omega_y\\ \omega_z\end{pmatrix}. \tag{4.1}$$

Before we are able to derive F we must think more about the meaning of a derivative in a non-inertial frame.

4.1 Equations of Motion in Rotating Reference Frames

There are often counter-intuitive results in dynamics in non-inertial reference frames. For example, a person riding in a circular track sees his or her velocity vector as constant and pointing straight forward (see Figure 4.1(a)). Since acceleration is usually the derivative of velocity, one might guess that for this constant body velocity one would experience no acceleration. However, the test pilots who ride the 20G centrifuge in Figure 4.1(b) would probably disagree! We must therefore exercise caution when taking derivatives in rotating frames.

Consider the special case of rotation parallel to the x-y plane where the frame is rotating with

(a) Free body diagram of particle in circular motion.

(b) The 20G centrifuge at NASA's Ames Research Center.

Figure 4.1: Dynamics in a rotating reference frame. Even though the velocity vector in the body frame is unchanging, mass in that frame experiences acceleration.

angular velocity $\vec{\omega} = [0\ 0\ \omega_z]^T$. The coordinate rotation equation is

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}. \tag{4.2}$$

Taking the derivative of both sides with respect to time yields

$$\frac{d}{dt}\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \frac{d}{dt}\left(\begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}\right) \tag{4.3}$$

$$\begin{pmatrix} \frac{du'}{dt} \\ \frac{dv'}{dt} \\ \frac{dw'}{dt} \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \frac{d}{dt}\begin{pmatrix} u \\ v \\ w \end{pmatrix} + \frac{d}{dt}\begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{4.4}$$

$$\begin{pmatrix} \frac{du'}{dt} \\ \frac{dv'}{dt} \\ \frac{dw'}{dt} \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{dw}{dt} \end{pmatrix} + \begin{pmatrix} -\omega_z \sin(\omega_z t) & \omega_z \cos(\omega_z t) & 0 \\ -\omega_z \cos(\omega_z t) & -\omega_z \sin(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{4.5}$$

$$\begin{pmatrix} \frac{du'}{dt} \\ \frac{dv'}{dt} \\ \frac{dw'}{dt} \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{dw}{dt} \end{pmatrix} + \begin{pmatrix} -\omega_z \sin(\omega_z t)u + \omega_z \cos(\omega_z t)v \\ -\omega_z \cos(\omega_z t)u - \omega_z \sin(\omega_z t)v \\ w \end{pmatrix} \tag{4.6}$$

$$\begin{pmatrix} \frac{du'}{dt} \\ \frac{dv'}{dt} \\ \frac{dw'}{dt} \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{dw}{dt} \end{pmatrix} - \begin{pmatrix} 0 & -\omega_z & 0 \\ \omega_z & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \cos(\omega_z t)u + \sin(\omega_z t)v \\ -\sin(\omega_z t)u + \cos(\omega_z t)v \\ w \end{pmatrix} \tag{4.7}$$

$$\begin{pmatrix} \frac{du'}{dt} \\ \frac{dv'}{dt} \\ \frac{dw'}{dt} \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{dw}{dt} \end{pmatrix} - \begin{pmatrix} 0 & -\omega_z & 0 \\ \omega_z & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} \tag{4.8}$$

$$\begin{pmatrix} \frac{du'}{dt} \\ \frac{dv'}{dt} \\ \frac{dw'}{dt} \end{pmatrix} = \begin{pmatrix} \cos(\omega_z t) & \sin(\omega_z t) & 0 \\ -\sin(\omega_z t) & \cos(\omega_z t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{dw}{dt} \end{pmatrix} - \vec{\omega} \times \begin{pmatrix} u' \\ v' \\ w' \end{pmatrix}. \tag{4.9}$$

In fact, it is a general result that

$$\frac{d\vec{v}_b}{dt} = T\frac{d\vec{v}_e}{dt} - \vec{\omega} \times \vec{v}_b \tag{4.10}$$

$$\frac{d\vec{v}_b}{dt} = T\frac{d\vec{v}_e}{dt} - \Omega\vec{v}_b, \tag{4.11}$$

where T is the DCM as usual, and Ω is the skew-symmetric matrix (or cross product operator)

$$\Omega = \vec{\omega}\times = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}. \tag{4.12}$$

Another important result is

$$\begin{aligned} \frac{d\vec{v}_b}{dt} = \frac{d}{dt}\left(T\vec{v}_e\right) = T\frac{d\vec{v}_e}{dt} + \frac{dT}{dt}\vec{v}_e &= T\frac{d\vec{v}_e}{dt} - \Omega\vec{v}_b \\ \Rightarrow \frac{dT}{dt}\vec{v}_e &= -\Omega\vec{v}_b \\ \frac{dT}{dt}\vec{v}_e &= -\Omega T\vec{v}_e \\ \frac{dT}{dt} &= -\Omega T. \end{aligned} \tag{4.13}$$

The real acceleration of the body (which would be used in Newton's second law) is $\frac{d\vec{v}_e}{dt}$, which in the body frame is experienced as $T\frac{d\vec{v}_e}{dt}$:

$$\vec{a}_b = T\frac{d\vec{v}_e}{dt} \tag{4.14}$$

$$\vec{a}_b = \frac{d\vec{v}_b}{dt} + \Omega\vec{v}_b. \tag{4.15}$$

In the previous example (Figure 4.1) the velocity in the body frame $\vec{v}_b$ is constant so it's derivative is 0. Since the rotation is about the negative z axis, $\vec{\omega} = [0\ 0\ -\omega]^T$. The measured acceleration in the body frame is therefore

$$\vec{a}_b = \Omega\vec{v}_b = \begin{pmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v \\ 0 \\ 0 \end{pmatrix} \tag{4.16}$$

$$\vec{a}_b = \begin{pmatrix} 0 \\ -\omega v \\ 0 \end{pmatrix}. \tag{4.17}$$

Substituting the angular velocity relationship $\omega = \frac{v}{r}$ yields

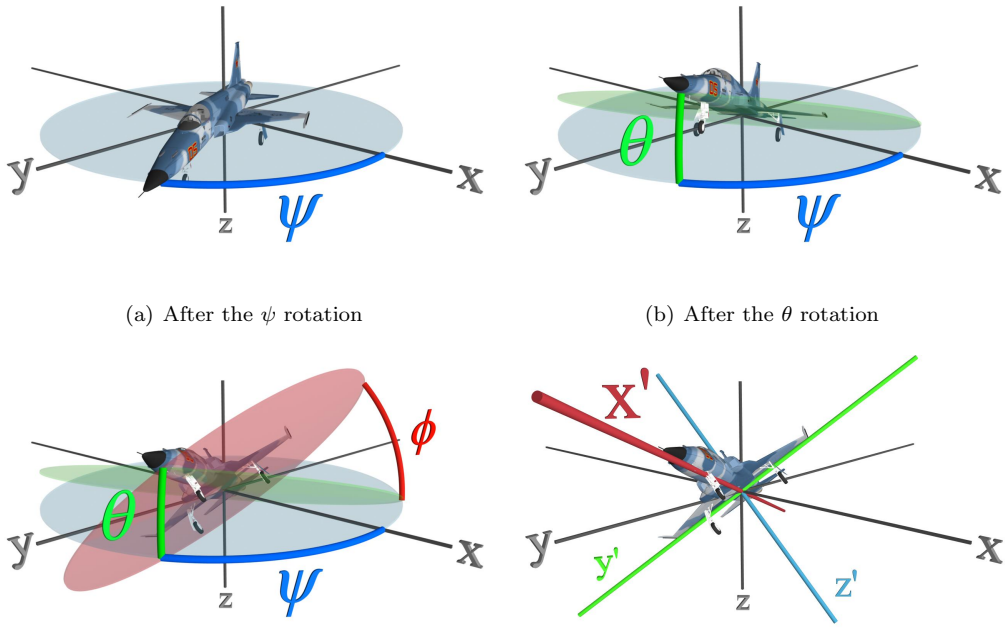
$$\vec{a}_b = \begin{pmatrix} 0 \\ -\frac{v^2}{r} \\ 0 \end{pmatrix}, \tag{4.18}$$

which is the familiar result in the correct direction.

4.2 Euler Angle Dynamics

It turns out that the Euler angle dynamics can be derived by inspection alone. Because the body rotation rates $\vec{\omega} = [\omega_x\ \omega_y\ \omega_z]^T$ are defined as instantaneous rotation rates about the x', y', and z' axes respectively (see Figure 4.2(d)), it can be observed from Figure 4.2(c) that ϕ is a rotation about the x' axis. θ is a rotation about an intermediate axis (see Figure 4.2(b)), so to project $\dot{\theta}$ onto the body frame it must be rotated using the ϕ rotation. Likewise, $\dot{\psi}$ must be rotated about first θ, and then ϕ. This is written as

$$\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + T_\phi \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + T_\phi T_\theta \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \tag{4.19}$$



(a) After the ψ rotation

(b) After the θ rotation

(c) After the ϕ rotation

(d) The body axes

Figure 4.2: The incremental axes in the 3-2-1 Euler rotation, and the final body axes

$$\begin{pmatrix}\omega_x\\ \omega_y\\ \omega_z\end{pmatrix} = \begin{pmatrix}\dot\phi\\ 0\\ 0\end{pmatrix} + \begin{pmatrix}1 & 0 & 0\\ 0 & \cos(\phi) & \sin(\phi)\\ 0 & -\sin(\phi) & \cos(\phi)\end{pmatrix}\begin{pmatrix}0\\ \dot\theta\\ 0\end{pmatrix} + \begin{pmatrix}1 & 0 & 0\\ 0 & \cos(\phi) & \sin(\phi)\\ 0 & -\sin(\phi) & \cos(\phi)\end{pmatrix}\begin{pmatrix}\cos(\theta) & 0 & -\sin(\theta)\\ 0 & 1 & 0\\ \sin(\theta) & 0 & \cos(\theta)\end{pmatrix}\begin{pmatrix}0\\ 0\\ \dot\psi\end{pmatrix}, \tag{4.20}$$

which can be simplified into

$$\begin{pmatrix}\omega_x\\ \omega_y\\ \omega_z\end{pmatrix} = \begin{pmatrix}1 & 0 & -\sin(\theta)\\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi)\\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi)\end{pmatrix}\begin{pmatrix}\dot\phi\\ \dot\theta\\ \dot\psi\end{pmatrix} \tag{4.21}$$

and inverted yielding

$$\begin{pmatrix}\dot\phi\\ \dot\theta\\ \dot\psi\end{pmatrix} = \begin{pmatrix}1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta)\\ 0 & \cos(\phi) & -\sin(\phi)\\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta)\end{pmatrix}\begin{pmatrix}\omega_x\\ \omega_y\\ \omega_z\end{pmatrix}. \tag{4.22}$$

Equation (4.22) is a system of three coupled, transcendental, nonlinear differential equations. They cannot be solved in closed form but must be integrated numerically, entailing the computationally intensive

task of computing a number of transcendental functions. There are also singularities at $\theta = \pm\frac{\pi}{2}$ which can cause cause a very rapid growth of errors. We will next use quaternions to solve these problems.

4.3 Quaternion Dynamics

As before, the DCM for the attitude quaternion is expressed as

$$\vec{v}_b = \begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \vec{v}_e. \tag{4.23}$$

Combining equations (4.12), (4.13), and (4.23) yields

$$\begin{aligned} \frac{d}{dt}&\begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix} \begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \end{aligned} \tag{4.24}$$

$$\begin{aligned} &\begin{pmatrix} 4q_0\dot{q}_0+4q_1\dot{q}_1 & 2\left(\dot{q}_0q_3+q_0\dot{q}_3+\dot{q}_1q_2+q_1\dot{q}_2\right) & 2\left(-\dot{q}_0q_2-q_0\dot{q}_2+\dot{q}_1q_3+q_1\dot{q}_3\right) \\ 2\left(-\dot{q}_0q_3-q_0\dot{q}_3+\dot{q}_1q_2+q_1\dot{q}_2\right) & 4q_0\dot{q}_0+4q_2\dot{q}_2 & 2\left(\dot{q}_0q_1+q_0\dot{q}_1+\dot{q}_2q_3+q_2\dot{q}_3\right) \\ 2\left(\dot{q}_0q_2+q_0\dot{q}_2+\dot{q}_1q_3+q_1\dot{q}_3\right) & 2\left(-\dot{q}_0q_1-q_0\dot{q}_1+\dot{q}_2q_3+q_2\dot{q}_3\right) & 4q_0\dot{q}_0+4q_3\dot{q}_3 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix} \begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix}, \end{aligned} \tag{4.25}$$

which can be rearranged algebraically into

$$\frac{d}{dt}\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{pmatrix}\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 0 & -\vec{\omega}^T \\ \vec{\omega} & -\vec{\omega}\times \end{pmatrix}\begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix}. \tag{4.26}$$

This dynamical equation is very elegant and is much simpler than the one for the Euler angles. There are no singularities and it is computationally simple. Furthermore, for constant body rates $\vec{\omega}$, equation (4.26) is linear and can be solved in closed form yielding

$$q(t_0+\Delta t) = \begin{pmatrix} \cos\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] \end{pmatrix} q(t_0), \tag{4.27}$$

where $|\omega| = \sqrt{\omega_1^2+\omega_2^2+\omega_3^2}$.

4.4 Error Quaternion Dynamics

As before, the error quaternion DCM is

$$T = \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix}\begin{pmatrix} (2q_0^2-1)+2q_1^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & (2q_0^2-1)+2q_2^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & (2q_0^2-1)+2q_3^2 \end{pmatrix} \tag{4.28}$$

$$= E\cdot T_0. \tag{4.29}$$

The time dependence is specified as

$$T(t) = E(t)T_0. \tag{4.30}$$

We let the large rotation T_0 be constant in time, and derive our dynamics using the linearized small rotation:

$$\Omega T(t) \quad = \quad \frac{d}{dt}\left(T(t)\right) \tag{4.31}$$

$$\Omega E(t)T_0 = \frac{d}{dt}\left(E(t)T_0\right) \tag{4.32}$$

$$\Omega E(t)T_0 = \frac{d}{dt}\left(E(t)\right)T_0 + E(t)\frac{d}{dt}\left(T_0\right) \tag{4.33}$$

$$\Omega E(t)T_0 = \frac{d}{dt}\left(E(t)\right)T_0 \tag{4.34}$$

$$\Omega E(t) = \frac{d}{dt}\left(E(t)\right) \tag{4.35}$$

$$\begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 2\dot{\epsilon_3} & -2\dot{\epsilon_2} \\ -2\dot{\epsilon_3} & 0 & 2\dot{\epsilon_1} \\ 2\dot{\epsilon_2} & -2\dot{\epsilon_1} & 0 \end{pmatrix} \tag{4.36}$$

which can easily be rearranged into

$$\frac{d}{dt}\begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \tag{4.37}$$

and solved, yielding

$$\vec{\epsilon}_k = \vec{\epsilon}_{k-1} + \frac{1}{2}\vec{\omega}_{k-1}\Delta t. \tag{4.38}$$

This is the simplest dynamical attitude equation we will come across. We will soon see how this can be utilized in an efficient extended Kalman filter.

5 Implementation Equations

The DCMs derived in Chapter 3 and the dynamics derived in Chapter 4 will now be used to derive f, Φ, h, H, Q, and R, the functions needed to implement the extended Kalman filter equations from Chapter 2 (see Table 2.2). Working MATLAB implementations are provided in Appendix B.

5.1 Full Quaternion Filter

The first attitude estimation system developed for this project was an extended Kalman filter using the full quaternion $[q_0 \ q_1 \ q_2 \ q_3]^T$. The computational complexity of this filter prevented it from being used in an embedded autopilot, which led to the use of the error quaternion filter. The original full quaternion implementation is included here for completeness.

The state is $\vec{x} = [q_0 \ \ q_1 \ \ q_2 \ \ q_3 \ \ \omega_1 \ \ \omega_2 \ \ \omega_3 \ \ b_1 \ \ b_2 \ \ b_3]^T$ where $\vec{b}$ is the gyro biases. The full implementation is summarized in Table 2.2 and it is necessary to derive the state transfer functions f and Φ, the measurement functions h and H, and the noise covariances Q and R.

5.1.1 State Transfer

Body rates $\vec{\omega}$ and gyro biases $\vec{b}$ are assumed to be random walk processes:

$$\vec{\omega}_k = \vec{\omega}_{k-1} + \vec{w}_{\omega,k-1} \tag{5.1}$$

$$\vec{b}_k = \vec{b}_{k-1} + \vec{w}_{b,k-1}. \tag{5.2}$$

Equations (5.1), (5.2), and (4.27) are combined to form f:

$$\begin{pmatrix} q_{0,k+1} \\ q_{1,k+1} \\ q_{2,k+1} \\ q_{3,k+1} \\ \omega_{1,k+1} \\ \omega_{2,k+1} \\ \omega_{3,k+1} \\ b_{1,k+1} \\ b_{2,k+1} \\ b_{3,k+1} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} \cos\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] \end{pmatrix} \begin{pmatrix} q_{0,k} \\ q_{1,k} \\ q_{2,k} \\ q_{3,k} \end{pmatrix} \\ \begin{matrix} \omega_{1,k} \\ \omega_{2,k} \\ \omega_{3,k} \\ b_{1,k} \\ b_{2,k} \\ b_{3,k} \end{matrix} \end{pmatrix}. \quad (5.3)$$

Partial derivatives of f are taken to form Φ

$$\Phi = \begin{pmatrix} \cos\left[\frac{\Delta t\omega}{2}\right] & -\frac{\omega_1 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & -\frac{\omega_2 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & -\frac{\omega_3 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \begin{matrix} -\frac{1}{2\omega^3}(\Delta t\omega\omega_1(q_1\omega_1 + q_2\omega_2 + q_3\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(q_0\Delta t\omega^2\omega_1 + 2\left(-q_2\omega_1\omega_2 - q_3\omega_1\omega_3 + q_1\left(\omega_2^2+\omega_3^2\right)\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right]) \end{matrix} \\ \frac{\omega_1 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \cos\left[\frac{\Delta t\omega}{2}\right] & \frac{\omega_3 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & -\frac{\omega_2 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_1(q_0\omega_1 - q_3\omega_2 + q_2\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(-q_1\Delta t\omega^2\omega_1 + 2\left(q_3\omega_1\omega_2 - q_2\omega_1\omega_3 + q_0\left(\omega_2^2+\omega_3^2\right)\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} \\ \frac{\omega_2 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & -\frac{\omega_3 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \cos\left[\frac{\Delta t\omega}{2}\right] & \frac{\omega_1 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_1(q_3\omega_1 + q_0\omega_2 - q_1\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(-q_2\Delta t\omega^2\omega_1 + 2\left(-q_0\omega_1\omega_2 + q_1\omega_1\omega_3 + q_3\left(\omega_2^2+\omega_3^2\right)\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} \\ \frac{\omega_3 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \frac{\omega_2 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & -\frac{\omega_1 \sin\left[\frac{\Delta t\omega}{2}\right]}{\omega} & \cos\left[\frac{\Delta t\omega}{2}\right] & \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_1(-q_2\omega_1 + q_1\omega_2 + q_0\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ -\left(q_3\Delta t\omega^2\omega_1 + 2\left(q_1\omega_1\omega_2 + q_0\omega_1\omega_3 + q_2\left(\omega_2^2+\omega_3^2\right)\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} \\ 0 & 0 & 0 & 0 & 1 \\ 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 \end{pmatrix}$$

$$\begin{pmatrix} \begin{matrix} -\frac{1}{2\omega^3}\Delta t\omega\omega_2(q_1\omega_1 + q_2\omega_2 + q_3\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(2q_2\left(\omega_1^2+\omega_3^2\right) + \omega_2\left(q_0\Delta t\omega^2 - 2(q_1\omega_1 + q_3\omega_3)\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & \begin{matrix} -\frac{1}{2\omega^3}\Delta t\omega\omega_3(q_1\omega_1 + q_2\omega_2 + q_3\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(2q_3\left(\omega_1^2+\omega_2^2\right) + q_0\Delta t\omega^2\omega_3 - 2(q_1\omega_1 + q_2\omega_2)\omega_3\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & 0 & 0 & 0 \\ \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_2(q_0\omega_1 - q_3\omega_2 + q_2\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ -\left(\omega_2\left(q_1\Delta t\omega^2 + 2q_0\omega_1 + 2q_2\omega_3\right) + 2q_3\left(\omega_1^2+\omega_3^2\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_3(q_0\omega_1 - q_3\omega_2 + q_2\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(2q_2\left(\omega_1^2+\omega_2^2\right) - \left(q_1\Delta t\omega^2 + 2q_0\omega_1 - 2q_3\omega_2\right)\omega_3\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & 0 & 0 & 0 \\ \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_2(q_3\omega_1 + q_0\omega_2 - q_1\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(-\omega_2\left(q_2\Delta t\omega^2 + 2q_3\omega_1 - 2q_1\omega_3\right) + 2q_0\left(\omega_1^2+\omega_3^2\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_3(q_3\omega_1 + q_0\omega_2 - q_1\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ -\left(2q_1\left(\omega_1^2+\omega_2^2\right) + \left(q_2\Delta t\omega^2 + 2q_3\omega_1 + 2q_0\omega_2\right)\omega_3\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & 0 & 0 & 0 \\ \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_2(-q_2\omega_1 + q_1\omega_2 + q_0\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(-\omega_2\left(q_3\Delta t\omega^2 - 2q_2\omega_1 + 2q_0\omega_3\right) + 2q_1\left(\omega_1^2+\omega_3^2\right)\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & \begin{matrix} \frac{1}{2\omega^3}\Delta t\omega\omega_3(-q_2\omega_1 + q_1\omega_2 + q_0\omega_3)\cos\left[\frac{\Delta t\omega}{2}\right] \\ +\left(2q_0\left(\omega_1^2+\omega_2^2\right) - \left(q_3\Delta t\omega^2 - 2q_2\omega_1 + 2q_1\omega_2\right)\omega_3\right)\sin\left[\frac{\Delta t\omega}{2}\right] \end{matrix} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (5.4)$$

5.1.2 Measurement

Accelerometers

The accelerometers measure gravity and the inertial acceleration in the body frame

$$\vec{z}_{accels} = T \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} - g \end{pmatrix}. \tag{5.5}$$

It was found to be most efficient to assume that tracking was accurate, i.e. $\hat{q}(-) \approx q$, and then subtract the GPS/barometer-calculated inertial acceleration from the accelerometer measurement, modifying equation (5.5) to

$$\vec{z}_{accels} = T(q) \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} - g \end{pmatrix} - T(\hat{q}) \begin{pmatrix} \ddot{x}_{gps} \\ \ddot{y}_{gps} \\ \ddot{z}_{baro} \end{pmatrix} \approx T(q) \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix}. \tag{5.6}$$

The barometer is used in the z-axis because it is much more accurate than GPS. The final accelerometer measurement is therefore

$$\vec{z}_{accels} = \begin{pmatrix} (2q_0^2 - 1) + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & (2q_0^2 - 1) + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & (2q_0^2 - 1) + 2q_3^2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix}$$

$$\vec{z}_{accels} = \begin{pmatrix} -2g(-q_0q_2 + q_1q_3) \\ -2g(q_0q_1 + q_2q_3) \\ -g\left(-1 + 2q_0^2 + 2q_3^2\right) \end{pmatrix}. \tag{5.7}$$

Magnetometers

The magnetometers measure the local magnetic field $\vec{B}$ in the body frame:

$$\vec{z}_{mags} = T \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} \tag{5.8}$$

$$\vec{z}_{mags} = \begin{pmatrix} (2q_0^2 - 1) + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & (2q_0^2 - 1) + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & (2q_0^2 - 1) + 2q_3^2 \end{pmatrix} \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix}$$

$$\vec{z}_{mags} = \begin{pmatrix} B_x\left(-1 + 2q_0^2 + 2q_1^2\right) + 2B_y(q_1q_2 + q_0q_3) + 2B_z(-q_0q_2 + q_1q_3) \\ B_y\left(-1 + 2q_0^2 + 2q_2^2\right) + 2B_x(q_1q_2 - q_0q_3) + 2B_z(q_0q_1 + q_2q_3) \\ B_z\left(-1 + 2q_0^2 + 2q_3^2\right) + 2B_x(q_0q_2 + q_1q_3) + 2B_y(-q_0q_1 + q_2q_3) \end{pmatrix} \tag{5.9}$$

Gyros

The rate gyros measure the body rotation $\vec{\omega}$ plus bias $\vec{b}$

$$\vec{z}_{\omega} = \begin{pmatrix} \omega_1 + b_1 \\ \omega_2 + b_2 \\ \omega_3 + b_3 \end{pmatrix}. \tag{5.10}$$

Full Measurement

Equations (5.7), (5.9), and (5.10) are combined to form the full measurement equation h

$$h(\vec{x}) = \begin{pmatrix} \vec{z}_{accels} \\ \vec{z}_{mags} \\ \vec{z}_{\omega} \end{pmatrix} \tag{5.11}$$

$$h(\vec{x}) = \begin{pmatrix} -2g(-q_0q_2 + q_1q_3) \\ -2g(q_0q_1 + q_2q_3) \\ -g\left(-1 + 2q_0^2 + 2q_3^2\right) \\ B_x\left(-1 + 2q_0^2 + 2q_1^2\right) + 2B_y(q_1q_2 + q_0q_3) + 2B_z(-q_0q_2 + q_1q_3) \\ B_y\left(-1 + 2q_0^2 + 2q_2^2\right) + 2B_x(q_1q_2 - q_0q_3) + 2B_z(q_0q_1 + q_2q_3) \\ B_z\left(-1 + 2q_0^2 + 2q_3^2\right) + 2B_x(q_0q_2 + q_1q_3) + 2B_y(-q_0q_1 + q_2q_3) \\ \omega_1 + b_1 \\ \omega_2 + b_2 \\ \omega_3 + b_3 \end{pmatrix}. \tag{5.12}$$

Taking partial derivatives forms the linearized measurement function H

$$H = \begin{pmatrix} 2gq_2 & -2gq_3 & 2gq_0 & -2gq_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2gq_1 & -2gq_0 & -2gq_3 & -2gq_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4gq_0 & 0 & 0 & -4gq_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4B_xq_0 - 2B_zq_2 + 2B_yq_3 & 4B_xq_1 + 2B_yq_2 + 2B_zq_3 & -2B_zq_0 + 2B_yq_1 & 2B_yq_0 + 2B_zq_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4B_yq_0 + 2B_zq_1 - 2B_xq_3 & 2B_zq_0 + 2B_xq_2 & 2B_xq_1 + 4B_yq_2 + 2B_zq_3 & -2B_xq_0 + 2B_zq_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4B_zq_0 - 2B_yq_1 + 2B_xq_2 & -2B_yq_0 + 2B_xq_3 & 2B_xq_0 + 2B_yq_3 & 2B_xq_1 + 2B_yq_2 + 4B_zq_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{5.13}$$

5.1.3 Noise Covariance Matrices

The process noise covariance is assumed to be

$$Q = \begin{pmatrix} \sigma_q^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_q^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_q^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_q^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_\omega^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_\omega^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_\omega^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_b^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_b^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_b^2 \end{pmatrix}, \tag{5.14}$$

where σ_q is the expected quaternion system noise, σ_ω is the random walk on the body rates, and σ_b is the random walk on the gyro biases. σ_b is identified off-line using static gyro data and σ_ω is tuned for expected real system dynamics. σ_q is time-dependent because a small rotation error in the airplane will couple into the quaternion coefficients according to equation (4.27). This is fixed by using the error quaternion implementation, but for now σ_q has been tuned with good results.

The sensor noise covariance is assumed to be

$$R = \begin{pmatrix} \sigma_{accel}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{accel}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{accel}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{mag}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{mag}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{mag}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{\omega}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{\omega}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{\omega}^2 \end{pmatrix}, \tag{5.15}$$

where σ_{accel}, σ_{mag}, and σ_{ω} are the standard deviations of the expected noise on the accelerometers, magnetometers, and rate gyros respectively. σ_{mag} and σ_{ω} can be identified off-line from static sensor data. σ_{accal} is a more difficult because the accelerometer measurement has had GPS/barometer-calculated inertial accelerations subtracted and that inertial acceleration has been rotated into the body frame using the *a priori* attitude quaternion. In future research this will be reflected by a modification of R, but for now σ_{accel} is tuned with good results.

This implementation is computationally simpler than an Euler angle implementation, but it is still too intensive to run in floating point on a 16-bit microcontroller at a reasonable update rate.

5.2 Error Quaternion

A major change from the full quaternion filter is that the gyro integration process is separate from the Kalman filter. The full quaternion $q = [q_0\ q_1\ q_2\ q_3]^T$ represents the attitude and it is propagated as the *a priori* attitude $q(-)$ using equation (4.27). When a GPS update becomes available the error quaternion is

initialized and defined as

$$q_{true} = q(-) \otimes \begin{pmatrix} 1 \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \tag{5.16}$$

from equation (3.46). Because the *a priori* attitude is taken as $q(-)$, the *a priori* error quaternion is zero. The Kalman filter is then used to estimate the error state $\vec{\epsilon}$ and gyro biases $\vec{b}$, and the attitude quaternion is updated by

$$\hat{q}(+) = \hat{q}(-) \otimes \begin{pmatrix} 1 \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix}. \tag{5.17}$$

At this point the attitude quaternion is re-normalized.

5.2.1 State Transfer

In the full quaternion implementation, the body rates were modeled as a random walk process with high noise. Since the gyro sensors have low noise, the Kalman filter weighs them with orders of magnitude more trust than the expected body rotation states, and the body rotation rates essentially become the sensor readings minus the bias.

To save processing power the error quaternion filter does this explicitly. The body rotation $\vec{\omega}$ is not a state and it is assumed that the rate gyros are perfectly noiseless, so the body rotation becomes the gyro sensor readings minus the gyro biases

$$\vec{\omega} = \vec{z}_\omega - \vec{b}. \tag{5.18}$$

The state for the error quaternion filter is therefore $\vec{x} = [\epsilon_1 \ \epsilon_2 \ \epsilon_3 \ b_1 \ b_2 \ b_3]^T$.

The biases are no longer modeled as random walks. Instead the first-order Gauss-Markov process is used:

$$\dot{b} = -\frac{b}{\tau} + w' \tag{5.19}$$

$$b_k = e^{-\frac{\Delta t}{\tau}} b_{k-1} + w. \tag{5.20}$$

This model is similar to a random walk on a short time scale $t << \tau$, but over a time scale $t \approx \tau$ the bias will be attracted to a zero state. This model is a better approximation of sensor drift in this case.

From equations (4.38) and (5.20), the Kalman filter state propagation function f is

$$\begin{pmatrix} \epsilon_{1,k} \\ \epsilon_{2,k} \\ \epsilon_{3,k} \\ b_{1,k} \\ b_{2,k} \\ b_{3,k} \end{pmatrix} = \begin{pmatrix} \epsilon_{1,k-1} + \frac{1}{2}\omega_{1,k-1}\Delta t \\ \epsilon_{2,k-1} + \frac{1}{2}\omega_{2,k-1}\Delta t \\ \epsilon_{3,k-1} + \frac{1}{2}\omega_{3,k-1}\Delta t \\ e^{-\frac{\Delta t}{\tau}} b_{1,k-1} \\ e^{-\frac{\Delta t}{\tau}} b_{2,k-1} \\ e^{-\frac{\Delta t}{\tau}} b_{3,k-1} \end{pmatrix},$$

which is modified by equation (5.18), becoming

$$\begin{pmatrix} \epsilon_{1,k} \\ \epsilon_{2,k} \\ \epsilon_{3,k} \\ b_{1,k} \\ b_{2,k} \\ b_{3,k} \end{pmatrix} = \begin{pmatrix} \epsilon_{1,k-1} + \frac{1}{2}(z_{1,k-1} - b_{1,k-1})\Delta t \\ \epsilon_{2,k-1} + \frac{1}{2}(z_{2,k-1} - b_{2,k-1})\Delta t \\ \epsilon_{3,k-1} + \frac{1}{2}(z_{3,k-1} - b_{3,k-1})\Delta t \\ e^{-\frac{\Delta t}{\tau}} b_{1,k-1} \\ e^{-\frac{\Delta t}{\tau}} b_{2,k-1} \\ e^{-\frac{\Delta t}{\tau}} b_{3,k-1} \end{pmatrix}. \tag{5.21}$$

A somewhat subtle point to be noted is that in this implementation, over a given time step the Gauss-Markov modification affects the biases but is not coupled into the corresponding body rotation rates as defined in equation (5.18). In this case the difference is negligible, and it may even be more correct this way depending on your philosophical interpretation of equation (5.18).

Partial derivatives of equation (5.21) are taken to form Φ

$$\Phi = \begin{pmatrix} 1 & 0 & 0 & -\frac{\Delta t}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & -\frac{\Delta t}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & -\frac{\Delta t}{2} \\ 0 & 0 & 0 & e^{-\frac{\Delta t}{\tau}} & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{-\frac{\Delta t}{\tau}} & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{-\frac{\Delta t}{\tau}} \end{pmatrix} \tag{5.22}$$

which is used to propagate the state covariance even though the error state $\vec{\epsilon}$ is not being used to integrate the gyros. This is appropriate so that the covariances on the error quaternion $\vec{\epsilon}$ will be correct when an update occurs. This form of Φ is much simpler than the previous one, and its sparseness is one of the key factors in the error quaternion filter's relative computational simplicity.

5.2.2 Measurement

Accelerometers

As before, the GPS/barometer-derived inertial acceleration is subtracted from the accelerometer measurements

$$\vec{z}_{accels} = T \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} - g \end{pmatrix} - T(\hat{q}) \begin{pmatrix} \ddot{x}_{gps} \\ \ddot{y}_{gps} \\ \ddot{z}_{baro} \end{pmatrix} \approx T \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix}. \tag{5.23}$$

From equation (3.43) the accelerometer measurement becomes

$$\vec{z}_{accels} = T \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix}$$

$$\vec{z}_{accels} = \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} \begin{pmatrix} (2q_0^2 - 1) + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & (2q_0^2 - 1) + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & (2q_0^2 - 1) + 2q_3^2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} \tag{5.24}$$

$$\vec{z}_{accels} = \begin{pmatrix} -g\left([2q_1q_3 - 2q_0q_2] - 2\epsilon_2\left[(2q_0^2-1) + 2q_3^2\right] + 2\epsilon_3\left[2q_2q_3 + 2q_0q_1\right]\right) \\ -g\left([2q_2q_3 + 2q_0q_1] + 2\epsilon_1\left[(2q_0^2-1) + 2q_3^2\right] - 2\epsilon_3\left[2q_1q_3 - 2q_0q_2\right]\right) \\ -g\left(\left[(2q_0^2-1) + 2q_3^2\right] - 2\epsilon_1\left[2q_2q_3 + 2q_0q_1\right] + 2\epsilon_2\left[2q_1q_3 - 2q_0q_2\right]\right) \end{pmatrix} \tag{5.25}$$

Magnetometers

Similarly, the magnetometer measurements are

$$\vec{z}_{mags} = T \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix}$$

$$\vec{z}_{mags} = \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} \begin{pmatrix} (2q_0^2-1) + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & (2q_0^2-1) + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & (2q_0^2-1) + 2q_3^2 \end{pmatrix} \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix}$$

$$\vec{z}_{mags} = \begin{pmatrix} B_y\left(2(q_1q_2 + q_0q_3) - 4(-q_0q_1 + q_2q_3)\epsilon_2 + 2\left(-1 + 2q_0^2 + 2q_2^2\right)\epsilon_3\right) + B_x\left(-1 + 2q_0^2 + 2q_1^2 - 4(q_0q_2 + q_1q_3)\epsilon_2 + 4(q_1q_2 - q_0q_3)\epsilon_3\right) \\ +B_z\left(2(-q_0q_2 + q_1q_3) - 2\left(-1 + 2q_0^2 + 2q_3^2\right)\epsilon_2 + 4(q_0q_1 + q_2q_3)\epsilon_3\right) \\ B_x\left(2(q_1q_2 - q_0q_3) + 4(q_0q_2 + q_1q_3)\epsilon_1 - 2\left(-1 + 2q_0^2 + 2q_1^2\right)\epsilon_3\right) + B_y\left(-1 + 2q_0^2 + 2q_2^2 + 4(-q_0q_1 + q_2q_3)\epsilon_1 - 4(q_1q_2 + q_0q_3)\epsilon_3\right) \\ +B_z\left(2(q_0q_1 + q_2q_3) + 2\left(-1 + 2q_0^2 + 2q_3^2\right)\epsilon_1 - 4(-q_0q_2 + q_1q_3)\epsilon_3\right) \\ B_x\left(2(q_0q_2 + q_1q_3) - 4(q_1q_2 - q_0q_3)\epsilon_1 + 2\left(-1 + 2q_0^2 + 2q_1^2\right)\epsilon_2\right) + B_y\left(2(-q_0q_1 + q_2q_3) - 2\left(-1 + 2q_0^2 + 2q_2^2\right)\epsilon_1 + 4(q_1q_2 + q_0q_3)\epsilon_2\right) \\ +B_z\left(-1 + 2q_0^2 + 2q_3^2 - 4(q_0q_1 + q_2q_3)\epsilon_1 + 4(-q_0q_2 + q_1q_3)\epsilon_2\right) \end{pmatrix}. \tag{5.26}$$

Full Measurement

Equations (5.25) and (5.26) are combined to form the full measurement function h

$$h = \begin{pmatrix} \vec{z}_{accels} \\ \vec{z}_{mags} \end{pmatrix}$$

$$h = \begin{pmatrix} -g\left([2q_1q_3 - 2q_0q_2] - 2\epsilon_2\left[(2q_0^2-1) + 2q_3^2\right] + 2\epsilon_3\left[2q_2q_3 + 2q_0q_1\right]\right) \\ -g\left([2q_2q_3 + 2q_0q_1] + 2\epsilon_1\left[(2q_0^2-1) + 2q_3^2\right] - 2\epsilon_3\left[2q_1q_3 - 2q_0q_2\right]\right) \\ -g\left(\left[(2q_0^2-1) + 2q_3^2\right] - 2\epsilon_1\left[2q_2q_3 + 2q_0q_1\right] + 2\epsilon_2\left[2q_1q_3 - 2q_0q_2\right]\right) \\ B_y\left(2(q_1q_2 + q_0q_3) - 4(-q_0q_1 + q_2q_3)\epsilon_2 + 2\left(-1 + 2q_0^2 + 2q_2^2\right)\epsilon_3\right) + B_x\left(-1 + 2q_0^2 + 2q_1^2 - 4(q_0q_2 + q_1q_3)\epsilon_2 + 4(q_1q_2 - q_0q_3)\epsilon_3\right) \\ +B_z\left(2(-q_0q_2 + q_1q_3) - 2\left(-1 + 2q_0^2 + 2q_3^2\right)\epsilon_2 + 4(q_0q_1 + q_2q_3)\epsilon_3\right) \\ B_x\left(2(q_1q_2 - q_0q_3) + 4(q_0q_2 + q_1q_3)\epsilon_1 - 2\left(-1 + 2q_0^2 + 2q_1^2\right)\epsilon_3\right) + B_y\left(-1 + 2q_0^2 + 2q_2^2 + 4(-q_0q_1 + q_2q_3)\epsilon_1 - 4(q_1q_2 + q_0q_3)\epsilon_3\right) \\ +B_z\left(2(q_0q_1 + q_2q_3) + 2\left(-1 + 2q_0^2 + 2q_3^2\right)\epsilon_1 - 4(-q_0q_2 + q_1q_3)\epsilon_3\right) \\ B_x\left(2(q_0q_2 + q_1q_3) - 4(q_1q_2 - q_0q_3)\epsilon_1 + 2\left(-1 + 2q_0^2 + 2q_1^2\right)\epsilon_2\right) + B_y\left(2(-q_0q_1 + q_2q_3) - 2\left(-1 + 2q_0^2 + 2q_2^2\right)\epsilon_1 + 4(q_1q_2 + q_0q_3)\epsilon_2\right) \\ +B_z\left(-1 + 2q_0^2 + 2q_3^2 - 4(q_0q_1 + q_2q_3)\epsilon_1 + 4(-q_0q_2 + q_1q_3)\epsilon_2\right) \end{pmatrix} \tag{5.27}$$

Taking partial derivatives of equation (5.27) yields H

$$H = \begin{pmatrix} 0 & 2g(-1+2q_0^2+2q_3^2) & -4g(q_0q_1+q_2q_3) & 0 & 0 & 0 \\ -2g(-1+2q_0^2+2q_3^2) & 0 & 4g(-q_0q_2+q_1q_3) & 0 & 0 & 0 \\ 4g(q_0q_1+q_2q_3) & -4g(-q_0q_2+q_1q_3) & 0 & 0 & 0 & 0 \\ 0 & \begin{matrix}-4B_x(q_0q_2+q_1q_3)-4B_y(-q_0q_1+q_2q_3)\\ -2B_z(-1+2q_0^2+2q_3^2)\end{matrix} & \begin{matrix}2B_y(-1+2q_0^2+2q_2^2)+4B_x(q_1q_2-q_0q_3)\\ +4B_z(q_0q_1+q_2q_3)\end{matrix} & 0 & 0 & 0 \\ \begin{matrix}4B_x(q_0q_2+q_1q_3)+4B_y(-q_0q_1+q_2q_3)\\ +2B_z(-1+2q_0^2+2q_3^2)\end{matrix} & 0 & \begin{matrix}-2B_x(-1+2q_0^2+2q_1^2)-4B_y(q_1q_2+q_0q_3)\\ -4B_z(-q_0q_2+q_1q_3)\end{matrix} & 0 & 0 & 0 \\ \begin{matrix}-2B_y(-1+2q_0^2+2q_2^2)-4B_x(q_1q_2-q_0q_3)\\ -4B_z(q_0q_1+q_2q_3)\end{matrix} & \begin{matrix}2B_x(-1+2q_0^2+2q_1^2)+4B_y(q_1q_2+q_0q_3)\\ +4B_z(-q_0q_2+q_1q_3)\end{matrix} & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{5.28}$$

Because the *a priori* error state is zero, the expected measurement is the same as in the full quaternion filter

$$\hat{z} = \begin{pmatrix} -2g(-q_0q_2+q_1q_3) \\ -2g(q_0q_1+q_2q_3) \\ -g\left(-1+2q_0^2+2q_3^2\right) \\ B_x\left(-1+2q_0^2+2q_1^2\right)+2B_y(q_1q_2+q_0q_3)+2B_z(-q_0q_2+q_1q_3) \\ B_y\left(-1+2q_0^2+2q_2^2\right)+2B_x(q_1q_2-q_0q_3)+2B_z(q_0q_1+q_2q_3) \\ B_z\left(-1+2q_0^2+2q_3^2\right)+2B_x(q_0q_2+q_1q_3)+2B_y(-q_0q_1+q_2q_3) \end{pmatrix}. \tag{5.29}$$

5.2.3 Noise Covariance Matrices

The process noise covariance is

$$Q = \begin{pmatrix} \sigma_\epsilon^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_\epsilon^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_\epsilon^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_b^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_b^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_b^2 \end{pmatrix}, \tag{5.30}$$

where σ_b is the expected random walk term in the gyro biases' Gauss-Markov drift, and σ_ϵ is the expected error state between the *a priori* and *a posteriori* attitudes. σ_b (and τ) are identified off-line using the real rate gyros. σ_ϵ includes a number of factors such as gyro gain error, gyro misalignment, unmodeled gyro noise, etc., and is therefore tuned for optimal results.

The sensor noise covariance is

$$R = \begin{pmatrix} \sigma_{accel}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{accel}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{accel}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{mag}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{mag}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{mag}^2 \end{pmatrix}, \tag{5.31}$$

where σ_{accel} and σ_{mag} are the standard deviations of the expected noise on the accelerometers and magnetometers respectively. σ_{mag} is identified off-line from static magnetometer data. As before, deriving σ_{accal} is difficult because of the subtraction of GPS/barometer-calculated inertial acceleration. σ_{accel} is again tuned with good results.

5.2.4 Implementation Summary

The complete error quaternion extended Kalman filter implementation is written explicitly in Table 5.1. A working MATLAB implementation is provided in Appendix B.

5.3 100 Hz Measurement Quaternion Filter

The error quaternion filter is much more simple computationally than the full quaternion filter, but it can be made to run significantly faster with a slight modification. The most computationally intensive step in the extended Kalman filter equations is an $n \times n$ matrix inverse where n is the length of the measurement $\vec{z}$. In the error quaternion filter this inverse is 6×6 which is very demanding for a small microcontroller. The measurement quaternion filter is exactly the same as the error quaternion filter except that the GPS/barometer, accelerometers, and magnetometers are used to form a direct measurement of the error state $\vec{\epsilon}$, and the measurement equation becomes

$$\vec{z}_k = \vec{\epsilon}_k + \vec{v}_k \tag{5.32}$$

$$\vec{v}_k \sim \mathcal{N}(0, R_k). \tag{5.33}$$

Φ	equation (5.22)
Q	equation (5.30)
R	equation (5.31)

Each time step

State propagation

$$\vec{\omega} = \vec{z}_{\omega_{k-1}} - \vec{b}_{k-1}$$

$$q_k(-) = \begin{pmatrix} \cos\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] \end{pmatrix} q_{k-1}$$

$$\vec{b}_k = e^{-\frac{\Delta t}{\tau}}\vec{b}_{k-1}$$

State covariance propagation:

$$P_k(-) = \Phi_{k-1}P_{k-1}\Phi_{k-1}^T + Q_{k-1}$$

When new GPS measurement arrives

Predicted measurement: $\hat{z}_k$ from equation (5.29)

Measurement linearization: H_k from equation (5.28)

Feedback gain:

$$K_k = P_k(-)H_k^T(H_kP_k(-)H_k^T + R_k)^{-1}$$

GPS acceleration correction:

$$\vec{z} = \left[\vec{z}_{acc} - T(q(-))\ddot{\vec{x}}_{gps/baro},\ \vec{z}_{mags}\right]^T$$

State update:

$$\begin{pmatrix} \epsilon_{1,k}(+) \\ \epsilon_{2,k}(+) \\ \epsilon_{3,k}(+) \\ b_{1,k}(+) \\ b_{2,k}(+) \\ b_{3,k}(+) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ b_{1,k}(-) \\ b_{2,k}(-) \\ b_{3,k}(-) \end{pmatrix} + K_k(z_k - \hat{z}_k)$$

Attitude quaternion update:

$$\hat{q}(+) = \hat{q}(-) \otimes \begin{pmatrix} 1 \\ \epsilon_1(+) \\ \epsilon_2(+) \\ \epsilon_3(+) \end{pmatrix}$$

Quaternion re-normalization:

$$\hat{q}(+) = \frac{\hat{q}(+)}{\sqrt{\hat{q}(+)^*\hat{q}(+)}}$$

State covariance update:

$$P_k(+) = (I - K_kH_k)P_k(-)$$

Table 5.1: Error quaternion filter implementation algorithm

Because the individual accelerometer and magnetometer sensor errors are propagated in a convoluted way through the measurement of $\vec{\epsilon}$, proper formation of R_k is the key to getting good performance with this filter.

5.3.1 Measuring $\vec{\epsilon}$ Directly

If two non-parallel vectors are known in one frame, then from the noiseless measurement of those vectors in another frame one can reconstruct the attitude of the rotated frame. This is described thoroughly in [6], which also presents an efficient method for calculating the rotated frame's attitude, represented as the measurement quaternion. A summary of that method is presented, along with my modification for noisy measurements.

The DCM is formulated using the error quaternion

$$\begin{pmatrix} v_{b1} \\ v_{b2} \\ v_{b3} \end{pmatrix} = \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} T(q) \begin{pmatrix} v_{e1} \\ v_{e2} \\ v_{e3} \end{pmatrix}. \tag{5.34}$$

The expectation value of the vector in the body frame is

$$\begin{pmatrix} \hat{v}_{b1} \\ \hat{v}_{b2} \\ \hat{v}_{b3} \end{pmatrix} = T(q) \begin{pmatrix} v_{e1} \\ v_{e2} \\ v_{e3} \end{pmatrix}. \tag{5.35}$$

The difference between the observed and expectation value of $\vec{v}_b$ is

$$\begin{aligned} \begin{pmatrix} \delta v_{b1} \\ \delta v_{b2} \\ \delta v_{b3} \end{pmatrix} &\equiv \begin{pmatrix} v_{b1} \\ v_{b2} \\ v_{b3} \end{pmatrix} - \begin{pmatrix} \hat{v}_{b1} \\ \hat{v}_{b2} \\ \hat{v}_{b3} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} T(q) \begin{pmatrix} v_{e1} \\ v_{e2} \\ v_{e3} \end{pmatrix} - T(q) \begin{pmatrix} v_{e1} \\ v_{e2} \\ v_{e3} \end{pmatrix} \end{aligned} \tag{5.36}$$

$$= \left[\begin{pmatrix} 1 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 1 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right] T(q) \begin{pmatrix} v_{e1} \\ v_{e2} \\ v_{e3} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 0 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 0 \end{pmatrix} T(q) \begin{pmatrix} v_{e1} \\ v_{e2} \\ v_{e3} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 2\epsilon_3 & -2\epsilon_2 \\ -2\epsilon_3 & 0 & 2\epsilon_1 \\ 2\epsilon_2 & -2\epsilon_1 & 0 \end{pmatrix} \begin{pmatrix} \hat{v}_{b1} \\ \hat{v}_{b2} \\ \hat{v}_{b3} \end{pmatrix}$$

$$\begin{pmatrix} \delta v_{b1} \\ \delta v_{b2} \\ \delta v_{b3} \end{pmatrix} = \begin{pmatrix} 0 & -2\hat{v}_{b3} & 2\hat{v}_{b2} \\ 2\hat{v}_{b3} & 0 & -2\hat{v}_{b1} \\ -2\hat{v}_{b2} & 2\hat{v}_{b1} & 0 \end{pmatrix} \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \tag{5.37}$$

The expected accelerometer measurement $\hat{z}_a$ is

$$\hat{z}_a = T(q) \begin{pmatrix} \ddot{x}_{gps} \\ \ddot{y}_{gps} \\ \ddot{z}_{baro} - g \end{pmatrix} \tag{5.38}$$

and the expected magnetometer measurement $\hat{z}_b$ is

$$\hat{z}_b = T(q) \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix}. \tag{5.39}$$

These are combined with equation (5.37) yielding

$$\begin{pmatrix} \vec{z}_a - \hat{\vec{z}}_a \\ \vec{z}_b - \hat{\vec{z}}_b \end{pmatrix} = \begin{pmatrix} 0 & -2\hat{z}_{a3} & 2\hat{z}_{a2} \\ 2\hat{z}_{a3} & 0 & -2\hat{v}_{a1} \\ -2\hat{z}_{a2} & 2\hat{z}_{a1} & 0 \\ 0 & -2\hat{v}_{z3} & 2\hat{v}_{z2} \\ 2\hat{v}_{z3} & 0 & -2\hat{v}_{z1} \\ -2\hat{v}_{z2} & 2\hat{v}_{z1} & 0 \end{pmatrix} \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \tag{5.40}$$

which can be solved with a standard weighted least squares formula *using only a 3 × 3 inverse.* The formula is taken from [3] and written as

$$A\vec{\epsilon} = \vec{y} + \vec{v} \tag{5.41}$$

$$\vec{v} \sim \mathcal{N}(0, R_0) \tag{5.42}$$

$$\vec{\epsilon}_{ls} = (A^T R^{-1} A)^{-1} A^T R^{-1} \vec{y} \tag{5.43}$$

$$\vec{\epsilon}_{ls} = M\vec{y} \tag{5.44}$$

where $\vec{\epsilon}_{ls}$ is the weighted least squares estimate of $\vec{\epsilon}$, and

$$\vec{\epsilon}_{ls} = \vec{\epsilon} + \vec{u} \tag{5.45}$$

$$\vec{u} \sim \mathcal{N}(0, R) \tag{5.46}$$

$$R = (A^T R^{-1} A)^{-1} = M R_0 M^T. \tag{5.47}$$

R_0 is taken to be the sensor noise from equation (5.31). It is diagonal and its inverse is easily pre-computed for computational speed. This method for directly measuring the error quaternion is very accurate if $T(q)$ is calculated using the *a priori* attitude quaternion $q(-)$.

5.3.2 Implementation Summary

Because the error quaternion is measured directly, the measurement function becomes

$$h(\vec{x}) = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \tag{5.48}$$

and the linearized measurement is therefore

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{5.49}$$

The sparseness of H is another factor which greatly reduces the computational complexity of this filter. Because the error state linearization is about the *a priori* attitude $q(-)$, both the expected error quaternion $\hat{\epsilon}$ and its expected measurement $\hat{z}$ are zero.

The implementation is therefore identical to the error quaternion filter with the modification of $\vec{z}$, $\hat{z}$, H, and R. A summary of the implementation equations is provided in Table 5.2 and a working MATLAB implementation is provided in Appendix B.

Φ	equation (5.22)
Q	equation (5.30)
R	equation (5.47)

Each time step

State propagation

$$\vec{\omega} = \vec{z}_{\omega_{k-1}} - \vec{b}_{k-1}$$

$$q_k(-) = \begin{pmatrix} \cos\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] \\ \frac{\omega_3}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \frac{\omega_2}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & -\frac{\omega_1}{|\omega|}\sin\left[\frac{|\omega|\Delta t}{2}\right] & \cos\left[\frac{|\omega|\Delta t}{2}\right] \end{pmatrix} q_{k-1}$$

$$\vec{b}_k = e^{-\frac{\Delta t}{\tau}}\vec{b}_{k-1}$$

State covariance propagation:

$$P_k(-) = \Phi_{k-1}P_{k-1}\Phi_{k-1}^T + Q_{k-1}$$

When new GPS measurement arrives

Measurement linearization:

$$H_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Feedback gain:

$$K_k = P_k(-)H_k^T(H_kP_k(-)H_k^T + R_k)^{-1}$$

Measurement:

$\vec{z} = \vec{\epsilon}_{ls}$ from equation (5.43)

State update:

$$\begin{pmatrix} \epsilon_{1,k}(+) \\ \epsilon_{2,k}(+) \\ \epsilon_{3,k}(+) \\ b_{1,k}(+) \\ b_{2,k}(+) \\ b_{3,k}(+) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ b_{1,k}(-) \\ b_{2,k}(-) \\ b_{3,k}(-) \end{pmatrix} + K_k\vec{z}_k$$

Attitude quaternion update:

$$\hat{q}(+) = \hat{q}(-) \otimes \begin{pmatrix} 1 \\ \epsilon_1(+) \\ \epsilon_2(+) \\ \epsilon_3(+) \end{pmatrix}$$

Quaternion re-normalization:

$$\hat{q}(+) = \frac{\hat{q}(+)}{\sqrt{\hat{q}(+)^*\hat{q}(+)}}$$

State covariance update:

$$P_k(+) = (I - K_kH_k)P_k(-)$$

Table 5.2: Measurement quaternion filter implementation algorithm

6 Results

6.1 Simulation

A MATLAB simulation was developed for debugging and testing the attitude estimation algorithms (see Figure 6.1). A Simulink remote control aircraft model developed in [7] was run in real time by a human pilot using a joystick. The model's state was used to generate sensor data, which were corrupted with noise. The gyro biases were corrupted with realistic first order Gauss-Markov drift which was identified off-line from an InvenSense IDG300 rate gyro. GPS measurements were simulated at a constant 2 Hz and were not corrupted with noise (which has yet to be characterized). With the addition of noise, σ_accel will have to be increased. The extended Kalman filter was run in real time as part of the simulation (see code in Appendix B), and all states and state estimates were logged.

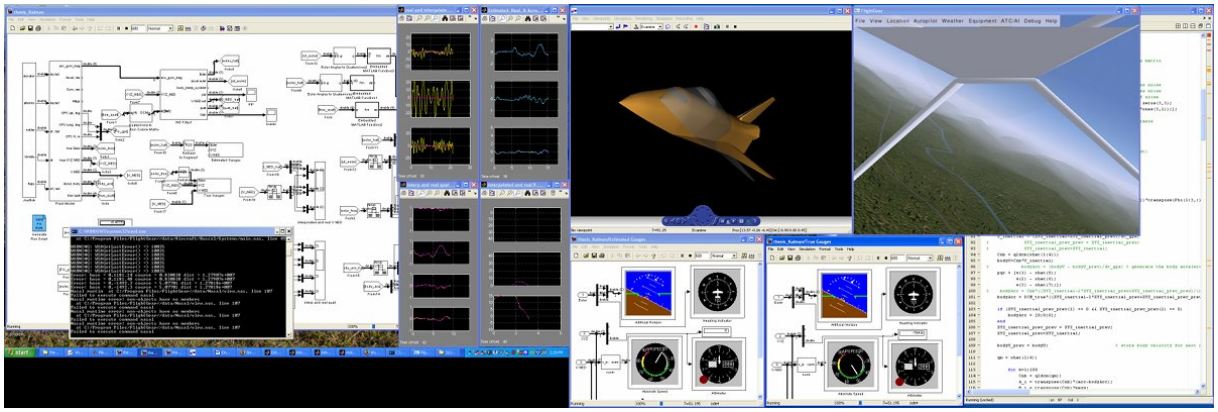


Figure 6.1: Simulation of attitude estimation algorithm

All implementations presented in Chapter 5 were simulated and have proven to work reliably. The

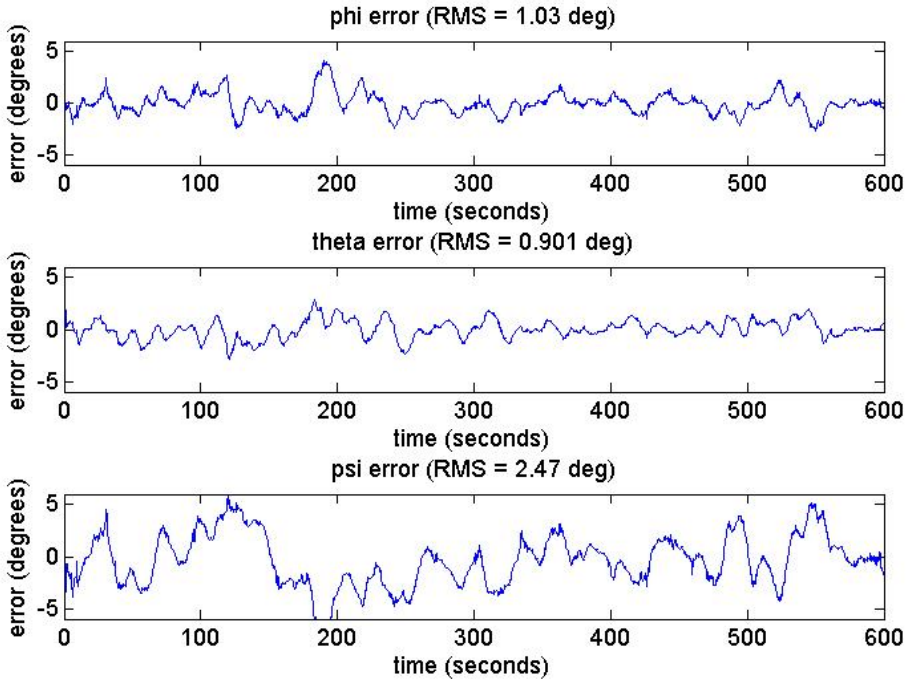
results for the error quaternion and measurement quaternion filters are presented here. Figure 6.2 shows the results of the measurement quaternion filter from a 10-minute (simulation time) virtual flight. The attitude errors are represented using Euler angles because of their clarity. Table 6.1 lists RMS errors of various error states of both the measurement and the error quaternion filters in different 10-minute simulations, showing that both filters work reliably. This table includes an error state expressed as the error quaternion $[1\ \bar{\epsilon}^T]^T = \hat{q}^* \otimes q_{true}$ because that is the error state the Kalman filter is formulated to minimize, and therefore is the most appropriate measure of tracking accuracy. Performance in the two filters is similar enough that the differences in tracking errors may be attributed to different flight paths in respective simulations.

Error state		**error quaternion filter RMS error**	**measurement quaternion filter RMS error**
euler angles	ϕ	0.857 deg	1.03 deg
	θ	0.939 deg	0.901 deg
	ψ	2.19 deg	2.47 deg
error quaternion	ϵ_1	0.00722	0.00891
	ϵ_2	0.0123	0.0135
	ϵ_3	0.0169	0.0185
gyro biases	x gyro bias	0.109 deg/s	0.114 deg/s
	y gyro bias	0.165 deg/s	0.182 deg/s
	z gyro bias	0.197 deg/s	0.184 deg/s

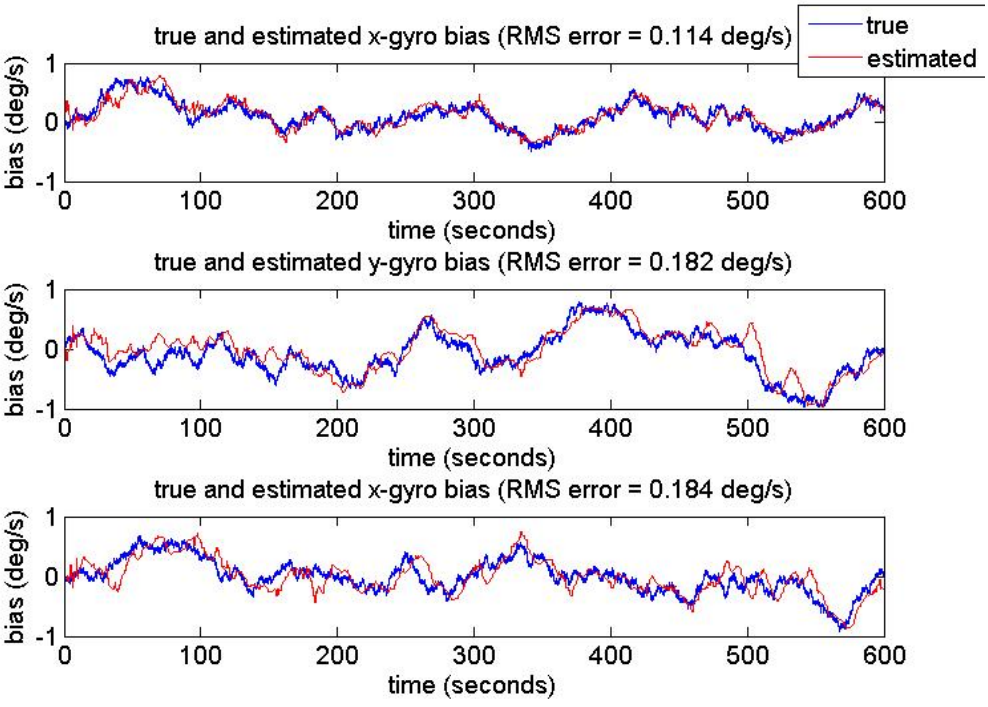
Table 6.1: Error quaternion simulation RMS errors

6.2 Benchtop Testing

The SLUGS board was designed and fabricated (see Figure 6.3 and Appendix A), and the measurement quaternion filter was embedded. Although there was no truth reference to compare to estimated attitude, to the human eye the filter seemed to perform very well (see Figure 6.4).



(a) Euler errors



(b) Bias tracking

Figure 6.2: Measurement quaternion simulation results

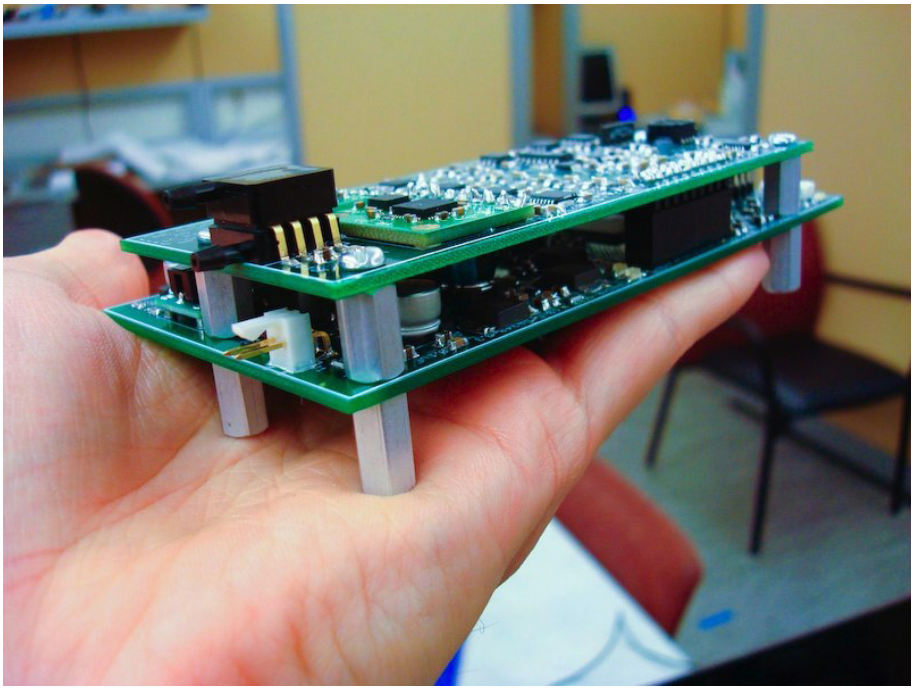


Figure 6.3: The SLUGS board

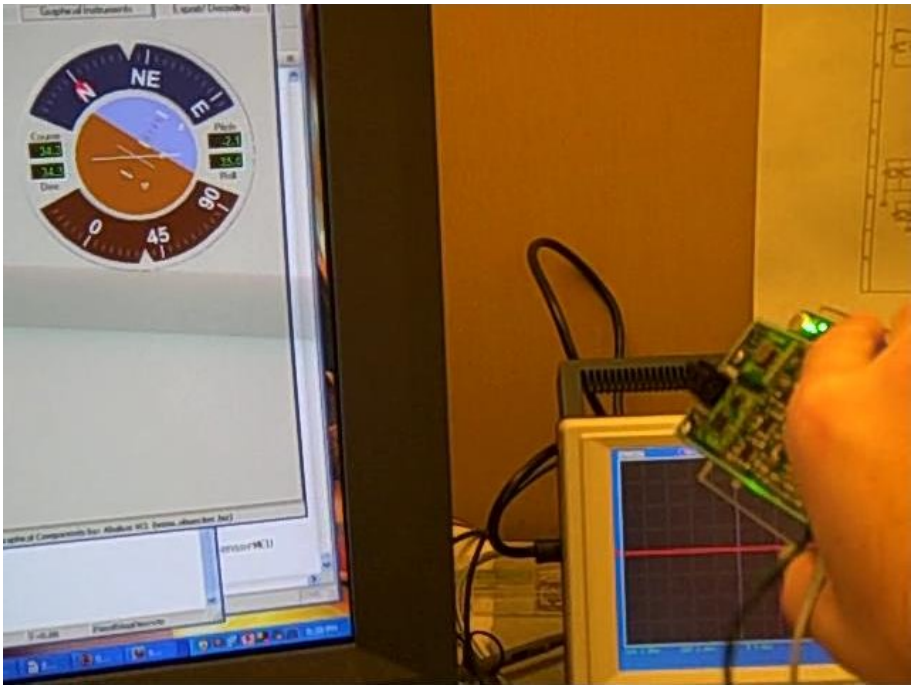
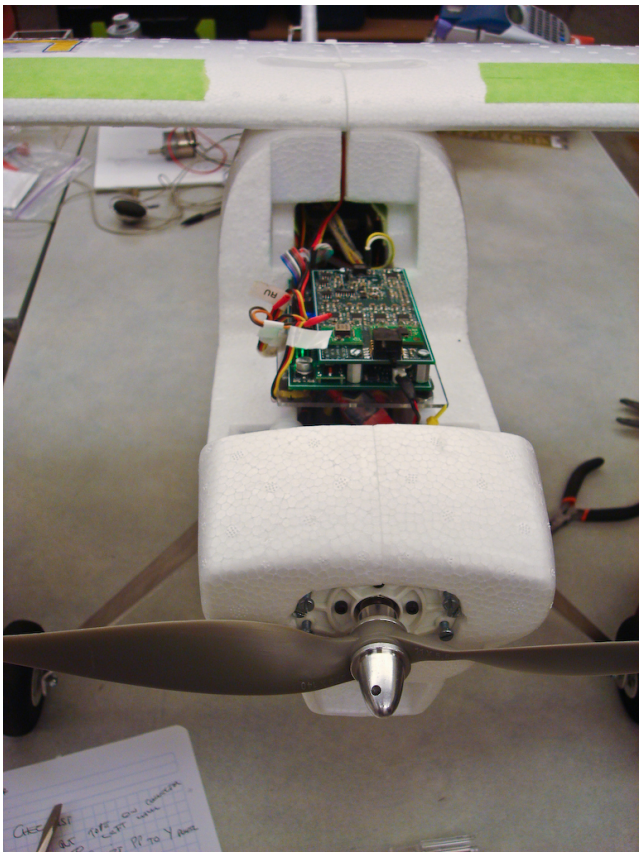


Figure 6.4: Benchtop testing of the embedded attitude estimator

6.3 Flight Testing

An aircraft based on the commercial Multiplex Mentor kit was constructed and the SLUGS board was mounted inside it (see Figure 6.5(a)). The plane was flown under full human control (see Figure 6.5(b)) while GPS and sensor data were recorded on a laptop via radio modem (Figure 6.5(c)).



(a) SLUGSv1 mounted in airplane

(b) The aircraft in flight

(c) Monitoring flight systems from the ground station

Figure 6.5: Flight testing the system

Due to unforseen vibration which coupled through the PCB mount, all sensor data were unusably noisy while the motor was running. During one flight the airplane was flown high and the motor was cut off. Though the propeller continued spinning because of the wind, the vibration was reduced enough for the ensuing 80 second glide to provide a sufficient data set for testing attitude estimation. In post-flight testing the measurement quaternion filter was run on this data set and provided very realistic-looking attitude tracking given the recorded flight path (i.e., the airplane banked left as it turned left, etc., see Figure 6.3).

The apparent success of the measurement quaternion filter on real flight data strongly indicates that the filter design is successful.

Figure 6.6: Post flight filtering

7 Conclusion

This thesis has covered the prerequisites for understanding attitude estimation using Kalman filtering and the quaternion attitude representation. The concepts of error and measurement quaternions were developed and used to implement an efficient Kalman filter. This thesis is intended both as documentation of the measurement quaternion filter, and as a reference on attitude estimation for myself and others.

I have demonstrates that the measurement quaternion filter provides accurate attitude information while maintaining relative computational simplicity. I have derived the measurement quaternion filter and proven its robustness through simulation. I have confirmed the filter's computational speed by embedding the algorithm on a dsPIC33F and getting successful benchtop attitude tracking. The filter has yielded believable attitude output from flight data recorded by SLUGS, which indicates that the SLUGS system works.

Because the measurement quaternion filter was not developed until after the flight data set was recorded, to date there are no results for live, in-flight embedded attitude estimation. However, benchtop attitude estimation has been successful, and because the successful post-flight filtering runs exactly the same code as is now embedded, I expect that the filter will work while in flight. Flight testing was suspended while I wrote this thesis and will continue in the very near future.

Advances which the measurement quaternion filter promotes include further miniaturization of UAVs, cost reduction of existing UAVs, and the expansion of UAV research and interest into a broader scientific and hobbyist community. We hope that by releasing the algorithm and hardware to the open source community, our work will contribute to the development of the next generation of inexpensive UAVs.

Appendix A SLUGSv1 PCB

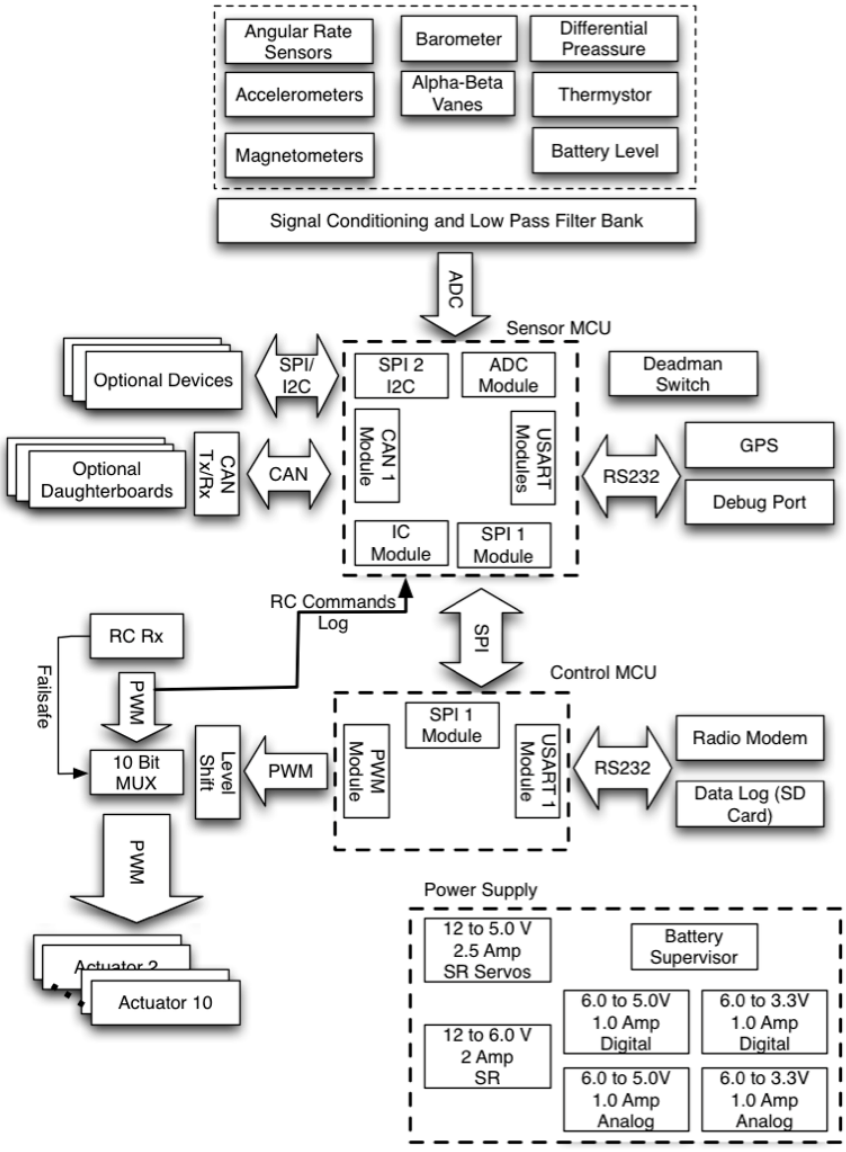


Figure A.1: SLUGSv1 block diagram (detail)

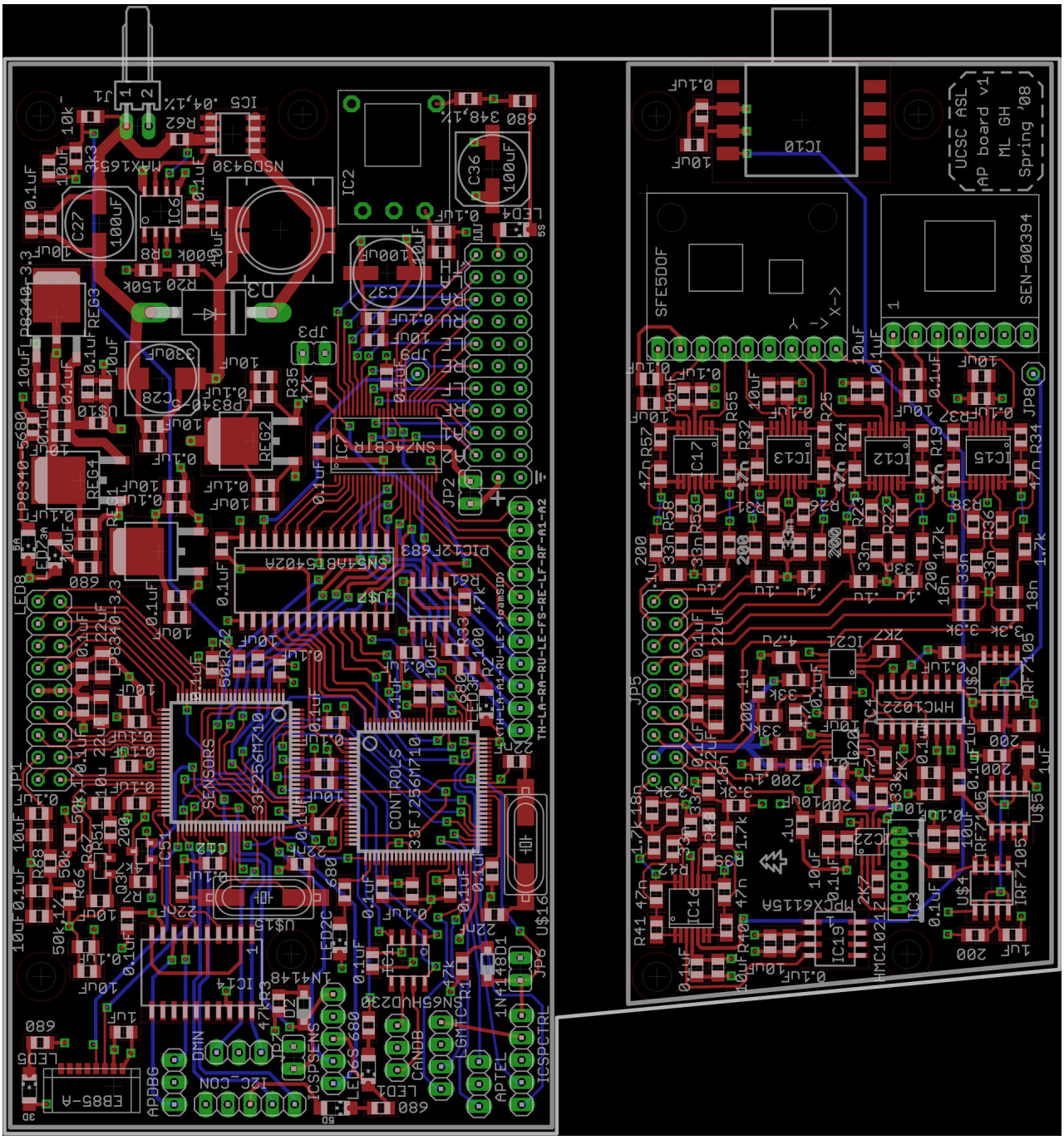


Figure A.2: SLUGSv1 layout

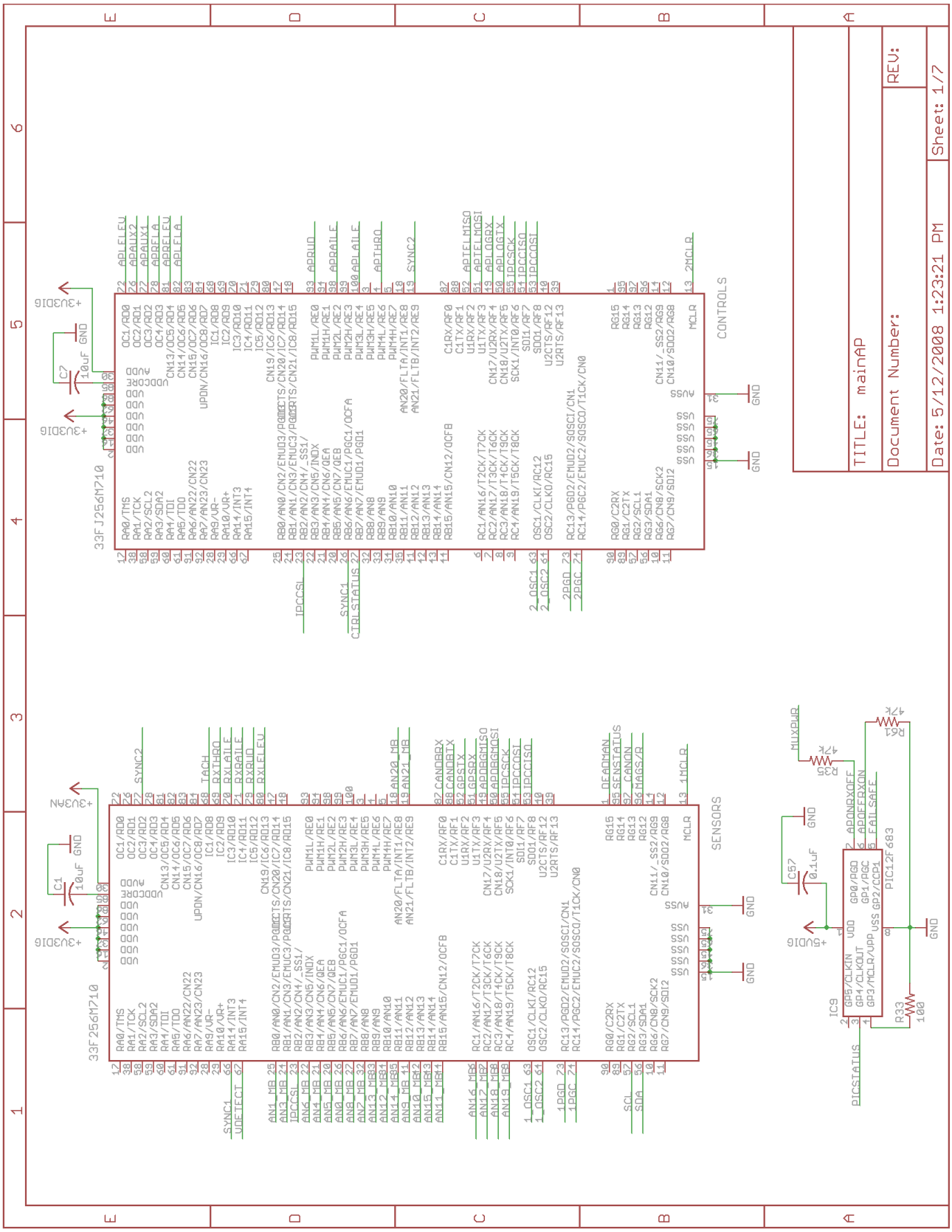


Figure A.3: SLUGSv1 schematic page 1

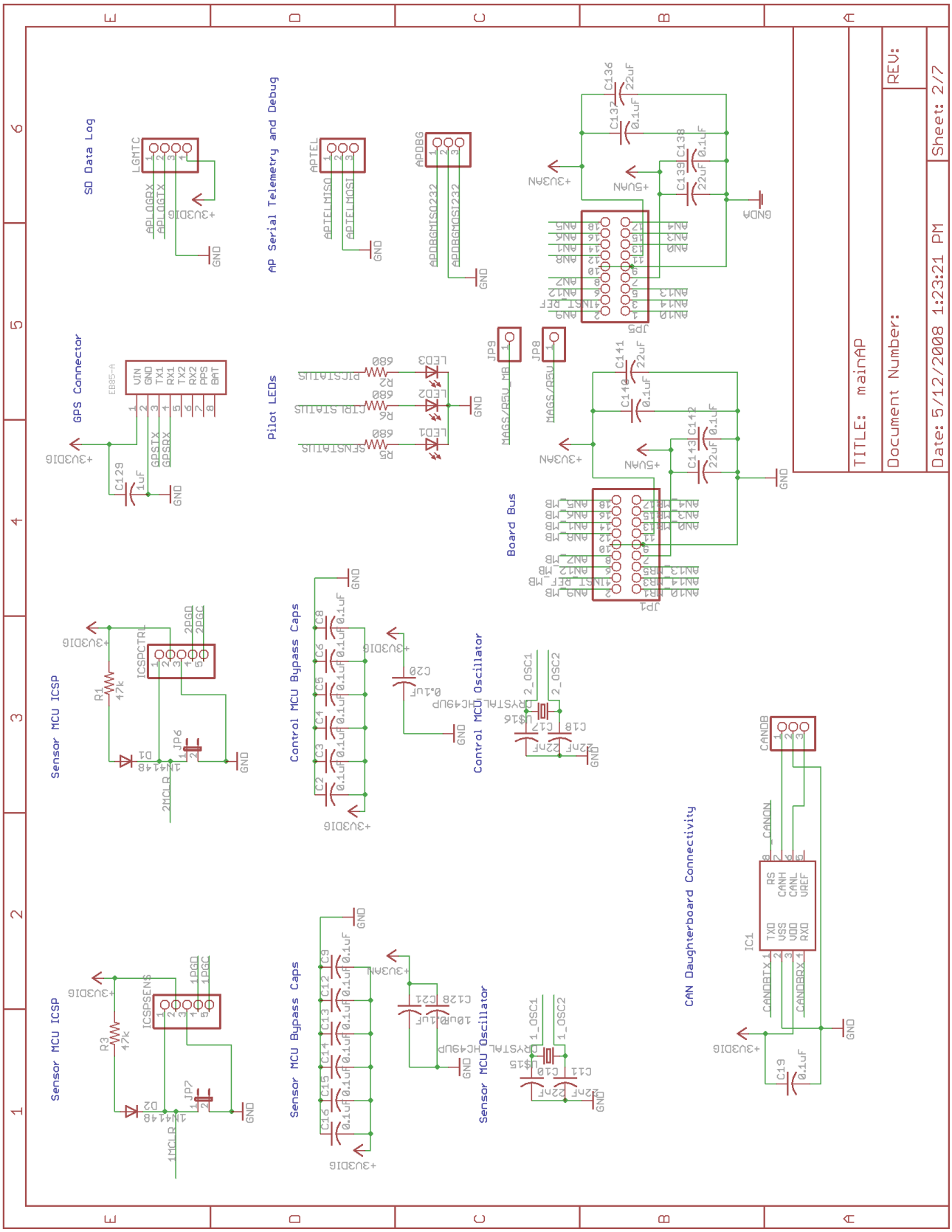


Figure A.4: SLUGSv1 schematic page 2

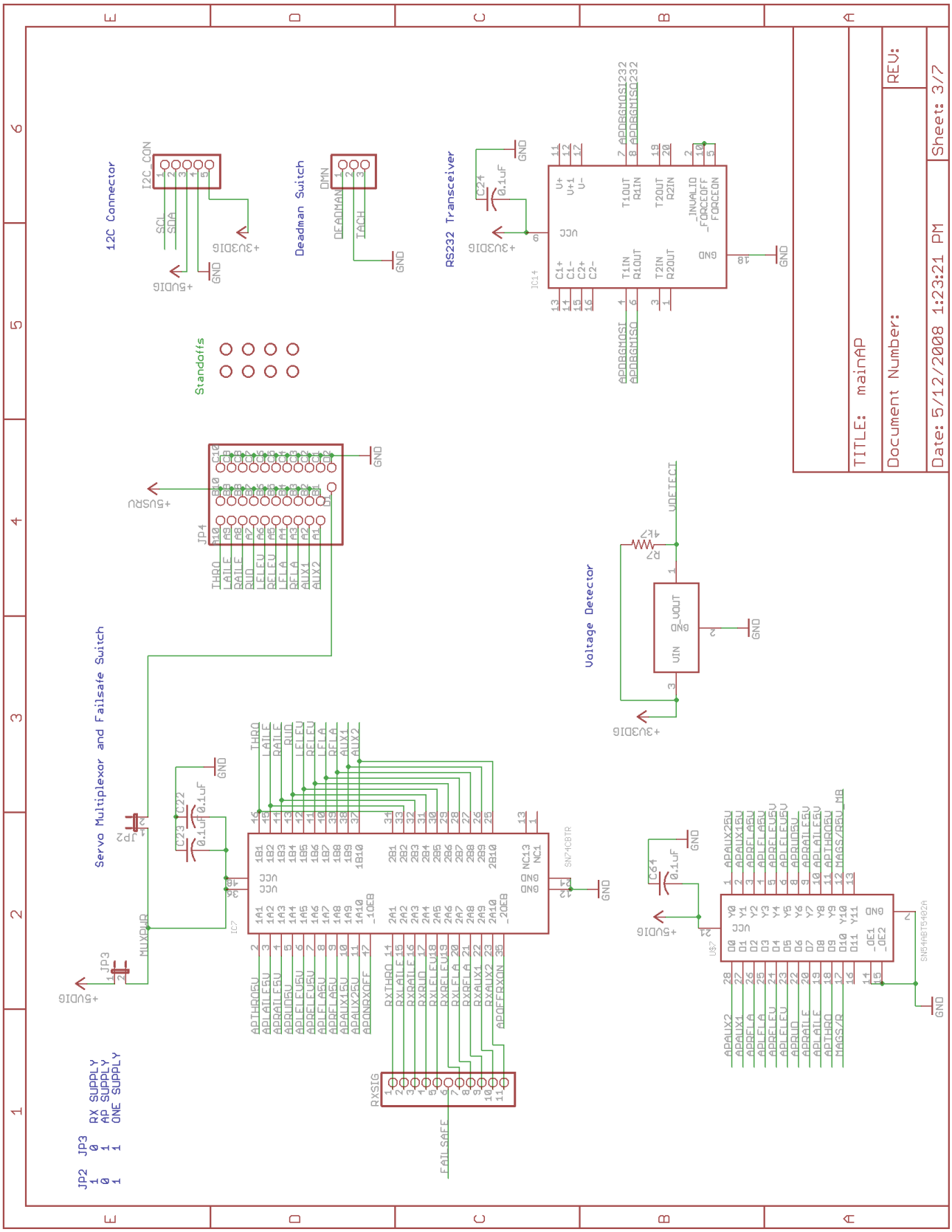


Figure A.5: SLUGSv1 schematic page 3

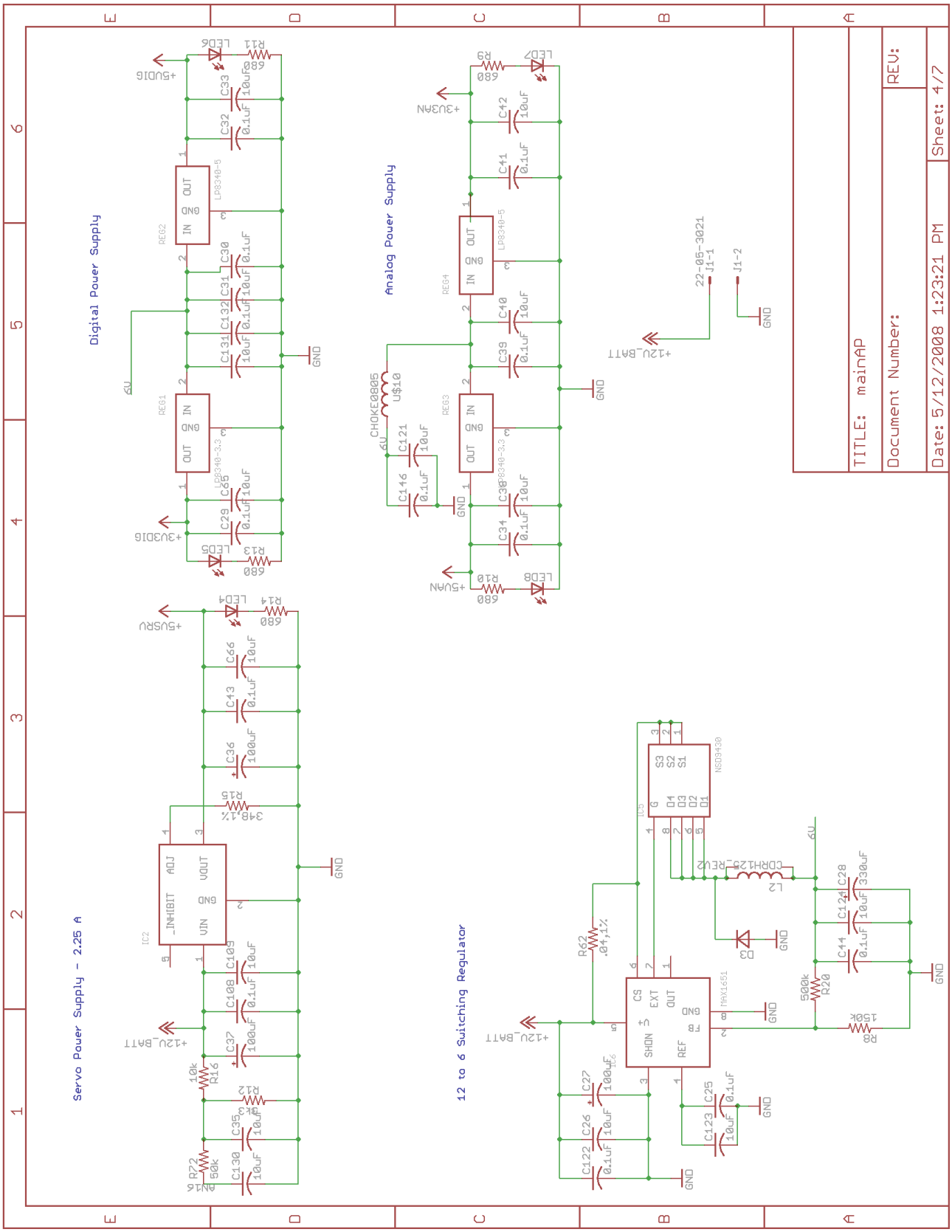


Figure A.6: SLUGSv1 schematic page 4

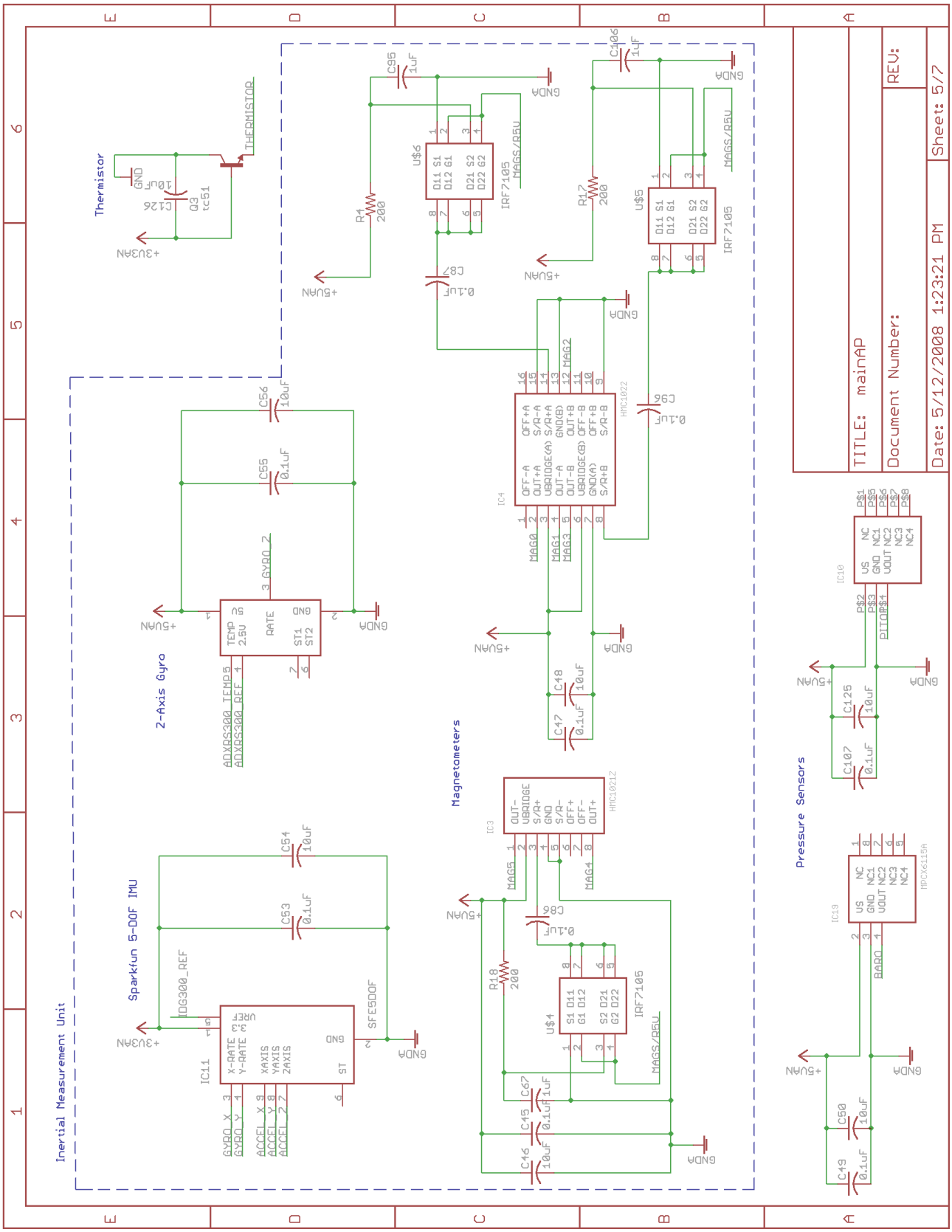


Figure A.7: SLUGSv1 schematic page 5

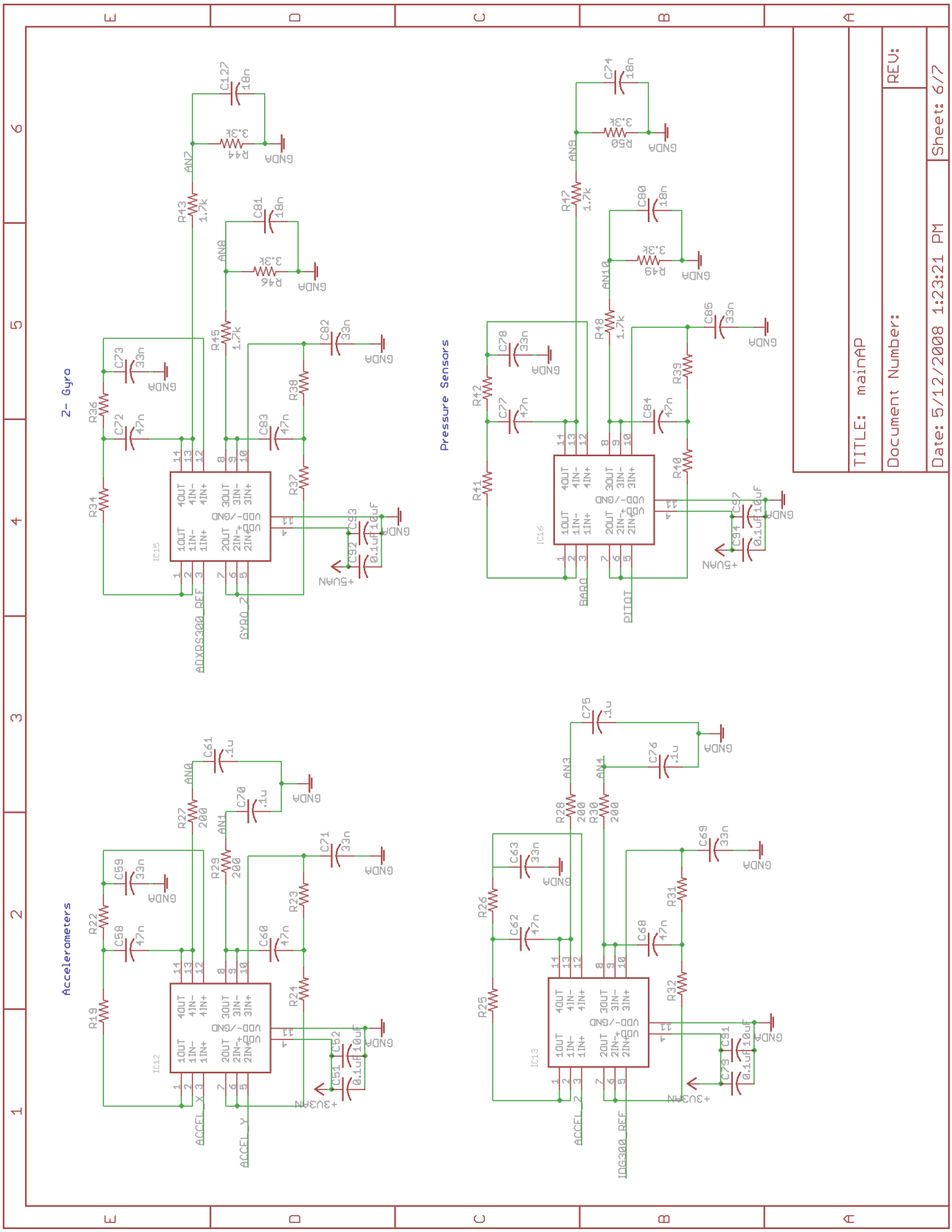


Figure A.8: SLUGSv1 schematic page 6

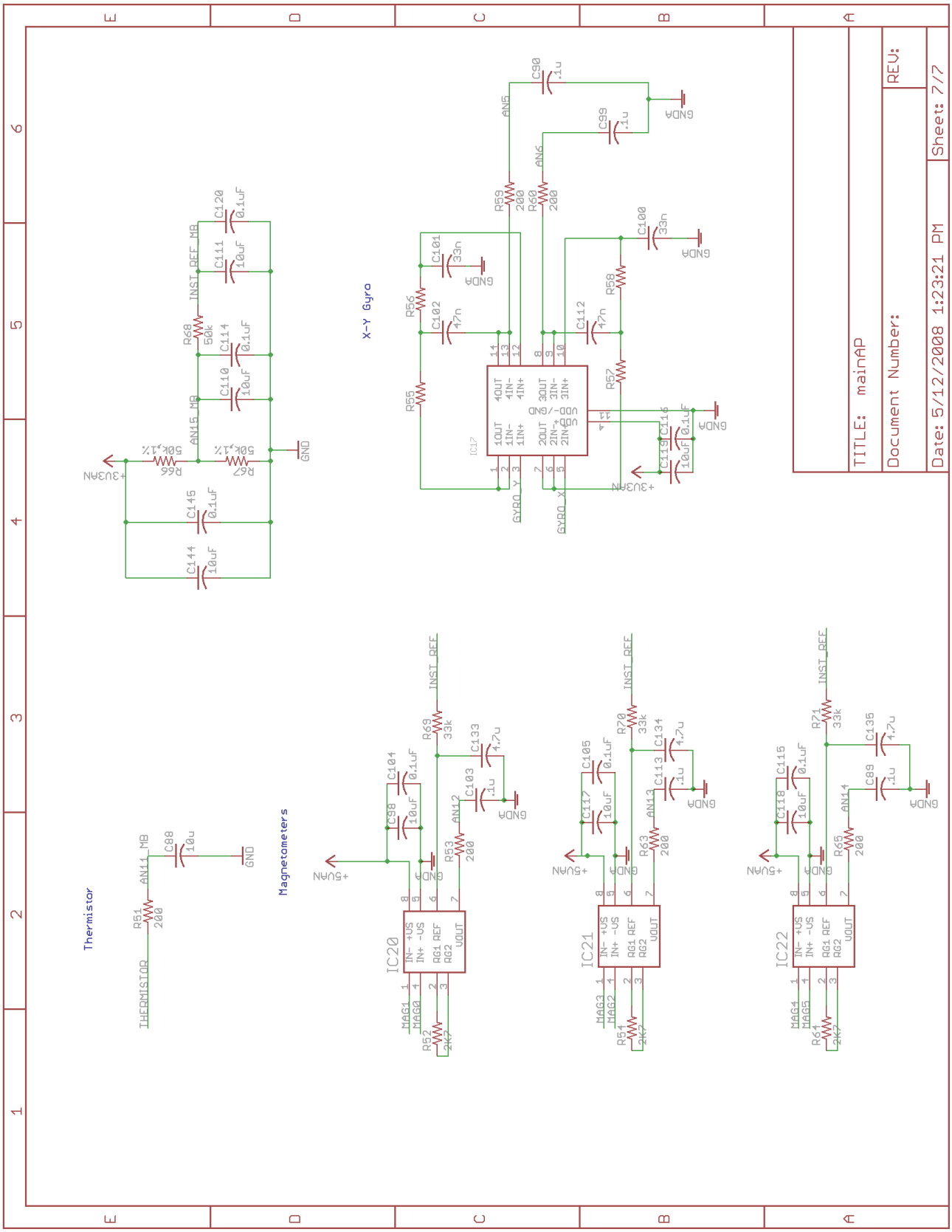


Figure A.9: SLUGSv1 schematic page 7

Appendix B MATLAB code

The following code was tested and shown to work before insertion into this appendix. However, it had to be altered to fit on the page, and some code which had been commented out and was no longer necessary was removed for readability. It is therefore unlikely but possible that an error has been introduced.

B.1 Measurement Quaternion Implementation

```
function [quatout,bias_out,V_inertial_out,euler_out,pqr_out,unfiltered_euler_out,bodyAcc_out,P_out]
    = fcn(gyros,accels,mags,dt_gps,gps_update,XYZ_inertial)

    %%*************************************************************
    %%***************** initialize variables **********************
    %%*************************************************************
    persistent xhat;
    persistent P;
persistent XYZ_inertial_prev;
    persistent XYZ_inertial_prev_prev;
    persistent unfiltered_euler;
    persistent bodyAcc;
    persistent V_inertial;
    persistent counter
    persistent gyros_old;
    if isempty(counter)
        counter = 0;
    end
    if isempty(xhat)
        xhat =[0.7934;
                    0;
                    0;
               0.6088;
                [1e-6;1e-6;1e-6];];
        P=diag( [1^2*ones(1,3) .1^2*ones(1,3)] );
        XYZ_inertial_prev = [0;0;0];
        XYZ_inertial_prev_prev = [0;0;0];
        unfiltered_euler = [0;0;0];
        bodyAcc = [0;0;0];
        V_inertial = [0;0;0];
```

```
        gps_alive = 0;
        gyros_old = gyros;
    end

    dt = 0.01; % fundamental time step
    g  = 9.81;

    B = [0.468741473405951;
         0.129010679377825;
         0.873863648240210]; %NED

    a_n = 5;
    m_n = .05;
    q_n = .6;
%     a_n = 50;
%     m_n = .5;
    Q_quat = 1e-3;
    Q_bias = 1e-2;

   tau_gyro = 100;
   exptau = exp(-.01/tau_gyro);

    R_sens = diag([a_n^2*ones(1,3) m_n^2*ones(1,3)]);
    Q = diag([Q_quat^2*ones(1,3) dt*Q_bias^2*ones(1,3)]);

    x_old=xhat;

    %%*****************************************************************
    %%******************** propogate state ***********************
    %%*****************************************************************
    counter = counter+1;

    p  = gyros_old(1)-x_old(5);
    q  = gyros_old(2)-x_old(6);
    r  = gyros_old(3)-x_old(7);

    normomega    = norm([p;q;r]);
    normomegainv = inv(normomega);
    C  = cos(.5*dt*normomega);
    Sn = sin(.5*dt*normomega)*normomegainv;

    Phi_old_small = [    C, -(p *Sn), -(q* Sn), -(r *Sn);
                  (p *Sn),        C,  (r* Sn), -(q *Sn);
                  (q* Sn), -(r *Sn),        C,  (p* Sn);
                  (r* Sn),  (q *Sn), -(p* Sn),        C;];
    x_minus=[Phi_old_small*x_old(1:4);     %(predict state)
                         x_old(5:7);];
    if x_minus(1)<0
        x_minus(1:4)=-x_minus(1:4);
    end
    x_minus(1:4) = x_minus(1:4)/norm(x_minus(1:4));  %normalize quaternion

%%*****************************************************************
```

```
    %%************ propogate covariance ***************************
    %%*************************************************************
    %old state transition matrix
    Phi_old = [1,0,0,-.5*dt,     0,     0;
               0,1,0,     0,-.5*dt,     0;
               0,0,1,     0,     0,-.5*dt;
               0,0,0,exptau,     0,     0;
               0,0,0,     0,exptau,     0;
               0,0,0,     0,     0,exptau;];

    % propagate covariance
    P = Phi_old*(P+Q)*transpose(Phi_old);

    %% ************************************************************************
    %% ******** translational filter ******************************************
    %% ************************************************************************
    %% check for gps update
if gps_update
    V_inertial = (XYZ_inertial-XYZ_inertial_prev)/dt_gps;
    Cnb = q2dcm(x_minus(1:4));

    bodyAcc = Cnb*((XYZ_inertial-2*XYZ_inertial_prev+XYZ_inertial_prev_prev)/(dt_gps^2));

    if ((XYZ_inertial_prev_prev(1) == 0) && (XYZ_inertial_prev_prev(2) == 0))
     bodyAcc = [0;0;0];
    end
    XYZ_inertial_prev_prev = XYZ_inertial_prev;
    XYZ_inertial_prev=XYZ_inertial;


    %% ************************************************************************
    %% ********************** update ******************************************
    %% ************************************************************************
    qm = xhat(1:4);
    MM=zeros(3,6);
    M = zeros(6,3);
    W = zeros(6,6);

    for k=1:1
     Cnb = q2dcm(qm);
        A_hat = Cnb*[0;0;-g];
        B_hat = Cnb*B;
        M = [2*sk(A_hat);2*sk(B_hat)]; %no noise
        W = diag([1/a_n^2*[1 1 1] 1/m_n^2*[1 1 1]]);
        MM = inv(transpose(M)*W*M)*transpose(M)*W; %with weighing

        q__err = MM*[ ((accels-bodyAcc)-A_hat) ; (mags-B_hat) ]; %no noise

        qm=quatmult(qm,[1;q__err]);

        if (qm(1)<0)
         qm = -qm;       % make scalar part positive
    end
```

```
        qm=qm/norm(qm);
    end

    unfiltered_euler=quat2eul(qm);

    z=q__err;
    % Expected measurement
    z_hat = [0;0;0];

    % Linearized measurement matrix H
    H= [eye(3,3) zeros(3,3)];
%    R= inv(transpose(M)*W*M);
    R=MM*R_sens*MM';

    K = P*H'*inv(H*P*H'+R); % kalman gain
    P = (eye(6)-K*H)*P; % update covariance
    xerror_hat = [zeros(3,1);x_old(5:7)]+K*(z-z_hat); % update state
    xhat(5:7)=xerror_hat(4:6);
    xhat(1:4)=quatmult(x_minus(1:4),[1;xerror_hat(1:3)]);
else
    xhat = x_minus;
end
    if xhat(1)<0
        xhat(1:4)=-xhat(1:4);
    end
    xhat(1:4) = xhat(1:4)./norm(xhat(1:4)); %normalize quaternion

    gyros_old = gyros; %save for next time

    %% ************************************************************************
    %% ********************* unfiltered euler angles ************************
    %% ************************************************************************
%     Gbody = accels-bodyAcc;
%     Gbody = Gbody/norm(Gbody)*.99999999;
%     Pitch = asin(Gbody(1));
%     temp = -Gbody(2)/cos(Pitch);
%     if abs(temp)<1
%      Roll = asin(-Gbody(2)/cos(Pitch));
%     else
%      Roll = pi/2*sign(temp);
%     end
%
%  if sign(Gbody(3)) >= 0   % if upside down, correct phi
%         Roll=-Roll-pi*sign(Gbody(2));
%  end
%
%     ct = cos(Pitch); %set up trig for less computation time
%     st = sin(Pitch);
%     cp = cos(Roll);
%     sp = sin(Roll);
%     Bflat = [ct, st*sp, cp*st;
%                 0,    cp,   -sp;]*mags; %rotate mags into phi=0, theta=0
%  Yaw = 0.2686 -atan2g(Bflat(2),Bflat(1)); %calculate psi
```

```
%     unfiltered_euler = [Pitch;Roll;Yaw];


    %% ************************************************************************
    %% ******************** outputs ******************************************
    %% ************************************************************************
    P_out = P;
    quatout = xhat(1:4);
    euler_out = quat2eul(quatout);
%    euler_out(1) = -euler_out(1); %reverse sign for GS
    bias_out = xhat(5:7);
    pqr_out = gyros-xhat(5:7);

    unfiltered_euler_out = unfiltered_euler;
    bodyAcc_out = bodyAcc;
    V_inertial_out = V_inertial;

end


%=====================================================================
%==================   additional functions ======================
%=====================================================================
function as=sk(a)
    % This function determines the skew-symmetric matrix
    % corresponding to a given vector a with three elements.
     as=[0 -a(3) a(2); a(3) 0 -a(1); -a(2) a(1) 0];
end

function quat = quatmult(q1,q2)
    %   Multiplies two quaternions.
    qv1 = q1(2:4);
    qs1 = q1(1);
    qv2 = q2(2:4);
    qs2 = q2(1);

    quatv = cross(qv1,qv2) + qs1*qv2 + qs2*qv1;
    quats = qs1*qs2 - dot(qv1,qv2);

    quat = [quats;quatv];
end

function quat = quatinv(q)
    %   Inverts a quaternion.
    quat = [q(1);-q(2:4)];
end

function angles = quat2eul( q )
    qin = q./norm(q);

    y=2.*(qin(3,1).*qin(4,1) + qin(1,1).*qin(2,1));
    x=qin(1,1).^2 - qin(2,1).^2 - qin(3,1).^2 + qin(4,1).^2;
    phi = atan2g(y,x);
```

```
    theta = asin(-2.*(qin(2,1).*qin(4,1) - qin(1,1).*qin(3,1)));

    y=2.*(qin(2,1).*qin(3,1) + qin(1,1).*qin(4,1));
    x=qin(1,1).^2 + qin(2,1).^2 - qin(3,1).^2 - qin(4,1).^2;
    psi = atan2g(y,x);

    angles = [phi; theta; psi];
end

function angle = atan2g(y,x)
    angle = 0.;
    if x>0
        angle = atan(y/x);
    end
    if x==0
        angle = .5*pi*sign(y);
    end
    if x<0
        angle = pi*sign(y)+atan(y/x);
    end
end

function q = eul2quat(angles)
    % converts phi,theta,phi into q
    % lifted straight from matlab's function (with a transposed output):
    cang = cos( angles/2 );
    sang = sin( angles/2 );

    q = [ cang(:,1).*cang(:,2).*cang(:,3) + sang(:,1).*sang(:,2).*sang(:,3);
           sang(:,1).*cang(:,2).*cang(:,3) - cang(:,1).*sang(:,2).*sang(:,3);
           cang(:,1).*sang(:,2).*cang(:,3) + sang(:,1).*cang(:,2).*sang(:,3);
           cang(:,1).*cang(:,2).*sang(:,3) - sang(:,1).*sang(:,2).*cang(:,3)];
end

function dcm = q2dcm( q )
    %   The direction cosine matrix performs the coordinate
    %   transformation of a vector in inertial axes to a vector in body axes.

    qin = q; % don't need to normalize because it is already done
    %qin = quatnormalize( q );

    dcm = zeros(3,3);

    dcm(1,1) = qin(1).^2 + qin(2).^2 - qin(3).^2 - qin(4).^2;
    dcm(1,2) = 2.*(qin(2).*qin(3) + qin(1).*qin(4));
    dcm(1,3) = 2.*(qin(2).*qin(4) - qin(1).*qin(3));
    dcm(2,1) = 2.*(qin(2).*qin(3) - qin(1).*qin(4));
    dcm(2,2) = qin(1).^2 - qin(2).^2 + qin(3).^2 - qin(4).^2;
    dcm(2,3) = 2.*(qin(3).*qin(4) + qin(1).*qin(2));
    dcm(3,1) = 2.*(qin(2).*qin(4) + qin(1).*qin(3));
    dcm(3,2) = 2.*(qin(3).*qin(4) - qin(1).*qin(2));
    dcm(3,3) = qin(1).^2 - qin(2).^2 - qin(3).^2 + qin(4).^2;
```

```
end
```

B.2 Error Quaternion Implementation

```
function [quatout,bias_out,V_inertial_out,euler_out,pqr_out,unfiltered_euler_out,bodyAcc_out,P_out] =
    fcn(gyros,accels,mags,dt_gps,gps_update,XYZ_inertial)

    %%*************************************************************
    %%***************** initialize variables **********************
    %%*************************************************************
    persistent xhat;
    persistent P;
persistent XYZ_inertial_prev;
    persistent XYZ_inertial_prev_prev;
    persistent unfiltered_euler;
    persistent bodyAcc;
    persistent V_inertial;
    persistent counter
    persistent gyros_old;
    if isempty(counter)
        counter = 0;
    end
    if isempty(xhat)
        xhat =[0.7934;
                    0;
                    0;
               0.6088;
                [1e-6;1e-6;1e-6];];
        P=diag( [1^2*ones(1,3) .1^2*ones(1,3)] );
        XYZ_inertial_prev = [0;0;0];
        XYZ_inertial_prev_prev = [0;0;0];
        unfiltered_euler = [0;0;0];
        bodyAcc = [0;0;0];
        V_inertial = [0;0;0];
        gps_alive = 0;
        gyros_old = gyros;
    end

    dt = 0.01; % fundamental time step
    g  = 9.81;

    B = [0.468741473405951;
         0.129010679377825;
         0.873863648240210]; %NED

    a_n = 5;
    m_n = .05;
%     a_n = 50;
%     m_n = .5;
   Q_quat = 1e-3;
   Q_bias = 1e-2;
```

```
   tau_gyro = 100;
   exptau = exp(-.01/tau_gyro);

    R = diag([a_n^2*ones(1,3) m_n^2*ones(1,3)]);
    Q = diag([Q_quat^2*ones(1,3) dt*Q_bias^2*ones(1,3)]);

    x_old=xhat;

    %%************************************************************
    %%******************** propogate state ***********************
    %%************************************************************
    counter = counter+1;

    p  = gyros_old(1)-x_old(5);
    q  = gyros_old(2)-x_old(6);
    r  = gyros_old(3)-x_old(7);

    normomega    = norm([p;q;r]);
    normomegainv = inv(normomega);
    C  = cos(.5*dt*normomega);
    Sn = sin(.5*dt*normomega)*normomegainv;

    Phi_old_small = [    C, -(p *Sn), -(q* Sn), -(r *Sn);
                   (p *Sn),        C,  (r* Sn), -(q *Sn);
                   (q* Sn), -(r *Sn),        C,  (p* Sn);
                   (r* Sn),  (q *Sn), -(p* Sn),        C;];
    x_minus=[Phi_old_small*x_old(1:4);     %(predict state)
                    exptau*x_old(5:7);];
    if x_minus(1)<0
        x_minus(1:4)=-x_minus(1:4);
    end
    x_minus(1:4) = x_minus(1:4)/norm(x_minus(1:4));  %normalize quaternion



%%************************************************************
    %%************* propogate covariance ***************************
    %%************************************************************
    %old state transition matrix
    Phi_old = [1,0,0,-.5*dt,     0,     0;
               0,1,0,     0,-.5*dt,     0;
               0,0,1,     0,     0,-.5*dt;
               0,0,0,exptau,     0,     0;
               0,0,0,     0,exptau,     0;
               0,0,0,     0,     0,exptau;];
% propogate covariance
    P = Phi_old*(P+Q)*transpose(Phi_old);

    %% *************************************************************************
    %% ******** translational filter ***********************************************
    %% *************************************************************************
    %% check for gps update
if gps_update
```

```
    V_inertial = (XYZ_inertial-XYZ_inertial_prev)/dt_gps;
    Cnb = q2dcm(x_minus(1:4));

    bodyAcc = Cnb*((XYZ_inertial-2*XYZ_inertial_prev+XYZ_inertial_prev_prev)/(dt_gps^2));

    if ((XYZ_inertial_prev_prev(1) == 0) && (XYZ_inertial_prev_prev(2) == 0))
     bodyAcc = [0;0;0];
    end
    XYZ_inertial_prev_prev = XYZ_inertial_prev;
    XYZ_inertial_prev=XYZ_inertial;


    %% ************************************************************************
    %% ******************** update ********************************************
    %% ************************************************************************
    z= [accels-bodyAcc;
        mags;];
    q0=x_minus(1);
    q1=x_minus(2);
    q2=x_minus(3);
    q3=x_minus(4);
    Bx=B(1);
    By=B(2);
    Bz=B(3);
    % Expected measurement
     z_hat = [2*g*( q0*q2 - q1*q3 );
              2*g*(-q0*q1 - q2*q3 );
                g*(-2*( q0^2 + q3^2 ) + 1 );
              Bx*(-1 + 2*q0^2 + 2*q1^2) + 2*By*(q1*q2 + q0*q3) + 2*Bz*(-q0*q2 + q1*q3);
              By*(-1 + 2*q0^2 + 2*q2^2) + 2*Bx*(q1*q2 - q0*q3) + 2*Bz*( q0*q1 + q2*q3);
              Bz*(-1 + 2*q0^2 + 2*q3^2) + 2*Bx*(q0*q2 + q1*q3) + 2*By*(-q0*q1 + q2*q3);];
    % Linearized measurement matrix H
    H=[                    0, 2*g*(-1+2*q0^2+2*q3^2), -4*g*(q0*q1+q2*q3),0,0,0;
     -2*g*(-1+2*q0^2+2*q3^2),                      0 , 4*g*(-q0*q2+q1*q3),0,0,0;
           4*g*(q0*q1+q2*q3),    -4*g*(-q0*q2+q1*q3),                  0 ,0,0,0;
      0 ,-4*Bx*(q0*q2+q1*q3)-4*By*(-q0*q1+q2*q3)-2*Bz*(-1+2*q0^2+2*q3^2),
      2*By*(-1+2*q0^2+2*q2^2)+4*Bx*(q1*q2-q0*q3)+4*Bz*( q0*q1+q2*q3),0,0,0;
     4*Bx*(q0*q2+q1*q3)+4*By*(-q0*q1+q2*q3)+2*Bz*(-1+2*q0^2+2*q3^2), 0,
     -2*Bx*(-1+2*q0^2+2*q1^2)-4*By*(q1*q2+q0*q3)-4*Bz*(-q0*q2+q1*q3),0,0,0;
     -2*By*(-1+2*q0^2+2*q2^2)-4*Bx*(q1*q2-q0*q3)-4*Bz*(q0*q1+q2*q3),
      2*Bx*(-1+2*q0^2+2*q1^2)+4*By*(q1*q2+q0*q3)+4*Bz*(-q0*q2+q1*q3),0,0,0,0;];

    K = P*transpose(H)*inv(H*P*transpose(H)+R); % kalman gain
    P = (eye(6)-K*H)*P; % update covariance
    xerror_hat = [zeros(3,1);x_old(5:7)]+K*(z-z_hat); % update state
    xhat(5:7)=xerror_hat(4:6);
    xhat(1:4)=quatmult(x_minus(1:4),[1;xerror_hat(1:3)]);

else
    xhat = x_minus;
end
    if xhat(1)<0
        xhat(1:4)=-xhat(1:4);
```

```
    end
    xhat(1:4) = xhat(1:4)./norm(xhat(1:4)); %normalize quaternion

    gyros_old = gyros; %save for next time

    %% ****************************************************************************
    %% ********************* unfiltered euler angles ************************
    %% ****************************************************************************
%     Gbody = accels-bodyAcc;
%     Gbody = Gbody/norm(Gbody)*.99999999;
%     Pitch = asin(Gbody(1));
%     temp = -Gbody(2)/cos(Pitch);
%     if abs(temp)<1
%      Roll = asin(-Gbody(2)/cos(Pitch));
%     else
%      Roll = pi/2*sign(temp);
%     end
%
%  if sign(Gbody(3)) >= 0   % if upside down, correct phi
%         Roll=-Roll-pi*sign(Gbody(2));
%  end
%
%     ct = cos(Pitch); %set up trig for less computation time
%     st = sin(Pitch);
%     cp = cos(Roll);
%     sp = sin(Roll);
%     Bflat = [ct, st*sp, cp*st;
%                0,    cp,   -sp;]*mags; %rotate mags into phi=0, theta=0
%  Yaw = 0.2686 -atan2g(Bflat(2),Bflat(1)); %calculate psi
%     unfiltered_euler = [Pitch;Roll;Yaw];


    %% ****************************************************************************
    %% ********************* outputs ************************************
    %% ****************************************************************************
    P_out = P;
    quatout = xhat(1:4);
    euler_out = quat2eul(quatout);
%    euler_out(1) = -euler_out(1); %reverse sign for GS
    bias_out = xhat(5:7);
    pqr_out = gyros-xhat(5:7);

    unfiltered_euler_out = unfiltered_euler;
    bodyAcc_out = bodyAcc;
    V_inertial_out = V_inertial;

end


%=================================================================
%=================   additional functions ======================
%=================================================================
function quat = quatmult(q1,q2)
```

```
    %   Multiplies two quaternions.
    qv1 = q1(2:4);
    qs1 = q1(1);
    qv2 = q2(2:4);
    qs2 = q2(1);

    quatv = cross(qv1,qv2) + qs1*qv2 + qs2*qv1;
    quats = qs1*qs2 - dot(qv1,qv2);

    quat = [quats;quatv];
end


function quat = quatinv(q)
    %   Inverts a quaternion.
    quat = [q(1);-q(2:4)];
end

function angles = quat2eul( q )
    qin = q./norm(q);

    y=2.*(qin(3,1).*qin(4,1) + qin(1,1).*qin(2,1));
    x=qin(1,1).^2 - qin(2,1).^2 - qin(3,1).^2 + qin(4,1).^2;
    phi = atan2g(y,x);

    theta = asin(-2.*(qin(2,1).*qin(4,1) - qin(1,1).*qin(3,1)));

    y=2.*(qin(2,1).*qin(3,1) + qin(1,1).*qin(4,1));
    x=qin(1,1).^2 + qin(2,1).^2 - qin(3,1).^2 - qin(4,1).^2;
    psi = atan2g(y,x);

    angles = [phi; theta; psi];
end

function angle = atan2g(y,x)
    angle = 0.;
    if x>0
        angle = atan(y/x);
    end
    if x==0
        angle = .5*pi*sign(y);
    end
    if x<0
        angle = pi*sign(y)+atan(y/x);
    end
end

function q = eul2quat(angles)
    % converts phi,theta,phi into q
    % lifted straight from matlab's function (with a transposed output):
    cang = cos( angles/2 );
    sang = sin( angles/2 );
```

```
    q = [ cang(:,1).*cang(:,2).*cang(:,3) + sang(:,1).*sang(:,2).*sang(:,3);
           sang(:,1).*cang(:,2).*cang(:,3) - cang(:,1).*sang(:,2).*sang(:,3);
           cang(:,1).*sang(:,2).*cang(:,3) + sang(:,1).*cang(:,2).*sang(:,3);
           cang(:,1).*cang(:,2).*sang(:,3) - sang(:,1).*sang(:,2).*cang(:,3)];
end

function dcm = q2dcm( q )
    %   The direction cosine matrix performs the coordinate
    %   transformation of a vector in inertial axes to a vector in body axes.

    qin = q; % don't need to normalize because it is already done
    %qin = quatnormalize( q );

    dcm = zeros(3,3);

    dcm(1,1) = qin(1).^2 + qin(2).^2 - qin(3).^2 - qin(4).^2;
    dcm(1,2) = 2.*(qin(2).*qin(3) + qin(1).*qin(4));
    dcm(1,3) = 2.*(qin(2).*qin(4) - qin(1).*qin(3));
    dcm(2,1) = 2.*(qin(2).*qin(3) - qin(1).*qin(4));
    dcm(2,2) = qin(1).^2 - qin(2).^2 + qin(3).^2 - qin(4).^2;
    dcm(2,3) = 2.*(qin(3).*qin(4) + qin(1).*qin(2));
    dcm(3,1) = 2.*(qin(2).*qin(4) + qin(1).*qin(3));
    dcm(3,2) = 2.*(qin(3).*qin(4) - qin(1).*qin(2));
    dcm(3,3) = qin(1).^2 - qin(2).^2 - qin(3).^2 + qin(4).^2;
end
```

B.3 Full Quaternion Implementation

```
function [quatout,bias_out,V_inertial_out,euler_out,pqr_out,unfiltered_euler_out,bodyAcc_out,P_out]
     = fcn(gyros,accels,mags,dt_gps,gps_update,XYZ_inertial)

    %%*************************************************************
    %%***************** initialize variables **********************
    %%*************************************************************
    persistent xhat;
    persistent P;
persistent XYZ_inertial_prev;
    persistent XYZ_inertial_prev_prev;
    persistent unfiltered_euler;
    persistent bodyAcc;
    persistent V_inertial;
    persistent counter
    if isempty(counter)
        counter = 0;
    end
    if isempty(xhat)
        xhat =[0.7934;
                    0;
                    0;
               0.6088;
                [.001;.001;.001];
```

```
            [.00001;.00001;.00001];];
    P=diag( [1^2*ones(1,4) .1^2*ones(1,3) .001^2*ones(1,3)] );
    XYZ_inertial_prev = [0;0;0];
    XYZ_inertial_prev_prev = [0;0;0];
    unfiltered_euler = [0;0;0];
    bodyAcc = [0;0;0];
    V_inertial = [0;0;0];
    gps_alive = 0;
end

dt = 0.01; % fundamental time step
g  = 9.81;

B = [0.468741473405951;
     0.129010679377825;
     0.873863648240210]; %NED

a_n = 4;
m_n = .05;
g_n = .00001;
Q_quat = 1e-3;
Q_pqr  = 1e-1;
Q_bias = 1e-3;

R = diag([a_n^2*ones(1,3) m_n^2*ones(1,3) g_n^2*ones(1,3)]);
Q = diag([Q_quat^2*ones(1,4) Q_pqr^2*ones(1,3) Q_bias^2*ones(1,3)]);

x_old=xhat;

%%*****************************************************************
%%******************** propogate state ***********************
%%*****************************************************************
counter = counter+1;

p  = x_old(5);
q  = x_old(6);
r  = x_old(7);

normomega    = norm([p;q;r]);
normomegainv = inv(normomega);
C  = cos(.5*dt*normomega);
Sn = sin(.5*dt*normomega)*normomegainv;

Phi_old_small = [    C, -(p *Sn), -(q* Sn), -(r *Sn);
                (p *Sn),        C,  (r* Sn), -(q *Sn);
                (q* Sn), -(r *Sn),        C,  (p* Sn);
                (r* Sn),  (q *Sn), -(p* Sn),        C;];
x_minus=[Phi_old_small*x_old(1:4);     %(predict state)
                      x_old(5:10);];
if x_minus(1)<0
    x_minus(1:4)=-x_minus(1:4);
end
x_minus(1:4) = x_minus(1:4)/norm(x_minus(1:4));  %normalize quaternion
```

```
%%************************************************************
    %%************ propogate covariance **************************
    %%************************************************************
    q0 = x_old(1);
    q1 = x_old(2);
    q2 = x_old(3);
    q3 = x_old(4);
    p  = x_old(5);
    q  = x_old(6);
    r  = x_old(7);
    omega= norm(x_old(5:7));
    C = cos(.5*dt*omega);
    S = sin(.5*dt*omega);

    %old state transition matrix
    Phi_old =    [C, -(p *S)/omega, -(q* S)/omega, -(r *S)/omega,
    -(C* dt *omega *(-q^2 *q1+p *q *q2+p* q3 *r+q1* (omega^2-r^2))
    +(dt *omega^2 *p* q0+2* q^2 *q1-2 *p *q* q2-2 *p* q3* r+2 *q1* r^2)* S)/(2 *omega^3),
    -(C* dt *omega *q* (p* q1+q* q2+q3 *r)+omega^2* (dt *q *q0+2 *q2) *S
    -2 *q *(p *q1+q *q2+q3 *r)* S)/(2 *omega^3),
    -(C *dt* omega* r *(p* q1+q *q2+q3 *r)+omega^2 *(2 *q3+dt *q0* r) *S
    -2* r* (p *q1+q *q2+q3* r) *S)/(2 *omega^3),0,0,0;
       (p *S)/omega,             C,  (r* S)/omega, -(q *S)/omega, (C *dt *omega^3 *q0
       -C *dt *omega *(q^2 *q0+p* q *q3+r *(-p *q2+q0* r))-dt* omega^2 *p *q1* S
       +2* (q^2* q0+p *q *q3+r* (-p *q2+q0* r))* S)/(2 *omega^3),
        (C *dt *omega* q* (p* q0-q* q3+q2* r)-(omega^2 *(dt* q *q1+2 *q3)
        +2 *q *(p* q0-q *q3+q2 *r))* S)/(2* omega^3), (C* dt* omega* r* (p *q0-q* q3+q2* r)
        +omega^2 *(2 *q2-dt* q1* r) *S-2* r* (p *q0-q *q3+q2* r)* S)/(2 *omega^3),0,0,0;
       (q* S)/omega, -(r *S)/omega,            C,  (p* S)/omega, (C* dt* omega *(p *(q* q0-q1 *r)
       -q3* (-omega^2+q^2+r^2))+(-p *(2 *q* q0+dt* omega^2 *q2-2 *q1* r)
       +2* q3* (q^2+r^2))* S)/(2* omega^3),
        (C *dt *omega *q* (q* q0+p* q3-q1* r)+omega^2 *(2* q0-dt *q *q2)* S
        -2 *q *(q *q0+p *q3-q1 *r) *S)/(2* omega^3), (C* dt *omega* r *(q* q0+p* q3-q1* r)
        -(2* r *(q* q0+p *q3-q1 *r)+omega^2 *(2* q1+dt* q2 *r))* S)/(2 *omega^3),0,0,0;
       (r* S)/omega,  (q *S)/omega, -(p* S)/omega,            C, (C* dt *omega* (p *(q *q1+q0 *r)
       +q2 *(-omega^2+q^2+r^2))-(p* (2 *q *q1+dt *omega^2 *q3+2 *q0 *r)
       +2* q2 *(q^2+r^2)) *S)/(2 *omega^3), (C *dt* omega* q *(q *q1-p *q2+q0* r)
       +omega^2* (2 *q1-dt* q *q3)* S-2* q *(q *q1-p *q2+q0* r) *S)/(2 *omega^3),
        (C* dt* omega* r* (q* q1-p* q2+q0* r)
        -2 *r* (q *q1-p *q2+q0* r) *S+omega^2 *(2* q0-dt* q3* r)* S)/(2* omega^3),0,0,0;
                  0,            0,            0,            0,   1,0,0,0,0,0;
                  0,            0,            0,            0,   0,1,0,0,0,0;
                  0,            0,            0,            0,   0,0,1,0,0,0;
                  0,            0,            0,            0,   0,0,0,1,0,0;
                  0,            0,            0,            0,   0,0,0,0,1,0;
                  0,            0,            0,            0,   0,0,0,0,0,1;];
    % propogate covariance
    P = Phi_old*(P+Q)*transpose(Phi_old);


    %% ************************************************************************
```

```
    %% ******** translational filter ***************************************
    %% ********************************************************************
    %% check for gps update
if gps_update
    V_inertial = (XYZ_inertial-XYZ_inertial_prev)/dt_gps;
    Cnb = q2dcm(x_minus(1:4));

    bodyAcc = Cnb*((XYZ_inertial-2*XYZ_inertial_prev+XYZ_inertial_prev_prev)/(dt_gps^2));

    if ((XYZ_inertial_prev_prev(1) == 0) && (XYZ_inertial_prev_prev(2) == 0))
     bodyAcc = [0;0;0];
    end
    XYZ_inertial_prev_prev = XYZ_inertial_prev;
    XYZ_inertial_prev=XYZ_inertial;

    %% ********************************************************************
    %% ********************* update ***************************************
    %% ********************************************************************
    z= [accels-bodyAcc;
        mags;
        gyros];

    q0=x_minus(1);
    q1=x_minus(2);
    q2=x_minus(3);
    q3=x_minus(4);
    Bx=B(1);
    By=B(2);
    Bz=B(3);
    % Expected measurement
    z_hat = [2*g*( q0*q2 - q1*q3 );
             2*g*(-q0*q1 - q2*q3 );
               g*(-2*( q0^2 + q3^2 ) + 1 );
             Bx*(-1 + 2*q0^2 + 2*q1^2) + 2*By*(q1*q2 + q0*q3) + 2*Bz*(-q0*q2 + q1*q3);
             By*(-1 + 2*q0^2 + 2*q2^2) + 2*Bx*(q1*q2 - q0*q3) + 2*Bz*( q0*q1 + q2*q3);
             Bz*(-1 + 2*q0^2 + 2*q3^2) + 2*Bx*(q0*q2 + q1*q3) + 2*By*(-q0*q1 + q2*q3);
               x_minus(5)+x_minus(8);
               x_minus(6)+x_minus(9);
               x_minus(7)+x_minus(10);];
    % Linearized measurement matrix H
    H = [        2*g*q2,            -2*g*q3,             2*g*q0,            -2*g*q1, 0,0,0,0,0,0;
                -2*g*q1,            -2*g*q0,            -2*g*q3,            -2*g*q2, 0,0,0,0,0,0;
                -4*g*q0,                  0,                  0,            -4*g*q3, 0,0,0,0,0,0;
4*Bx*q0-2*Bz*q2+2*By*q3, 4*Bx*q1+2*By*q2+2*Bz*q3, -2*Bz*q0+2*By*q1,  2*By*q0+2*Bz*q1, 0,0,0,0,0,0;
4*By*q0+2*Bz*q1-2*Bx*q3,  2*Bz*q0+2*Bx*q2, 2*Bx*q1+4*By*q2+2*Bz*q3, -2*Bx*q0+2*Bz*q2, 0,0,0,0,0,0;
4*Bz*q0-2*By*q1+2*Bx*q2, -2*By*q0+2*Bx*q3,  2*Bx*q0+2*By*q3, 2*Bx*q1+2*By*q2+4*Bz*q3, 0,0,0,0,0,0;
                      0,                  0,                  0,                   0, 1,0,0,1,0,0;
                      0,                  0,                  0,                   0, 0,1,0,0,1,0;
                      0,                  0,                  0,                   0, 0,0,1,0,0,1;];

    K = P*transpose(H)*inv(H*P*transpose(H)+R); % kalman gain
    P = (eye(10)-K*H)*P; % update covariance
    xhat = x_minus+K*(z-z_hat); % update state
```

```
else
    z2=gyros;
    z2_hat = [x_minus(5)+x_minus(8);
              x_minus(6)+x_minus(9);
              x_minus(7)+x_minus(10);];
    H2 = [0,0,0,0,1,0,0,1,0,0;
          0,0,0,0,0,1,0,0,1,0;
          0,0,0,0,0,0,1,0,0,1;];
    K2 = P*transpose(H2)*inv(H2*P*transpose(H2)+R(7:9,7:9)); % kalman gain
    P = (eye(10)-K2*H2)*P; % update covariance
    xhat = x_minus+K2*(z2-z2_hat); % update state
end
    if xhat(1)<0
        xhat(1:4)=-xhat(1:4);
    end
    xhat(1:4) = xhat(1:4)./norm(xhat(1:4)); %normalize quaternion



    %% ************************************************************************
    %% ********************* unfiltered euler angles ************************
    %% ************************************************************************
%     Gbody = accels-bodyAcc;
%     Gbody = Gbody/norm(Gbody)*.99999999;
%     Pitch = asin(Gbody(1));
%     temp = -Gbody(2)/cos(Pitch);
%     if abs(temp)<1
%      Roll = asin(-Gbody(2)/cos(Pitch));
%     else
%      Roll = pi/2*sign(temp);
%     end
%
%  if sign(Gbody(3)) >= 0   % if upside down, correct phi
%         Roll=-Roll-pi*sign(Gbody(2));
%  end
%
%     ct = cos(Pitch); %set up trig for less computation time
%     st = sin(Pitch);
%     cp = cos(Roll);
%     sp = sin(Roll);
%     Bflat = [ct, st*sp, cp*st;
%              0,    cp,   -sp;]*mags; %rotate mags into phi=0, theta=0
%  Yaw = 0.2686 -atan2g(Bflat(2),Bflat(1)); %calculate psi
%     unfiltered_euler = [Pitch;Roll;Yaw];


    %% ************************************************************************
    %% ********************* outputs ****************************************
    %% ************************************************************************
    P_out = P;
    quatout = xhat(1:4);
    euler_out = quat2eul(quatout);
%    euler_out(1) = -euler_out(1); %reverse sign for ground station
```

```
    bias_out = xhat(8:10);
    pqr_out = xhat(5:7);

    unfiltered_euler_out = unfiltered_euler;
    bodyAcc_out = bodyAcc;
    V_inertial_out = V_inertial;

end


%=====================================================================
%==================   additional functions ======================
%=====================================================================
function angles = quat2eul( q )
    qin = q./norm(q);

    y=2.*(qin(3,1).*qin(4,1) + qin(1,1).*qin(2,1));
    x=qin(1,1).^2 - qin(2,1).^2 - qin(3,1).^2 + qin(4,1).^2;
    phi = atan2g(y,x);

    theta = asin(-2.*(qin(2,1).*qin(4,1) - qin(1,1).*qin(3,1)));

    y=2.*(qin(2,1).*qin(3,1) + qin(1,1).*qin(4,1));
    x=qin(1,1).^2 + qin(2,1).^2 - qin(3,1).^2 - qin(4,1).^2;
    psi = atan2g(y,x);

    angles = [phi; theta; psi];
end

function angle = atan2g(y,x)
    angle = 0.;
    if x>0
        angle = atan(y/x);
    end
    if x==0
        angle = .5*pi*sign(y);
    end
    if x<0
        angle = pi*sign(y)+atan(y/x);
    end
end

function q = eul2quat(angles)
    % converts phi,theta,phi into q
    % lifted straight from matlab's function (with a transposed output):
    cang = cos( angles/2 );
    sang = sin( angles/2 );

    q = [ cang(:,1).*cang(:,2).*cang(:,3) + sang(:,1).*sang(:,2).*sang(:,3);
           sang(:,1).*cang(:,2).*cang(:,3) - cang(:,1).*sang(:,2).*sang(:,3);
           cang(:,1).*sang(:,2).*cang(:,3) + sang(:,1).*cang(:,2).*sang(:,3);
           cang(:,1).*cang(:,2).*sang(:,3) - sang(:,1).*sang(:,2).*cang(:,3)];
end
```

```
function dcm = q2dcm( q )
    %   The direction cosine matrix performs the coordinate
    %   transformation of a vector in inertial axes to a vector in body axes.

    qin = q; % don't need to normalize because it is already done
    %qin = quatnormalize( q );

    dcm = zeros(3,3);

    dcm(1,1) = qin(1).^2 + qin(2).^2 - qin(3).^2 - qin(4).^2;
    dcm(1,2) = 2.*(qin(2).*qin(3) + qin(1).*qin(4));
    dcm(1,3) = 2.*(qin(2).*qin(4) - qin(1).*qin(3));
    dcm(2,1) = 2.*(qin(2).*qin(3) - qin(1).*qin(4));
    dcm(2,2) = qin(1).^2 - qin(2).^2 + qin(3).^2 - qin(4).^2;
    dcm(2,3) = 2.*(qin(3).*qin(4) + qin(1).*qin(2));
    dcm(3,1) = 2.*(qin(2).*qin(4) + qin(1).*qin(3));
    dcm(3,2) = 2.*(qin(3).*qin(4) - qin(1).*qin(2));
    dcm(3,3) = qin(1).^2 - qin(2).^2 - qin(3).^2 + qin(4).^2;
end
```

Bibliography

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME Journal of Basic Engineering*, no. 82 (Series D), pp. 35–45, 1960.

[2] M. S. Grewal and A. P. Andrews, *Kalman Filtering : Theory and Practice Using MATLAB.* Wiley-Interscience, January 2001.

[3] G. F. Franklin, D. J. Powell, and M. L. Workman, *Digital Control of Dynamic Systems (3rd Edition).* Prentice Hall, December 1997.

[4] M. Sidi, *Spacecraft Dynamics and Control*, p. 323. Cambridge: Cambridge University Press, 1997.

[5] J. B. Kuipers, *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality.* Princeton University Press, August 2002.

[6] D. Gebre-Egziabher and G. Elkaim, "Mav attitude determination by vector matching," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 44, pp. 1012–1028, July 2008.

[7] M. I. Lizarraga, "Autonomous landing system for a UAV," Master's thesis, Naval Postgraduate School, Monterey, CA, USA., March 2004.