The Design and Implementation of a Robust AHRS for Integration into a Quadrotor Platform.

http://www.botched.co.uk/384/

MATTHEW WATSON
100169708

MENG ELECTRONIC ENGINEERING

DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING

MAY 2013

SUPERVISOR DR LUKE SEED
SECOND MARKER DR CHEE HING TAN

WORD COUNT 8202

Abstract

The quadrotor is a small to medium sized unmanned aerial vehicle (UAV) that has seen a massive surge in interest in recent years, both in military and commercial applications. This is down to a combination of its mechanical simplicity, its stability, and its low cost compared to similar platforms.

A core component of a quadrotor, or any UAV, is an attitude and heading reference system, or AHRS for short. This provides a low noise high bandwidth estimate of the vehicles orientation in space, achieved by mathematically combining data from multiple sensor sources. This project follows the development and implementation of such an AHRS, plus its integration into a quadrotor platform, which will also be designed and built as part of this project. The final goal is to enable the quadrotor to hover for short durations without any pilot input, without a large degree of lateral acceleration.

This was achieved through a combination of effective sensor fusion and tuned control loops, implemented in real-time onboard the quadrotor, successfully achieving very stable stationary flight. The sensor fusion was performed by a 7 state 9 input extended Kalman filter, and the onboard processing was provided by a Raspberry Pi single board computer running Debian Linux. This marks the first quadrotor to be fully controlled by a Raspberry Pi single board computer, and possibly the first non-military platform to be controlled from within the Linux OS.

Contents

1 Introduction

1.1 Background

The quadrotor is a small to medium sized unmanned aerial vehicle (UAV) that has seen a massive surge in interest in recent years, both in military and commercial applications. This popularity is due to the quadrotor's mechanical simplicity, its high stability, and its low cost.

A core component of a quadrotor, or any UAV, is an attitude and heading reference system, or AHRS for short. This provides a low noise high bandwidth estimate of the vehicles orientation in space, achieved by mathematically combining data from multiple sensor sources. This project follows the development and implementation of such an AHRS, plus its integration into a quadrotor platform, which will also be designed and built as part of this project. The final goal is to enable the quadrotor to hover for short durations without any pilot input, without drifting at too great of a speed away from its starting position.

1.2 Specification

Given the complexity of a complete quadrotor system it can be very difficult to numerically quantify its performance, especially given the large sensitivity to environmental conditions such as wind and room layout. Therefore, the performance of this quadrotor will be visually evaluated relative to existing commercial flight controllers.

While performance criteria are difficult to define, there still exists a set of features a successful quadrotor must possess:

- The AHRS must be able to operate in any orientation without malfunctioning.
- The quadrotor must be able to hover for brief durations with very little pilot input.
- The quadrotor must prove equivalent in performance to common low end flight controllers, ideally exceeding them.

1.3 Risks

Given the dangerous nature of flying propellers, a number of safety features need to be implemented. The pilot must always have the ability to completely stop the motors at a moment's notice, ideally from a switch on the RC transmitter. It must also be impossible for the motors to stay active after a crash of the flight controller software. The implementation of these safety features are described in section 3.7.

1.4 Existing products

In just the past few years the commercial multicopter market has exploded, with the first ready to fly models recently hitting the market. The dominant flight controllers are produced by the Chinese company DJI Innovations and the German company MikroKopter, with a large number of open source open hardware projects available for the more DIY inclined, dominated by the MultiWii (1) and ArduCopter (2) projects. The best performing controller at the time of this reports writing is the AutoQuad flight controller (3). This controller sets the upper bar for quadrotor performance, which I would ideally like to emulate in this project, but the other DIY projects such as the ArduCopter would make for a more realistic target.

All of these flight controllers are based upon embedded microcontrollers, generally of the Atmel AVR variety. This allows for excellent real-time performance, but misses out on some the advantages offered by a full operating system such as Linux. Using such an OS allows for easy peripheral integration, a multitude of different debugging methods, and easier much easier file and log management. This project marks the first quadrotor fully controlled by Raspberry Pi single board computer running Linux, and one of the first to be fully controlled from within a Linux environment all together. This potentially opens the way for greatly reduced quadrotor development times thanks to the rich development and debugging environment the Linux OS offers.

2 The Quadrotor

2.1 Quadrotor mechanics

A quadrotor uses four propellers to generate lift. The lift generated from each motor can be individually controlled, allowing for a full range of rotational. Pitching and rolling are achieved by equally increasing and decreasing the speed of one opposing pair of props, while keeping the remaining pair constant, as shown below. Translational acceleration is achieved by maintaining a non-level attitude.

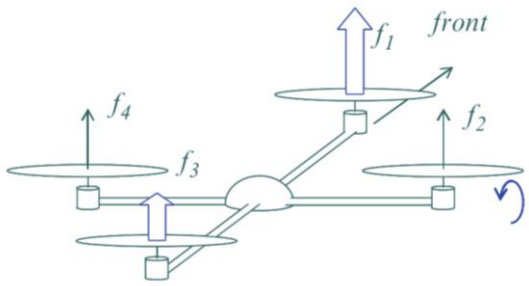

Figure 1 - Quadrotor pitching up, f1>f3, f4=f2 (4)

Yaw control is made possible through using clockwise spinning positively pitched blades on one pair of opposing motors, with CCW spinning negative pitch props on the other opposing pair, as shown in Figure 2. Each of these propellers produces a torque in the yaw axis opposite to its direction of rotation. When an equal speed is applied to each propeller these individual torques sum to zero, but by varying the thrust ratio between the two opposing pairs of blades a net torque can be produced, rotating the quadrotor. Altering each pair by an equal amount allows this to be achieved without affecting the total generated thrust.

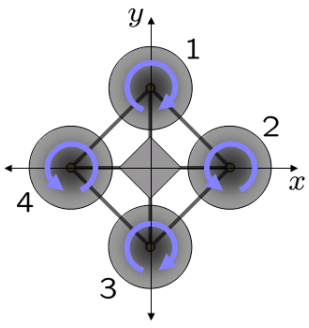

Figure 2 - Quadrotor yaw control (5)

2.2 Quadrotor Control

A quadrotor can be controlled by its pilot in a variety of ways, generally with the level of autonomy being inversely proportional to the level of pilot skill required.

The most basic form of control is known as rate control. This is analogous to the control of a traditional helicopter, with the pilot controlling throttle and three angular rate demands, in the pitch roll and yaw axes. The quadrotor then attempts to match the measured angular rates provided by the onboard gyroscopes with the pilot's demanded rates.

The next step up from this is known as attitude control. In this mode the quadrotor uses two layers of control to match its attitude to an attitude demand provided by the pilot. The yaw axis is still controlled using rate control, as this is a more intuitive interface than having the quadrotor control its own yaw angle. This is the control scheme the quadrotor in this paper is using, with the ability to revert into rate control using a toggle switch on the transmitter.

The integration of a GPS module allows for two more control layers to be implemented; velocity control and position control. These two layers would generally be implemented simultaneously, with the pilot being taken completely out of the loop. This allows for features such as position hold and waypoint following, all the way up to fully autonomous navigation.

3 Hardware

Figure 3 - Completed platform

3.1 Frame

The main quadrotor framework was bought as a kit, sold under the name 'Lunar Lander 4'. The center section and landing gear are assembled from a CNC machined fiberglass composite, with aluminium 12mm square tubes used for the four motor arms. This forms a very durable structure, as at least a few crashes were expected during this quadrotor's development.

3.2 Motors

This quadrotor uses four identical Turnigy L2215J-900 motors (6). These are brushless outrunner motors, meaning the outer casing functions as the rotor. They are rated at 200W drawing a peak of 18A, with a Kv constant of 900rpm/V. The motor winding consists of three phases, which are alternately driven by a DC voltage to produce commutation.

Figure 4 – One of four brushless motors and its attached propeller

3.3 Motor controllers

Since the motors used are brushless, commutation must be handled electronically. This is done using an electronic speed controller (ESC). The ESC applies a voltage across two of the motor phases at a time, while measuring the back EMF generated in the third phase, allowing the timing of the zero crossing to be measured in order for the timing of the next commutation cycle to be calculated. This driving voltage is then pulse width modulated in relation to an input 50Hz PWM signal, in this case being supplied by a microcontroller.

Since these ESC's are designed to be controlled by a RC receiver, they are usually only able to accept PWM pulses at a rate of 50Hz. This is too slow for optimal stability in a quadrotor, so a custom firmware was installed by manually connecting to the programming pins on the ATMEGA8 microcontroller the ESC's use. This improved firmware allows PWM data to be sent at a rate of 400Hz, and removes the low pass filter some models implement on the PWM input. This greatly improves the step response of the motor, translating to more stable flight.

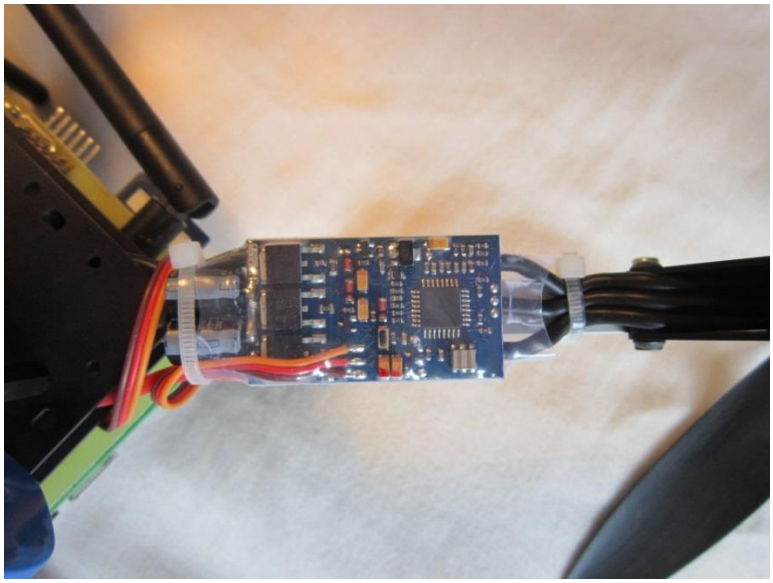

Figure 5 – Brushless ESC (electronic speed controller), with programming pads visible on the far right of the PCB

3.4 Battery

Due to the importance of the thrust to weight ratio of the quadrotor, only lithium polymer batteries are used. These represent the best energy density commercially available in a battery, with the chosen model providing 5AH at between 11.1V and 12.6V with a weight of 404g (7). This is a 20C battery, which equates to a maximum constant current draw of 100A, leaving at least 28A headroom even when running the motors at full power.

3.5 RC Radio Transmitter/Receiver

A standard RC transmitter/receiver combination allows the pilot to send commands to the quadrotor in the form of six 50Hz PWM widths. The left stick represents throttle and yaw rate, and the right stick either represents pitch and roll angles or pitch and roll rates.

Figure 6 - The RC transmitter used

3.6 Flight Controller

At the heart of the quadrotor is a Raspberry Pi single board computer running Raspbian, a free open source Linux distribution released under the GNU GPL. This distribution is a port of Debian, designed

to fully utilise the hardware floating point module available on the Raspberry Pi's ARM11 core. This is connected to a FreeIMU v4.3 sensor board and a Microchip dspic30f microcontroller via a 400 KHz I^2C bus. This is a two wire serial interface comprising of one data line, SDA, and a clock line, SCL. Although this protocol supports multi-master operation, this application uses the Raspberry Pi as the single master. The FreeIMU functions as a breakout board for a triaxis MPU6050 gyroscope/accelerometer combination, a triaxis HMC5883L magnetometer, and an MS5611-01BA high resolution barometer.

The board was initially intended to be powered by an Analogue Devices ADP2302 buck converter, but transient current spikes from the Wi-Fi adaptor proved large enough to repeatedly trip the IC's overcurrent protection. This was solved by powering the board via the linear 5V voltage regulators built into each ESC. This increases power usage by around 2W, a negligible difference when compared to the combined 800W peak draw by the motors.

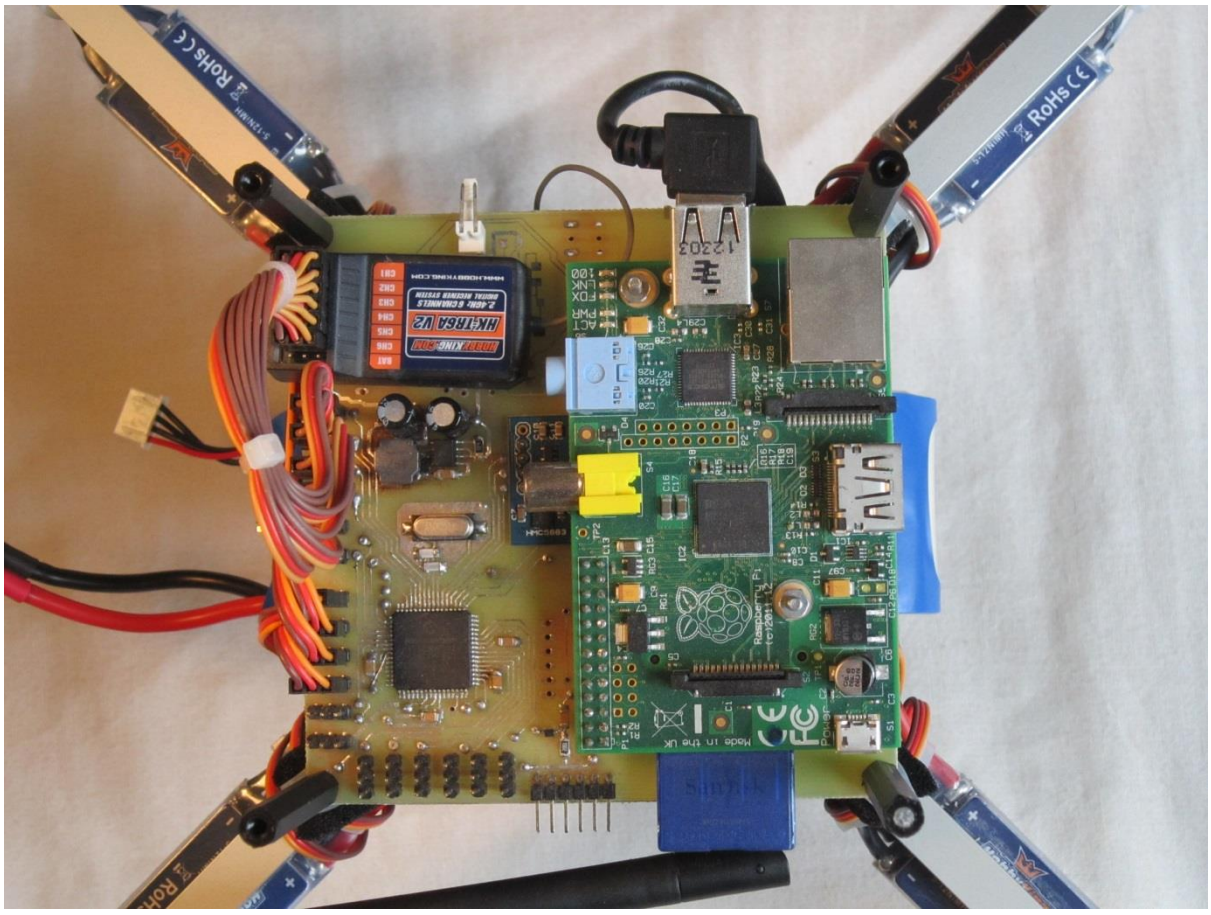

Figure 7 - Flight control board. Right - RPi, center - FreeIMU, bottom left - dsPIC microcontroller, top left - RC receiever. Center – FreeIMU v4.3

3.7 PIC PWM Interface

The control board's dspic30f microcontroller is used to provide a true real-time interface to the ESCs and the radio receiver. This is required due to the Raspberry Pi's lack of GPIO and true real-time support, meaning generating and reading PWM signals across 10 channels to the required microsecond resolution is close to impossible. This would result in the receiving and transmission of drastically wrong PWM widths, effectively transferring incorrect data.

The PIC operates as a slave device on the I^2C bus, emulating a set of registers. The Raspberry Pi can write desired motor PWM lengths to these registers and read received receiver PWM pulse lengths using standard SMBUS protocols. This is the same communication method all of the sensor ICs use, meaning only one protocol is required.

The output PWM to the motors is handled using the PIC's output compare modules in single shot mode. This module compares a stored period value against the incrementing counter value of a timer module, generating an interrupt once the two values match. In this application this is configured to set the output PWM pin high at the start of the PWM pulse, while simultaneously starting a timer. This timer increments at a defined division of the clock frequency, until the value in its counter register matches the value stored in the output compare modules period register. This event triggers an interrupt, which sets the pin low again and puts the OC module and timer to sleep until a new pulse is required. This means PWM pulses to the ESC's will only be generated while the PIC is being given new period values by the RPi, stopping the ESC's as soon as the RPi stops requesting new output periods, such as in the event of the flight controller crashing.

The measurement of the 6 PWM channels from the RC reciever is handled using the PIC's input capture modules. These operate in a similar manner to the output capture modules, generating interrupts on the event of an input change, while simultaneously capturing a timer value. These are configured to generate interrupts on both the rising and falling edges of the input PWM pulse. A timer is started on the rising edge interrupt, and its value is captured and stored on the falling edge. These values stored in the simulated register bank, ready for reading by the RPi.

3.8 MPU6050

The InvenSense MPU6050 contains three gyroscopes and three accelerometers, with complementary 16bit ADCs and control registers. It also contains a low power microprocessor referred to as a digital motion processor (DMP) that can be used to implement basic sensor fusion algorithms onboard the sensor, but this feature was completely disabled during the project due to the limitations of this processor and the poor quality of documentation describing these features.

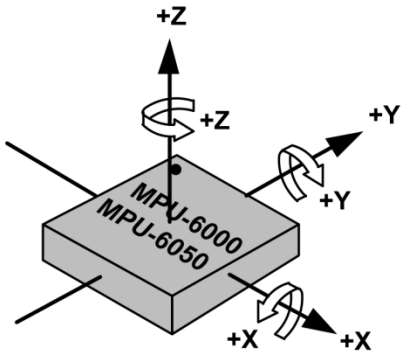

Figure 8 - MPU6050 (8)

3.8.1 Gyroscopes

The MPU6050's three gyroscopes each measure angular velocity around the three Euclidean axes using a vibrating micro electromechanical system (MEMS). This is comprised of a vibrating tuning fork element surrounded by capacitive sensors. Any rotation in the sensed axis will result in a deflection of the vibrating element due to the gyroscope effect, which translates into a changing

capacitance between this and the surface of the compartment. The degree of this change in capacitance is proportional to the angular velocity of the device, which is converted to a voltage, amplified, and digitized by a 16bit analogue to digital converter (ADC), before being stored in a set of registers accessible by the I^2C bus.

The output is very resistant to vibrational noise, but must be integrated over dt to produce an absolute angle estimate. This introduces an unknown varying error which causes a rapid drift in the angle estimate. Both this raw angular rate output and the result of the integration are demonstrated in Figure 9 for the pitch axis, featuring a positive pitch of 15° followed by a negative pitch of 15°.

The three measured axes are generally either referred to as $\omega = [\omega_x \quad \omega_y \quad \omega_z]^T$ or $\omega = [p \quad q \quad r]^T$, signifying rotations around the x, y, and z axes.

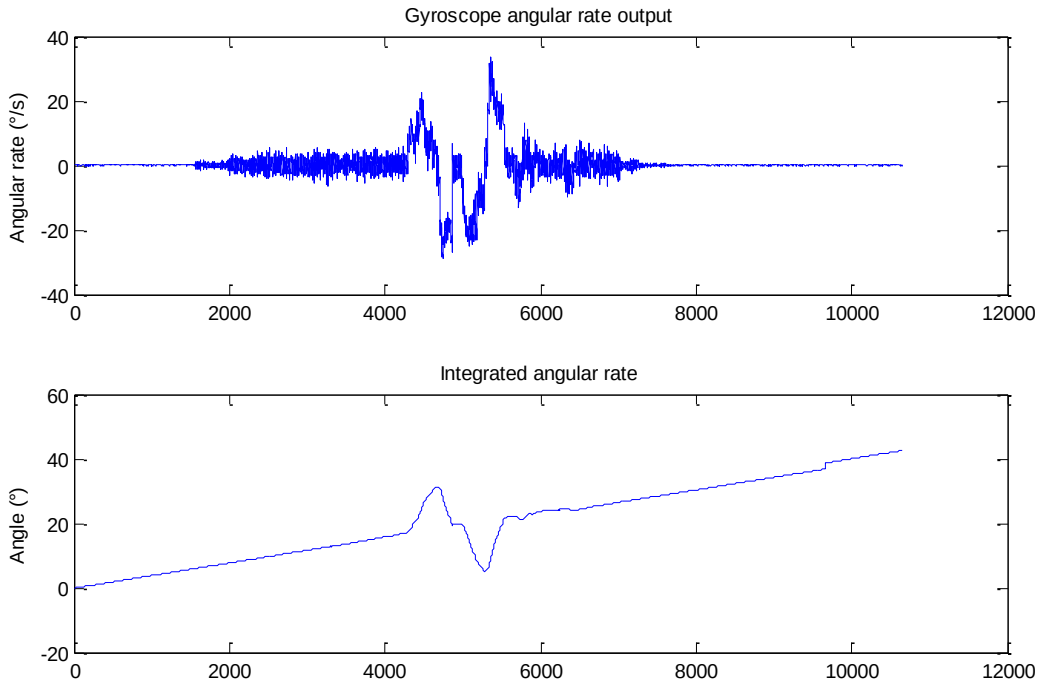

Figure 9 - Gyroscope raw angular rate vs. integrated angular rate for the pitch axis

3.8.2 Accelerometers

The MPU6050's triaxis accelerometer measures instantaneous acceleration along the three Euclidian axes. This is also achieved using a MEM structure, this time a mass suspended between two plates. An acceleration in the measured direction will result in a deflection of this mass, again resulting in a change in capacitance proportional to acceleration.

This triaxial acceleration measurement allows the direction of the gravity vector $[0 \quad 0 \quad g]^T$ to be calculated to provide an absolute attitude estimate relative to this vector. This is done using basic trigonometry, calculating pitch and roll relative to the horizon.

$$pitch = atan2(y, \sqrt{x^2 + z^2})$$

$$roll = atan2(x, \sqrt{y^2 + z^2})$$

The main weakness of this estimate is the addition of vibrational noise from the rest of the aircraft to the accelerometer outputs, which is indistinguishable from accelerations due to gravity, shown in Figure 10. This makes this estimate absolute yet very noisy. It is also impossible to determine the orientation around the gravity vector, which in this case means a yaw measurement cannot be taken.

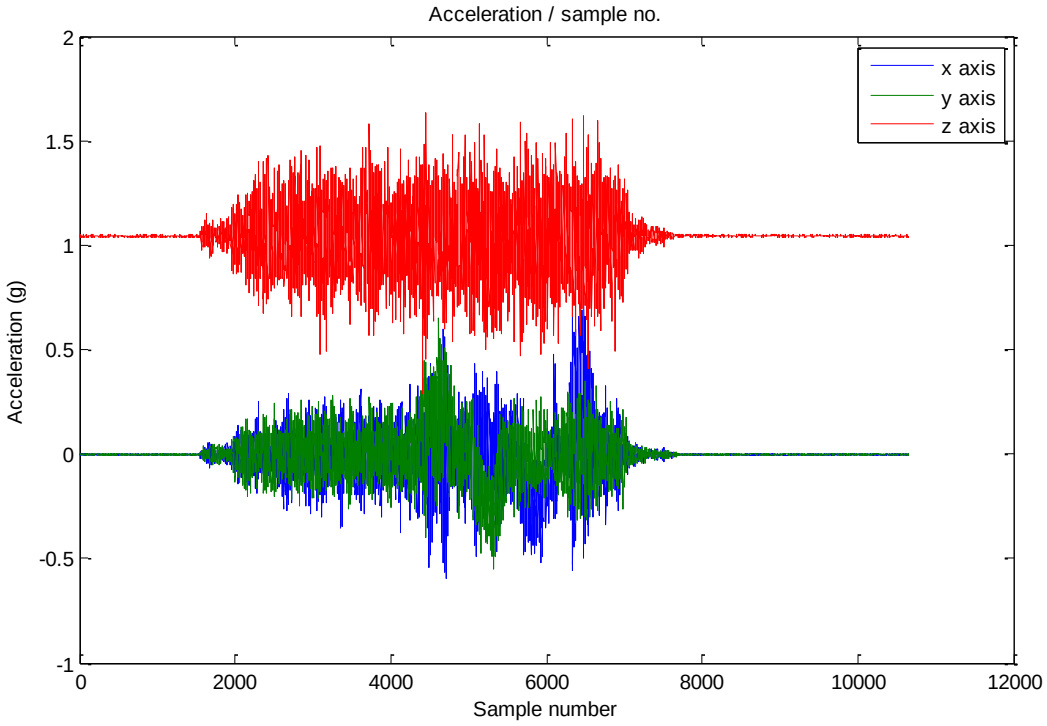

Figure 10 - Raw accelerometer output with motors active

3.9 HMC5883L Magnetometer

This is also a triaxial component, measuring magnetic field strength along the three Euclidian axes. This is achieved using three strips of magnetoresistive material, a type of material whose resistivity varies with directional magnetic field strength. Just like the accelerometer, these three directional measurements can be compared against the known vector of the earth's magnetic field, allowing an orientation relative to this to be calculated. The direction of the earth's field is generally defined as an inclination and a declination. The inclination, or magnetic dip, is the vertical angle between the field and the horizon, around -68° for the UK. The declination represents the horizontal angle between true north and the magnetic north, around -1.8° in the UK.

In reality this output is limited to only affect the yaw axis, as this is the one area where the accelerometer proves insufficient. This vector also suffers from two distinct types of distortion, soft iron and hard iron.

Hard iron distortions simply refer to a constant addition to the ambient magnetic field, usually due to magnetized material in the vehicle. This is analogous to the bias error/offset in the accelerometer, and is easily removed by subtraction from the measured field.

Soft iron interference is the distortion of the ambient field by soft iron metals in the vehicle, resulting in a varying magnitude measurement as the vehicle rotates. This is analogous to the accelerometer scale errors, and is compensated for in the same manner. The compensation for these effects is described in greater detail in section 4.3.

3.10 MS5611-01BA Barometer

The MS5611-01BA measures both temperature and ambient air pressure to 24bit resolution, which can be used to produce an altitude estimate accurate to 10cm (9). Although this sensor was used to provide an altitude hold mode using an additional PID loop, due to it not being integrated into the Kalman filter state estimation it won't be talked about anymore in this paper.

4 Sensor Calibration

4.1 Gyroscope temperature calibration

The gyroscope bias is very sensitive to temperature changes. Although this can be accounted for through bias estimation algorithms, it is best to attempt to remove as much of the offset as possible before processing. This was achieved by cooling the board to -15°C before allowing it to return to an equilibrium temperature under its own waste heat while recording the angular rate output from the gyroscope and the temperature from the built in thermistor. A 4th degree polynomial was then created using the Matlab plot fitting toolbox, producing equations describing the gyroscope bias as a function of temperature over the -15°C to 25°C temperature range shown in Figure 11, Figure 12, and Figure 13. This bias is then subtracted from every reading before any further processing is attempted.

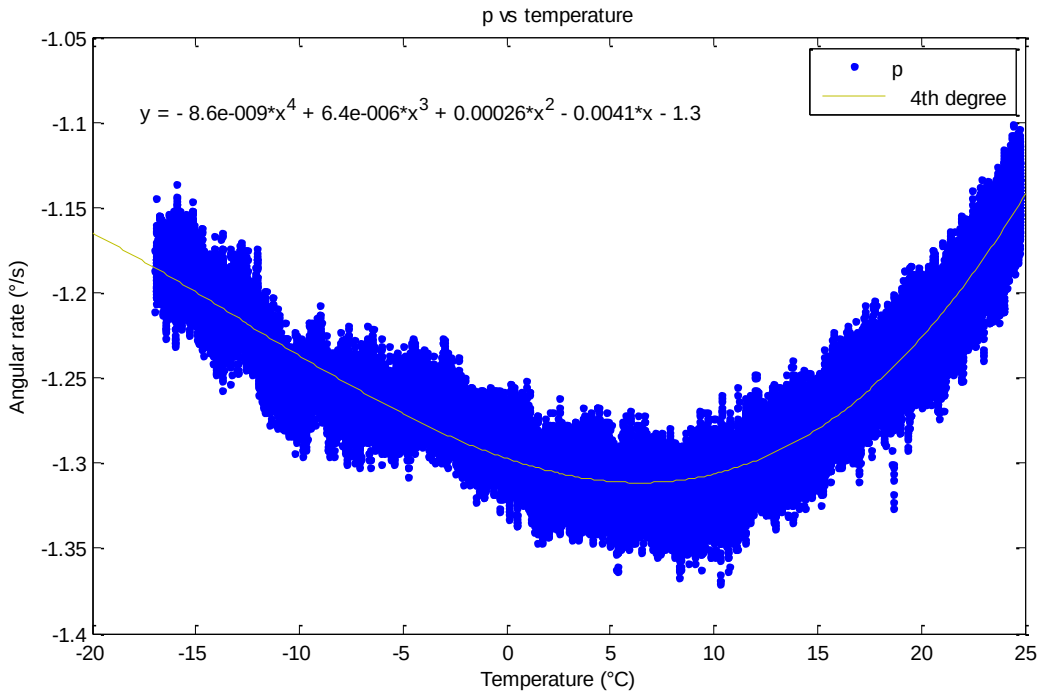

Figure 11 - p gyroscope temperature calibration

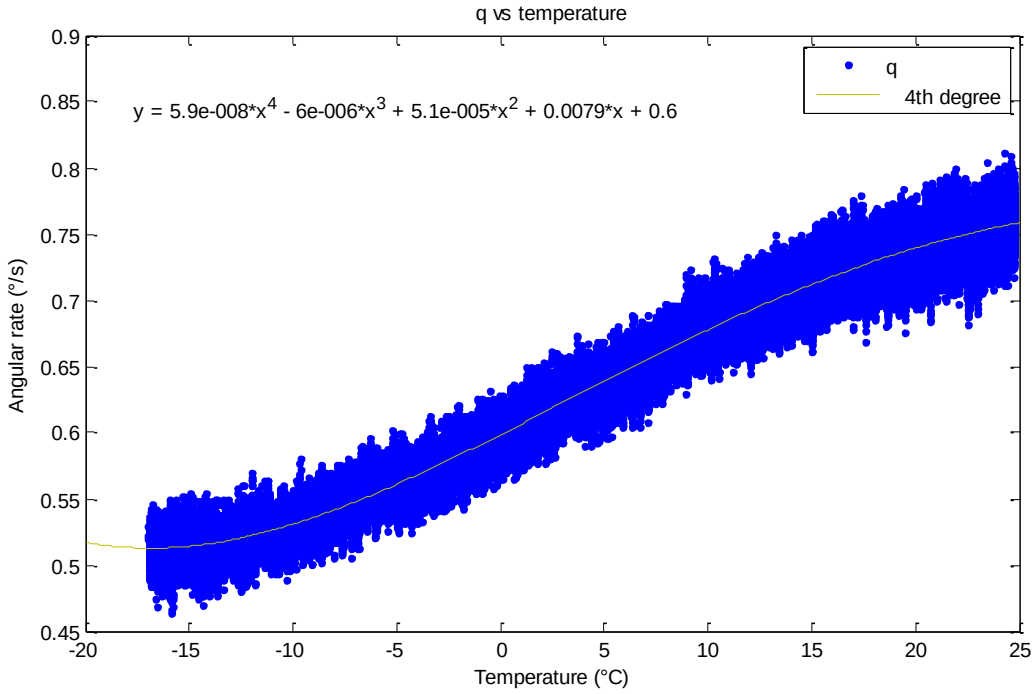

Figure 12 - q gyroscope temperature calibration

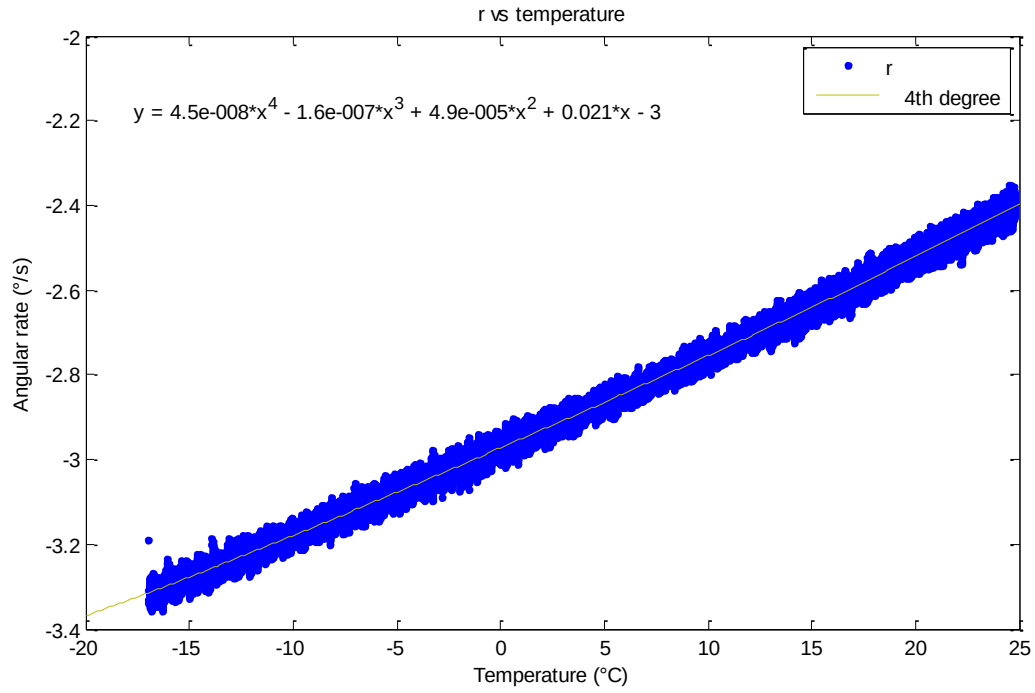

Figure 13 - r gyroscope temperature calibration

4.2 Accelerometer scale and bias calibration

The accelerometer suffers from scale and bias errors. The MPU6050 datasheet defines the scale (LSB/m/s^2) to an accuracy of ±3%, with a 0g calibration tolerance of ±50mg for the X and Y axes and

±80mg for the Z axis (8). This can introduce a significant error to the acceleration vector, meaning these values must be calibrated.

This is achieved using the knowledge that the gravity vector is of a constant magnitude when measured over every possible orientation. This means through any sequence of rotations the norm of the acceleration vector should evaluate to g. However, plotting the norm over a sequence of 6 orthogonal orientations shows a substantial change in the measured norm. This is solved using the open source Matlab script MgnCalibration.m (see section 10.1). Although designed to calibrate magnetometer triads, it can also be used with an acceleration vector. Given an input triad this function produces a 3X3 triangular matrix describing an ellipse representing the scale distortion and a vector describing the ellipse's center. These can be used as inputs to the script ApplyMgnCalibration.m (see section 10.2) to correct the accelerometer output vector, ensuring a constant magnitude across all orientations. The initial varying norm and the calibrated norm are shown in Figure 14 and Figure 15.

Unfortunately, both the scale and bias values are affected by temperature, meaning ideally this calibration needs to be performed many times over a large temperature range, allowing the scale and bias errors to be described as functions of temperature. However, in reality this is rather difficult to achieve without a temperature controlled chamber, so this was not performed for this project.

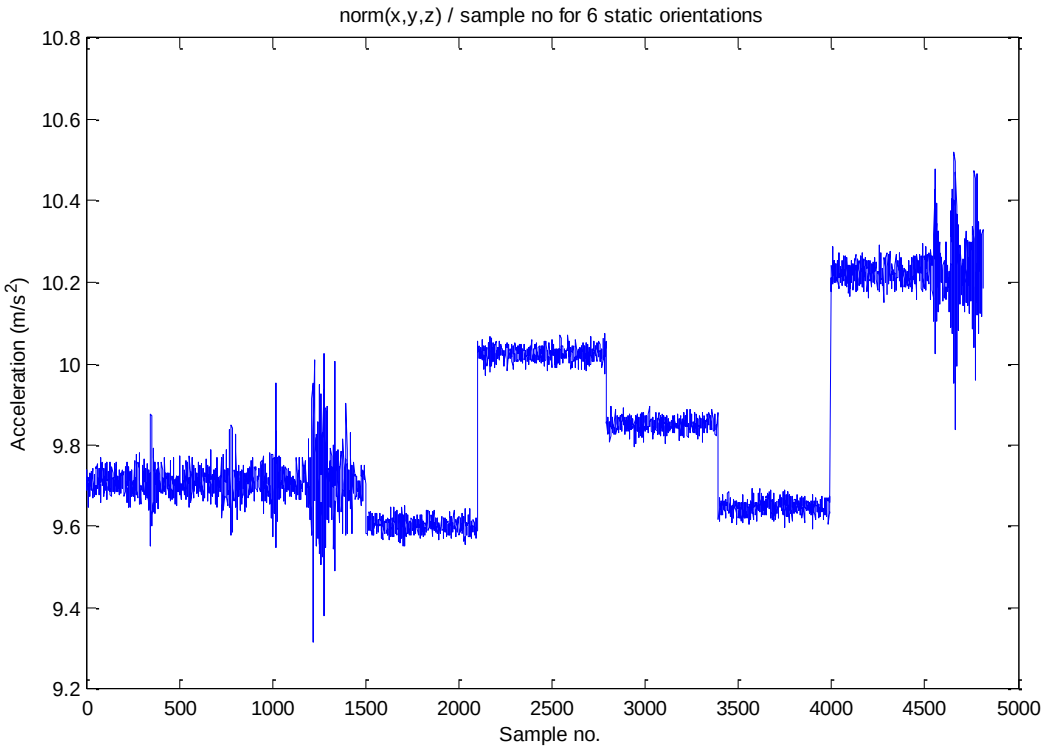

Figure 14 - norm(x,y,z) uncalibrated

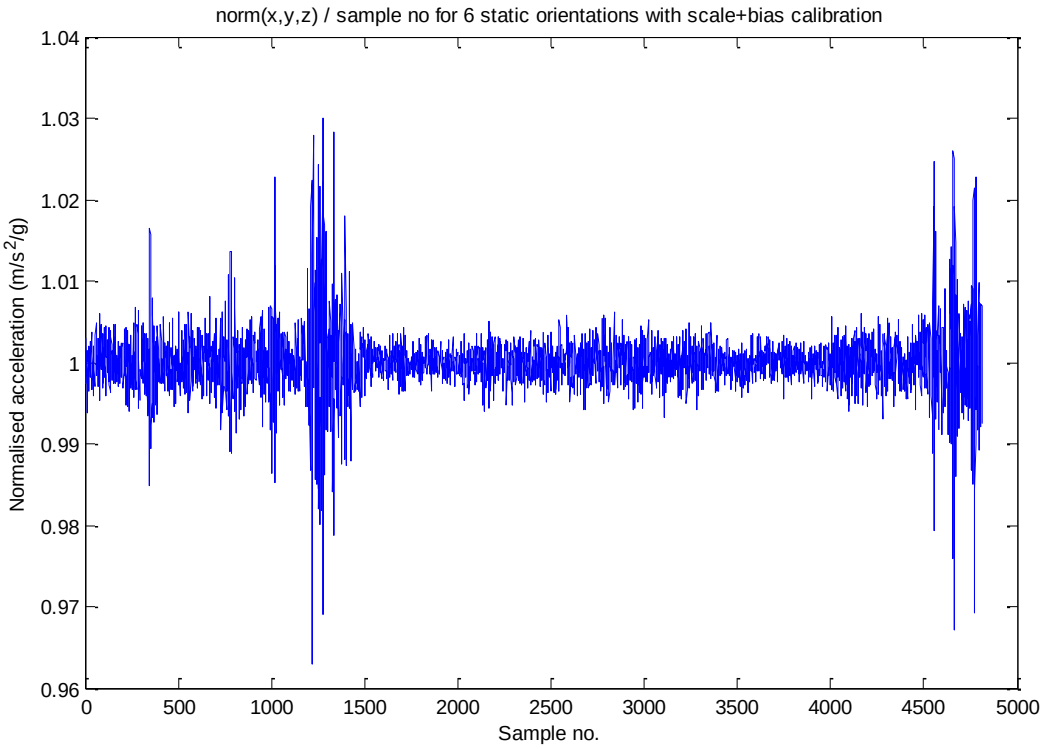

Figure 15 - norm(x,y,z) calibrated

4.3 Magnetometer hard and soft iron compensation

The magnetometer also suffers from scale and bias errors, but these are largely worsened by soft and hard iron distortions, described in section HMC5883L Magnetometer3.9. These errors are compensated in the same way as the accelerometer, as the earth's magnetic field vector can be assumed to be constant when measured from any orientation in the same location. Moving the platform through a large range of orientations whilst logging the measured magnetic field vector produces the plot shown in Figure 16. Ideally this would show a perfect sphere of radius B and center (0,0,0), where B represents the magnitude of the earth's magnetic field. Although hard to see, this sphere shows a significant negative offset in the z axis and scale errors in every axis. This data is run through the MgnCalibration.m Matlab function in the same way as the accelerometer, creating the desired perfect sphere.

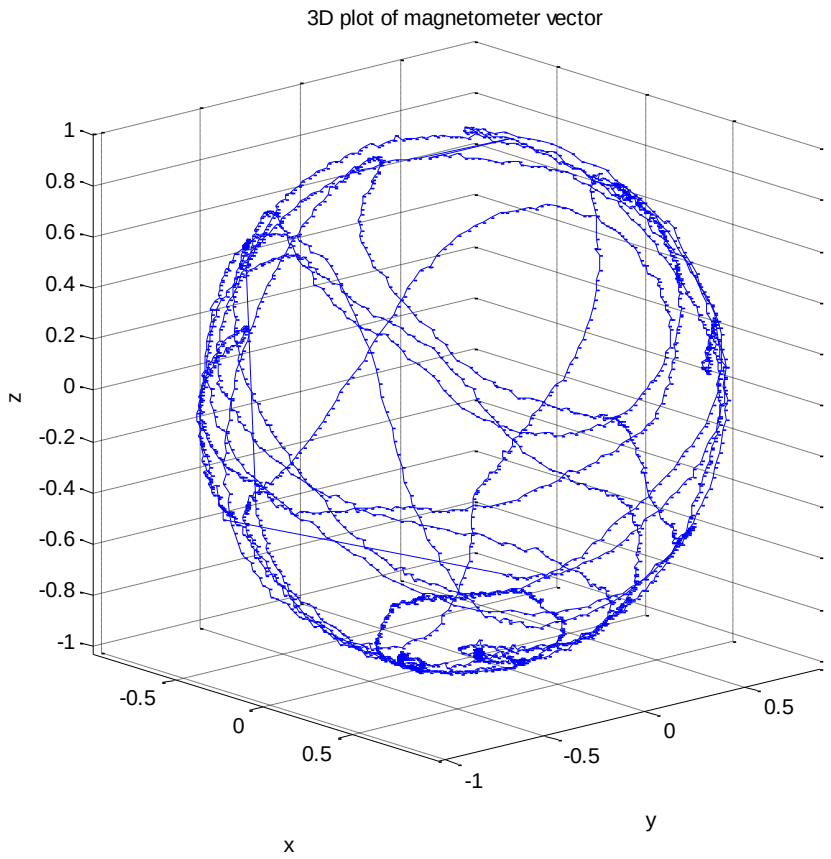

Figure 16 - 3D plot of magnetometer vector

5 Orientation Conventions

There exist many methods of describing the orientation of an object with respect to a reference frame. This paper considers the use of Euler angles and quaternions relative to the horizontal and the magnetic north. Another commonly used system is the direction cosine matrix (DCM), but this isn't considered in this paper due to its inferiority to quaternions.

5.1 Euler Angles

Euler angle representations use three separate single axis rotations to represent a complete 3D rotation. These three rotations can be organised into 12 different conventions, 6 known as Tait-Bryan angles, 6 known as proper Euler angles. This paper only considers the Z-X'-Z" Tait-Bryan convention, as this represents a rotation around the yaw axis, a rotation around the new pitch axis, and a rotation around the new roll axis, as shown in Figure 17.

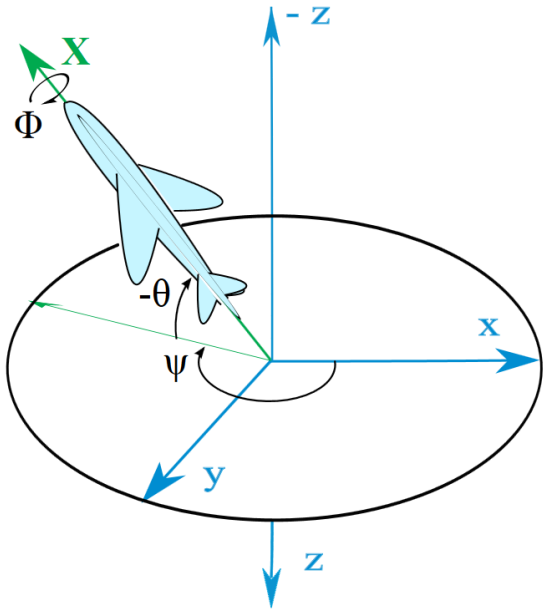

Figure 17 - Tait-Bryan angles. Ψθϕ order, yaw pitch roll. (10)

This method of representing orientation suffers from a significant number of disadvantages.

The first of these is simply the potential for a large amount of confusion due to the 12 different possible rotation sequences. For example, an aircraft that has pitched 90° then rolled 90° ends up in a very different position to an aircraft that rolls 90° then pitches 90°.

Secondly, a number of singularities exist in Euler angle representations. In the case of the order system used above, these occur at a yaw of ±2π, a pitch of ±0.5π, and a roll of ±2π. These sudden changes can play havoc on sensor fusion algorithms, requiring an extensive and messy use of conditional statements to attempt to correct, which gets increasingly difficult to implement with increasing filter complexity.

Finally, Euler angles of any order suffer from a concept known as gimbal lock. This occurs when the axes of two of the three gimbals in the system align in a parallel combination. This effectively prevents the system from moving in one extrinsic axis, making the system act like a two dimensional gimbal. This can be visualised in Figure 17 by imagining the aircraft rotating through the yaw axis by 0 rad, the pitch axis by 0.5π rad, and the roll axis by 0 rad. The yaw and roll axes now both lie in the xy plane, meaning the aircraft can no longer rotate in the yz plane. This can cause a lot of strange effects in orientation systems, resulting in the designer generally just trying to avoid approaching the point where these issues start to appear in the first place (11).

5.2 Quaternions

A quaternion is an extension to the complex numbers, commonly defined in two forms:

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$q = [q_0 \quad q_1 \quad q_2 \quad q_3]^T$$

When having a unity magnitude the quaternion can be used to represent a rotation. Unit quaternions used to represent rotations are known as rotation quaternions, and unit quaternions used to represent orientation relative to a fixed reference are known as attitude quaternions, both of which are used in this paper. They are continuous over any rotation, meaning they do not suffer from gimbal lock or the singularities that plague Euler angles. This makes manipulating an orientation mathematically simpler and infinitely more robust.

5.2.1 Norm

The norm of a quaternion is defined as

$$|q| = \sqrt{{q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2}$$

This must evaluate to unity for the quaternion to represent a rotation, which applies to all the quaternions used in this paper. This is achieved by normalising the quaternion after every operation using

$$q = \frac{q}{|q|}$$

5.2.2 Multiplication

A quaternion multiplication $p \otimes q$ represents the sequential combination of the two rotations (12 p. 14). This is a non-commutative operation defined as

$$p \otimes q = Q(p)q = \bar{Q}(q)p$$

where the quaternion matrix function is defined as

$$Q(q) = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix}$$

and the conjugate quaternion matrix is defined as

$$\bar{Q}(q) = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix}$$

5.2.3 Conjugate

The quaternion conjugate is defined in the same manner as the complex conjugate, the sign of the complex part is inverted

$$Conj(q) = q^* = [q_0 \quad -q_1 \quad -q_2 \quad -q_3]^T$$

5.2.4 Vector rotation

A vector $\boldsymbol{x}$ can be rotated from the reference frame to the body frame represented by the unit quaternion q to produce the vector $\boldsymbol{x}'$ using

$$\begin{bmatrix} 0 \\ \boldsymbol{x}' \end{bmatrix} = q \otimes \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \otimes q^*$$

These two quaternion multiplications can be combined to form a rotation matrix $\boldsymbol{R}(q)$ so that $\boldsymbol{x}' = \boldsymbol{R}(q)\boldsymbol{x}$ (13 p. 3)

$$\boldsymbol{R} = \begin{bmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{bmatrix}$$

Likewise, a vector $\boldsymbol{x}$ can be rotated from the body frame represented by the unit quaternion q to the reference frame to produce the vector $\boldsymbol{x}'$ using

$$\begin{bmatrix} 0 \\ \boldsymbol{x}' \end{bmatrix} = q^* \otimes \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \otimes q$$

This can also be simplified to form a rotation matrix $\boldsymbol{R}^*(q)$ so that $\boldsymbol{x}' = \boldsymbol{R}^*(q)\boldsymbol{x}$

$$\boldsymbol{R}^* = \begin{bmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{bmatrix}$$

5.2.5 Derivative

The derivative of a quaternion can be defined by two equations, one if the angular rate vector is in the fixed frame of reference and one if in the body frame (12 p. 16). This paper only considers angular rates in the body frame, with the perturbed quaternion defined as

$$\dot{q}_\omega(q, \omega) = \frac{1}{2}\begin{bmatrix} 0 \\ \omega \end{bmatrix} \otimes q = \frac{1}{2}\bar{Q}(q)\begin{bmatrix} 0 \\ \omega \end{bmatrix} = \frac{1}{2}\begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

5.2.6 Quaternions to Euler angles

An attitude quaternion can be converted into Euler angles in the Tait-Bryan YPR (yaw pitch roll) order using

$$yaw = atan2(2 \cdot (q_0q_3 + q_1q_2), 1 - 2 \cdot ({q_2}^2 + {q_3}^2))$$

$$pitch = atan2(2 \cdot (q_0q_1 + q_2q_3), 1 - 2 \cdot ({q_1}^2 + {q_2}^2))$$

$$roll = \text{asin}(2(q_0q_2 - q_3q_1))$$

6 Sensor Fusion

Since these sensors cannot be used independently for attitude estimation they must be combined using a concept known as sensor fusion, designed to play one sensor's weaknesses against another sensor's strengths. The ideal algorithm will keep the low noise shape of the integrated gyroscope output, but without any of the integration drift this plot usually exhibits.

6.1 Test Data

The same set of test data was used to compare each sensor fusion method. The data set consists of a positive pitch of 15° followed by a negative pitch of 15°, with the same repeated in the roll axis. This was performed with all four motors running at hovering speeds in order to best simulate the noise levels encountered during a flight. The plotted results only compare the filters results on the pitch axis estimate, with the integrated angular rate about the pitch axis and the accelerometer derived pitch estimate plotted for reference. It is important to note that the small step increase on the gyroscope estimate plot around sample 9500 is an error caused by the flight controller pushing the log data from a memory buffer to the disk, resulting in a large dt measurement. This only occurs when the motors are inactive, so the flight performance is unaffected.

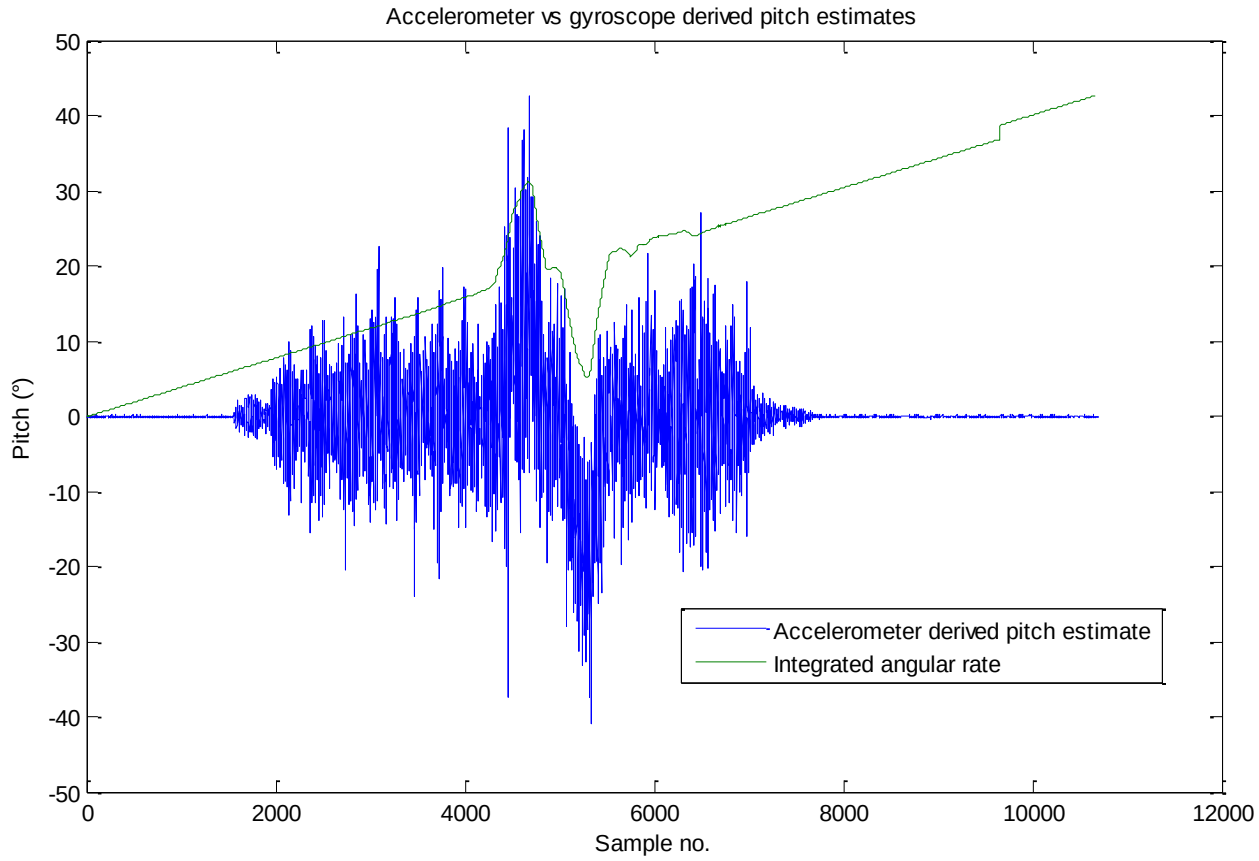

Figure 18 - Accelerometer vs. gyroscope derived angle estimates

6.2 Complementary Filter

The most basic commonly used form of sensor fusion is the complementary filter. This can be used to separately combine the pitch and roll estimates from the accelerometer with the x and y gyroscope outputs, and the z gyro output with the magnetometer yaw estimate

$$\theta_k = (\theta_{k-1} + \dot{\theta} * dt) * a + \theta_{acc} * (1 - a)$$

This equation operates in the frequency domain to essentially sum a low pass filtered accelerometer angle estimate with a high pass filtered gyroscope estimate, with both filters holding equal corner frequencies. This can also be viewed as a basic precursor to the Kalman filter or Wiener filter, as a Kalman filter similarly uses a weighted average to sum two outputs, just with additional statistical analysis in the time domain to calculate the required weightings on the fly (14).

Running the test data through this filter produces the estimate shown in Figure 19. This estimate exhibits the same shape as the integrated angular rate, but without the long term drift. There is however a constant offset proportional to gyroscope bias error, which can only be lessened by reducing a, introducing more noise from the accelerometers. Clearly this bias error needs to be calculated as the filter progresses, allowing it to be subtracted from each gyroscope reading, achieved using a concept known as Kalman filtering.

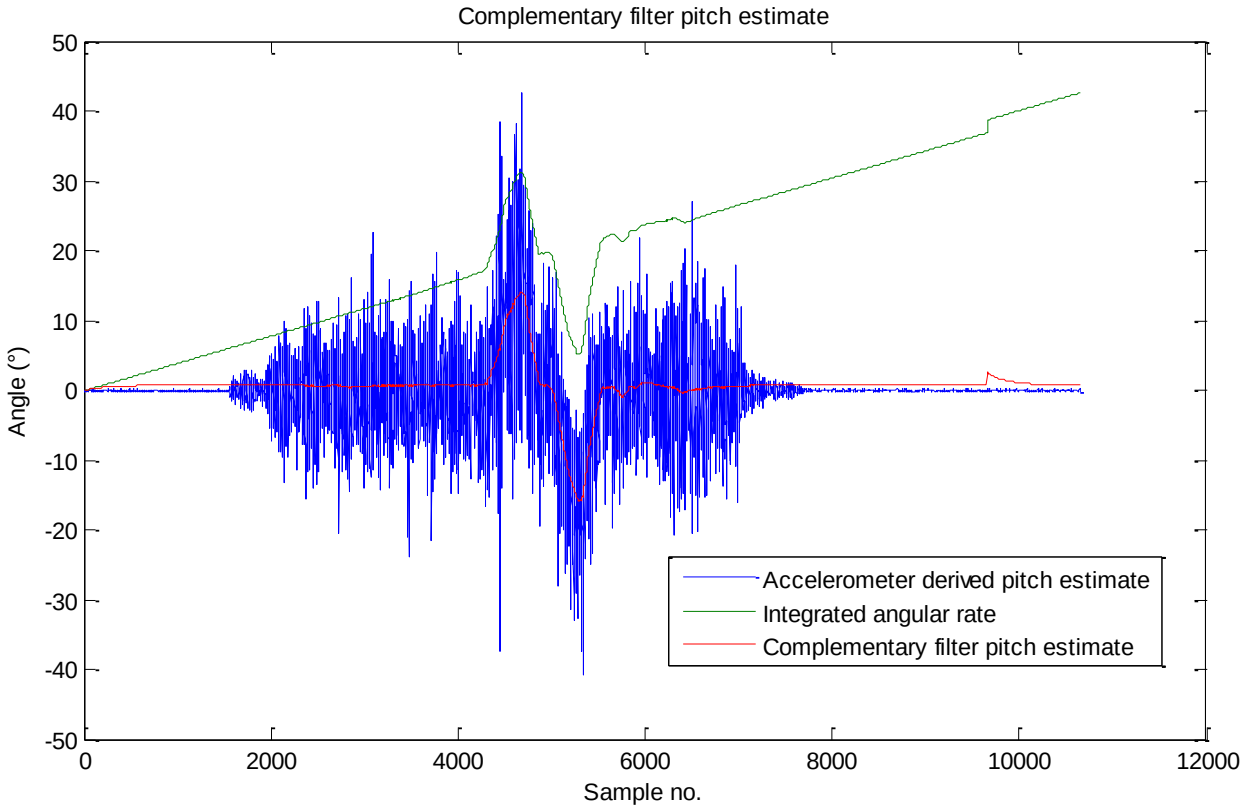

Figure 19 - Complementary filter pitch estimate

6.3 The Kalman Filter

The Kalman filter is a linear recursive minimum mean squared error estimator that produces a statistically optimal estimate of an underlying system state represented by a stream of noisy data. It is the go-to method for fusing sensor data in AHRS applications, as it can be designed to estimate both the orientation as well as other variables such as sensor biases.

The Kalman filter can be split into two steps. The prediction step produces an estimate of the current state variables, as well as their uncertainties. This can be performed many successive times, with the prediction becoming less certain with each iteration. The update step then compares these estimates against a noisy measurement, updating the estimate using a weighted combination of the

prediction and the measurement values, with more weight being given to the estimate with the higher certainty. Finally the covariance of this estimate can be updated, representing the accuracy of the new state estimation following the comparison to the measured data.

These two steps can be described analogously by imagining a person in a known location in a dark room. The prediction step involves the person taking a step forward, estimating their new position/state based on the length of their stride. With every step the certainty of their position decreases due to the compounding uncertainties in the length of every stride. The update step can then be represented by the person reaching a known wall in the room, representing a measurement, allowing them to correct their previous position prediction and increase the certainty of their position estimate. Successive measurements can be taken to further reduce the uncertainty of their position, with the covariances of each method being updated depending on how much the prediction and measurement results agree.

This loop can be repeated recursively with every time step, requiring no history of the variables used. The produced estimate can be described as the optimal solution, but only when the underlying system can be assumed to be completely linear and only affected by process and measurement noise sources with Gaussian distributions. This requirement is represented mathematically as the prediction model and the measurement model (15 p. 219).

$$x_k = Fx_{k-1} + Bu_k + N(0, Q_k)$$

$$z_k = Hx + N(0, R_k)$$

Where

F is the transition model matrix that maps the previous state x_{k-1} to the next state x_k.

B is the control transition model matrix that maps the control vector u_k onto the next state x_k.

H is the observation model matrix that maps the predicted state x_k onto the measurement vector z_k.

$N(0, Q_k)$ and $N(0, R_k)$ represent Gaussian noise sources of variances $Q_k$and R_k and mean 0.

6.3.1 State Vector Definition

This filter is going to be used to estimate the pitch and roll of the AHRS, along with the biases of the two gyroscopes that represent these axes (for small deflections from the horizontal). Therefore the state vector to be estimated can be defined as

$$x = \begin{bmatrix} pitch \\ \text{roll} \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_k$$

6.3.2 AHRS Prediction and Measurement Models

In this application the prediction and measurement models are defined as follows

$$\begin{bmatrix} pitch \\ \text{roll} \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & -dt & 0 \\ 0 & 1 & 0 & -dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_k . \begin{bmatrix} pitch \\ roll \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_{k-1} + \begin{bmatrix} dt & 0 & 0 & 0 \\ 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_k . \begin{bmatrix} \omega_x \\ \omega_y \\ 0 \\ 0 \end{bmatrix}_k + N(0, Q_k)$$

$$\begin{bmatrix} A_{pitch} \\ A_{roll} \\ 0 \\ 0 \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_k . \begin{bmatrix} pitch \\ \text{roll} \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_k + N(0, R_k)$$

These can be expanded to

$$\begin{aligned} pitch_k &= pitch_{k-1} + dt(\omega_x - (\omega_{xb})_{k-1}) \\ roll_k &= roll_{k-1} + dt(\omega_y - (\omega_{yb})_{k-1}) \\ (\omega_{xb})_k &= (\omega_{xb})_{k-1} \\ (\omega_{yb})_k &= (\omega_{yb})_{k-1} \end{aligned}$$

Where ω_x and ω_{xb} represent the x axis gyroscope angular velocity and its respective bias.

6.3.3 Covariance matrix derivation

The filter is tuned to the system it is estimating using the two covariance matrices, Q_k and R_k. In these application they are defined as

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \qquad R = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

6.3.4 Kalman Filter Equations

Once these two models are defined it is possible to propagate the calculated matrix values through the Kalman filter equations to produce an estimate.

Predict

Predicted state estimate $\boldsymbol{x}_{k|k-1} = F\boldsymbol{x}_{k-1|k-1} + B\boldsymbol{u}$

Predicted estimate covariance $P_{k|k-1} = FP_{k-1|k-1}F^T + Q$

Update

Measurement residual $\boldsymbol{y} = \boldsymbol{z} - H\boldsymbol{x}_{k|k-1}$

Residual covariance $S = HP_{k|k-1}H' + R$

Optimal Kalman gain $K = P_{k|k-1}H^T S^{-1}$

Updated state estimate $\boldsymbol{x}_{k|k} = \boldsymbol{x}_{k|k-1} + K\boldsymbol{y}$

Updated estimate covariance $P_{k|k} = (I - KH)P_{k|k-1}$

6.3.5 Kalman Filter Results

Propagating the test data from section 6.1 through this filter produces the estimate shown in Figure 20. This gives an angle estimate with the same noiseless shape as the integrated gyroscope angular rate, but with no offset from the accelerometer estimate as was evident in the complementary filter. It is also important to note that the Kalman filter has nearly completely filtered out the gyroscope step error at sample 9650 due to the large uncertainty of that iterations estimate, caused by the large difference between the accelerometer and gyroscope based estimates.

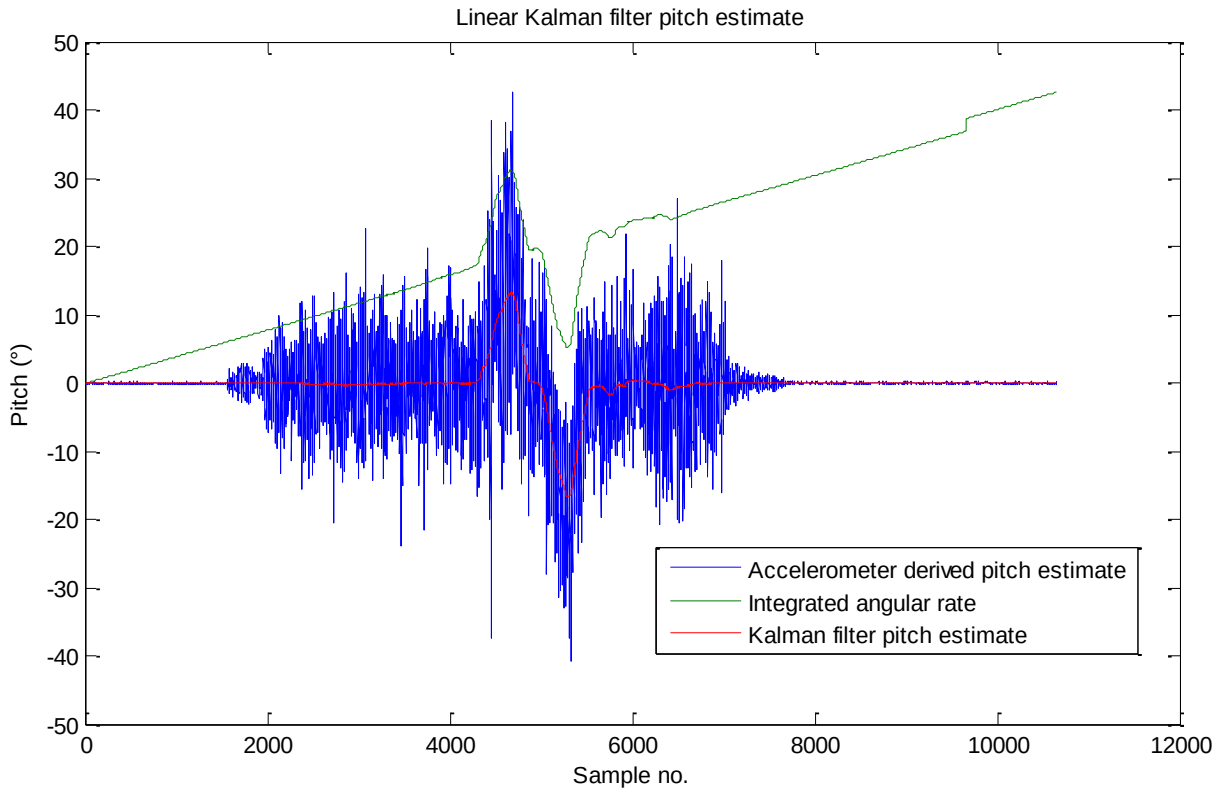

Figure 20 - Linear Kalman filter output

6.3.6 Disadvantages of the Standard Kalman Filter

This Kalman filter implementation has a number of disadvantages. First, when a rotation through an axis is attempted when another axis is already deflected from the horizontal, the trigonometric angle estimate from the accelerometer and the angular rate measurement from the corresponding gyroscope will no longer lie in the same plane. This isn't an issue when operating very close to the horizontal, but becomes a large error source at larger deflections. The use of Euler angles in this filter also introduces all of their inherent problems, such as gimbal lock and singularities (see section 5.1).

Both of these issues can be solved by using a quaternion based representation of the AHRS's attitude, which requires the ability to use nonlinear prediction and measurement models. This is made possible using the extended Kalman filter.

6.4 The Extended Kalman Filter

The extended Kalman filter is the nonlinear version of the regular Kalman filter, which uses the Jacobian of the prediction and measurement functions to linearise about an estimate of the current state and covariance. This allows a whole new set of prediction and measurement models to be used for estimation, as long as the prediction and measurement models f and h can be defined as differentiable functions of $\boldsymbol{x}$ and $\boldsymbol{u}$ with zero mean multivariate Gaussian process and measurement noises with covariances Q_k and R_k.

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}) + N(0, Q_k)$$

$$\boldsymbol{z} = h(\boldsymbol{x}_k) + N(0, R_k)$$

6.4.1 State vector definition

The state vector represents the variables we wish to recursively estimate using the filter. For this application this translates to an attitude estimate quaternion and three gyroscope biases.

$$\boldsymbol{x} = [q_0 \quad q_1 \quad q_2 \quad q_3 \quad \omega_{xb} \quad \omega_{yb} \quad \omega_{zb}]^T$$

Since the quadrotor is expected to start in a level position $q = [1 \quad 0 \quad 0 \quad 0]$ and with unknown gyroscope biases, the state is initialised as

$$\boldsymbol{x} = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

The estimate covariance P is initialised with large initial variances, representing the complete lack of knowledge of the current orientation and gyroscope biases

$$P = I_{7x7} * 100000$$

6.4.2 Prediction model formulation

In this application of the EKF, the prediction model represents a function that describes the next attitude estimate in terms of the previous estimate and the gyroscope angular rate vector. The next quaternion is simply found through numerical integration (see section 5.2.5) using

$$q_k = q_{k-1} + dt * \dot{q}_k$$

where

$$\dot{q}_\omega(q, \omega) = \frac{1}{2}\begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & q_3 & -q_2 \\ -q_3 & q_0 & q_1 \\ q_2 & -q_1 & q_0 \end{bmatrix}\begin{bmatrix} \omega_x - \omega_{xb} \\ \omega_y - \omega_{yb} \\ \omega_z - \omega_{zb} \end{bmatrix}$$

The gyroscope biases are unchanged during the prediction step. This allows f to be built as

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, u_k) = \begin{bmatrix} q_0 + \frac{dt}{2} * (-q_1(\omega_x - \omega_{xb}) - q_2(\omega_y - \omega_{yb}) - q_3(\omega_z - \omega_{zb}) \\ q_1 + \frac{dt}{2} * (q_0(\omega_x - \omega_{xb}) + q_3(\omega_y - \omega_{yb}) - q_2(\omega_z - \omega_{zb}) \\ q_2 + \frac{dt}{2} * (-q_3(\omega_x - \omega_{xb}) + q_0(\omega_y - \omega_{yb}) + q_1(\omega_z - \omega_{zb}) \\ q_3 + \frac{dt}{2} * (q_2(\omega_x - \omega_{xb}) - q_1(\omega_y - \omega_{yb}) + q_0(\omega_z - \omega_{zb}) \\ \omega_{xb} \\ \omega_{yb} \\ \omega_{zb} \end{bmatrix}$$

The Jacobian can then be calculated to produce matrix F

$$\frac{\partial f}{\partial \boldsymbol{x}} = F = \begin{bmatrix} 1 & -\frac{dt}{2}(\omega_x - \omega_{xb}) & -\frac{dt}{2}(\omega_y - \omega_{yb}) & -\frac{dt}{2}(\omega_z - \omega_{zb}) & \frac{dt}{2}q_1 & \frac{dt}{2}q_2 & \frac{dt}{2}q_3 \\ \frac{dt}{2}(\omega_x - \omega_{xb}) & 1 & -\frac{dt}{2}(\omega_z - \omega_{zb}) & \frac{dt}{2}(\omega_y - \omega_{yb}) & -\frac{dt}{2}q_0 & -\frac{dt}{2}q_3 & \frac{dt}{2}q_2 \\ \frac{dt}{2}(\omega_y - \omega_{yb}) & \frac{dt}{2}(\omega_z - \omega_{zb}) & 1 & -\frac{dt}{2}(\omega_x - \omega_{xb}) & \frac{dt}{2}q_3 & -\frac{dt}{2}q_0 & -\frac{dt}{2}q_1 \\ \frac{dt}{2}(\omega_z - \omega_{zb}) & -\frac{dt}{2}(\omega_y - \omega_{yb}) & \frac{dt}{2}(\omega_x - \omega_{xb}) & 1 & -\frac{dt}{2}q_2 & \frac{dt}{2}q_1 & -\frac{dt}{2}q_0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

6.4.3 Measurement model formulation

The measurement model $h(\boldsymbol{x}_k)$ serves to map the current state estimate $\boldsymbol{x}_k$ onto the measurement vector $\boldsymbol{z}$, effectively allowing the prediction to be compared with the real world measurements. In this case this involves describing what both the accelerometer and magnetometer should be reading given the current quaternion prediction, so the measurement vector can be defined as

$$\boldsymbol{z} = [a_x \quad a_y \quad a_z \quad m_x \quad m_y \quad m_z]^T$$

6.4.3.1 Accelerometer mapping

The fixed frame gravity vector is rotated into the body frame, represented by the unit quaternion q, mapping it into the accelerometer vector. Since this application only cares about the direction of the gravity vector and not its magnitude, the vector is normalised before being operated on.

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = h(\boldsymbol{x}_k) = R(q) * \vec{g} = R(q) * \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -2(q_1 q_3 - q_0 q_2) \\ -2(q_2 q_3 + q_0 q_1) \\ -{q_0}^2 + {q_1}^2 + {q_2}^2 - {q_3}^2 \end{bmatrix}$$

6.4.3.2 Magnetometer mapping

The magnetometer mapping is performed in the same way as with the accelerometer, rotating the reference frame magnetic field strength vector b into the body frame represented by the estimate quaternion

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = h(\boldsymbol{x}_k) = R(\vec{q}) \cdot \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} b_x({q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2) + 2b_y(q_1 q_2 + q_0 q_3) + 2b_z(q_1 q_3 - q_0 q_2) \\ 2b_x(q_1 q_2 - q_0 q_3) + b_y({q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2) + 2b_z(q_2 q_3 + q_0 q_1) \\ 2b_x(q_1 q_3 + q_0 q_2) + 2b_y(q_2 q_3 - q_0 q_1) + b_z({q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2) \end{bmatrix}$$

Unfortunately, using this model allows the magnetometer reading to influence the pitch and roll estimates, meaning a change in the reference field due to nearby power lines or large magnetic objects can have a direct effect on the pitch and roll estimates, potentially leading to a large

attitude error. It also requires the direction of the earth's magnetic field to be specified, which varies considerably across the earth. This was solved using a solution presented by S. Madgwick (16), where the reference field direction is calculated to have an identical inclination to the measured field.

This is done by first rotating the measured magnetic vector from the body frame to the reference frame to produce m'

$$\begin{bmatrix} m'_x \\ m'_y \\ m'_z \end{bmatrix} = R^*(\vec{q}) \cdot \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} m_x(q_0{}^2 + q_1{}^2 - q_2{}^2 - q_3{}^2) + 2m_y(q_1q_2 + q_0q_3) + 2m_z(q_1q_3 - q_0q_2) \\ 2m_x(q_1q_2 - q_0q_3) + m_y(q_0{}^2 - q_1{}^2 + q_2{}^2 - q_3{}^2) - 2m_z(q_2q_3 + q_0q_1) \\ 2m_x(q_1q_3 + q_0q_2) + 2m_y(q_2q_3 - q_0q_1) + m_z(q_0{}^2 - q_1{}^2 - q_2{}^2 + q_3{}^2) \end{bmatrix}$$

The new reference field is then calculated to share the same inclination as the measured vector, with a declination of 0 to simplify the calculation

$$b = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} \sqrt{m'_x{}^2 + m'_y{}^2} \\ 0 \\ m'_z \end{bmatrix}$$

These new reference values are then substituted into the original mapping, with the b_y term disappearing

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = h(\boldsymbol{x}_k) = R(\vec{q}) \cdot \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} b_x(q_0{}^2 + q_1{}^2 - q_2{}^2 - q_3{}^2) + 2b_z(q_1q_3 - q_0q_2) \\ 2b_x(q_1q_2 - q_0q_3) - 2b_z(q_2q_3 + q_0q_1) \\ 2b_x(q_1q_3 + q_0q_2) + b_z(q_0{}^2 - q_1{}^2 - q_2{}^2 + q_3{}^2) \end{bmatrix}$$

This limits the magnetometer to only influencing the yaw estimate, filling in the one area where the accelerometer is insufficient.

These two mappings can then be combined to form the full transition function h and its Jacobian H

$$h(\boldsymbol{x}_k) = \begin{bmatrix} -2(q_1q_3 - q_0q_2) \\ -2(q_2q_3 + q_0q_1) \\ -q_0{}^2 + q_1{}^2 + q_2{}^2 - q_3{}^2 \\ b_x(q_0{}^2 + q_1{}^2 - q_2{}^2 - q_3{}^2) + 2b_z(q_1q_3 - q_0q_2) \\ 2b_x(q_1q_2 - q_0q_3) + 2b_z(q_2q_3 + q_0q_1) \\ 2b_x(q_1q_3 + q_0q_2) + b_z(q_0{}^2 - q_1{}^2 - q_2{}^2 + q_3{}^2) \end{bmatrix}$$

$$H = \frac{\partial h}{\partial \boldsymbol{x}} = \begin{bmatrix} 2q_2 & -2q_3 & 2q_0 & -2q_1 & 0 & 0 & 0 \\ -2q_1 & -2q_0 & -2q_3 & -2q_2 & 0 & 0 & 0 \\ -2q_0 & 2q_1 & 2q_2 & -2q_3 & 0 & 0 & 0 \\ 2(q_0b_x - q_2b_z) & 2(q_1b_x + q_3b_z) & 2(-q_2b_x - q_0b_z) & 2(-q_3b_x + q_1b_z) & 0 & 0 & 0 \\ 2(-q_3b_x + q_1b_z) & 2(q_2b_x + q_0b_z) & 2(q_1b_x + q_3b_z) & 2(-q_0b_x + q_2b_z) & 0 & 0 & 0 \\ 2(q_2b_x + q_0b_z) & 2(q_3b_x - q_1b_z) & 2(q_0b_x - q_2b_z) & 2(q_1b_x + q_3b_z) & 0 & 0 & 0 \end{bmatrix}$$

6.4.4 Covariance matrices

The EKF is tuned through the specification of the Q and R covariance matrices. These are diagonal matrices representing the variance of the noise expected on the predicted state estimate and the

measurement vector ***z***. Ideally these would be calculated from test data, but in reality the variable non-Gaussian distribution of the noise present means they are simply modified until the desired estimate is produced. For this application there values were chosen as

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 \end{bmatrix} \quad R = \begin{bmatrix} 1e6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e7 \end{bmatrix}$$

6.4.5 EKF Equations

These models are then propagated through the extended Kalman filter equations, which appear very similar to those of the linear KF.

Predict

Predicted state estimate	$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, u)$
Predicted estimate covariance	$P = FPF' + Q$

Update

Measurement residual	$\boldsymbol{y} = \boldsymbol{z} - h(\boldsymbol{x}_k)$
Residual covariance	$S = HPH' + R$
Optimal Kalman gain	$K = PH'S^{-1}$
Updated state estimate	$\boldsymbol{x}_k = \boldsymbol{x}_k + Ky$
Updated estimate covariance	$P = (I - KH)P$

With $F = \frac{\partial f}{\partial \boldsymbol{x}}$ and $H = \frac{\partial h}{\partial \boldsymbol{x}}$. I represents the identity matrix.

The estimated quaternion can then be converted to the yaw pitch roll values required by the control loop using

$$\begin{bmatrix} pitch \\ roll \\ yaw \end{bmatrix}(q) = \begin{bmatrix} atan2(2(q_0q_1 + q_2q_3), 1 - 2({q_1}^2 + {q_2}^2)) \\ \arcsin(2(q_0q_2 - q_3q_1)) \\ atan2(2(q_0q_3 + q_1q_2), 1 - 2({q_2}^2 + {q_3}^2)) \end{bmatrix}$$

6.4.6 EKF Results

The EKF was implemented in the Matlab function EKF.m (see appendix 10.7), and the standard set of test data was propagated through the filter. This produces an estimate very similar to the standard Kalman filter, with a few important differences. First, this estimator is able to operate in any orientation, avoiding all of the issues introduced by Euler angles, such as the differing axis problem mentioned in section 6.3.6. This is most obvious around sample 6500 in Figure 21, where the regular Kalman filter implementation exhibits a small exponential as the incorrect prediction slowly converges back to the correct measured value. This same portion of the EKF estimate instead shows the pitch estimate immediately returning to 0° as the roll manoeuvre is completed.

The EKF is also able to produce a yaw estimate with respect to the magnetic north. This works very well when logging data without the motors active, as shown in Figure 22. However, once the motors powered up it was found that the amount of magnetic interference they generated excessively distorted the direction of the magnetic vector, to the point where it was nearly impossible to obtain a yaw estimate (see Figure 24).

Due to this and the fact that the yaw axis is normally operated in rate control (which doesn't require any form of EKF yaw estimation), the magnetometer was not integrated into the flight controller. This is represented by the Matlab function EKF2Simple.m (see appendix 10.5).

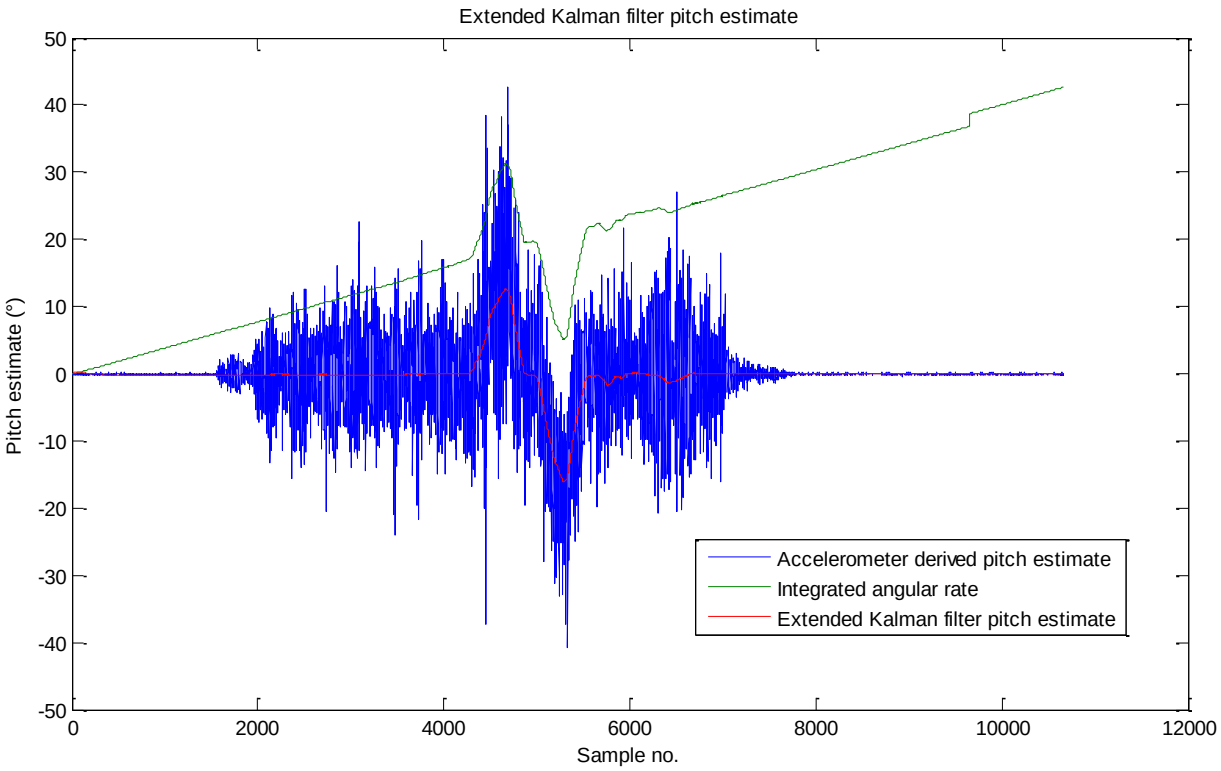

Figure 21 - Extended Kalman filter pitch estimate

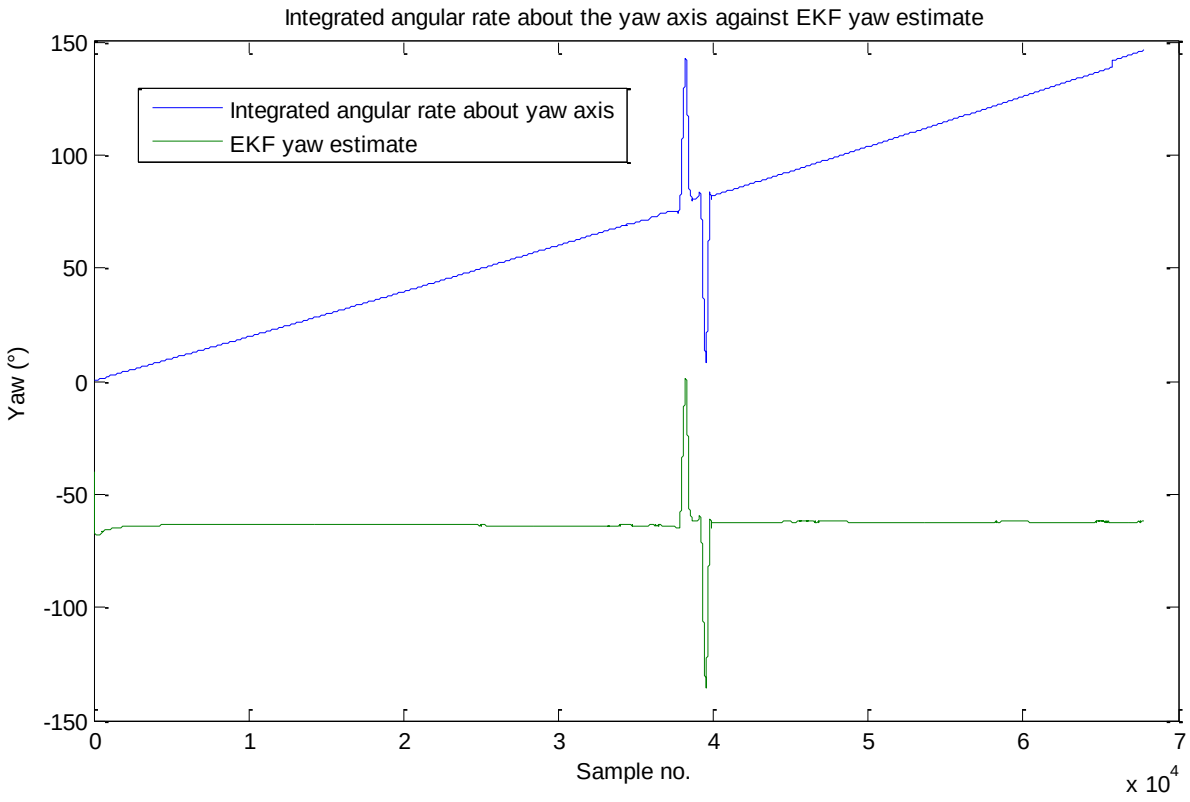

Figure 22 - Extended Kalman filter yaw estimate

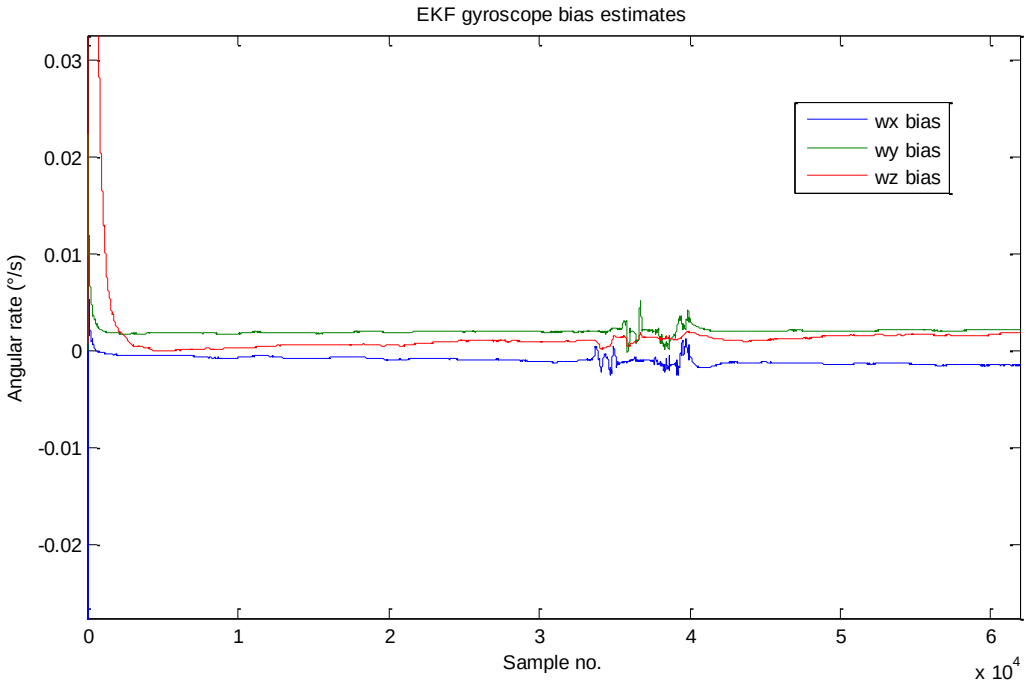

Figure 23 - EKF gyroscope bias estimate

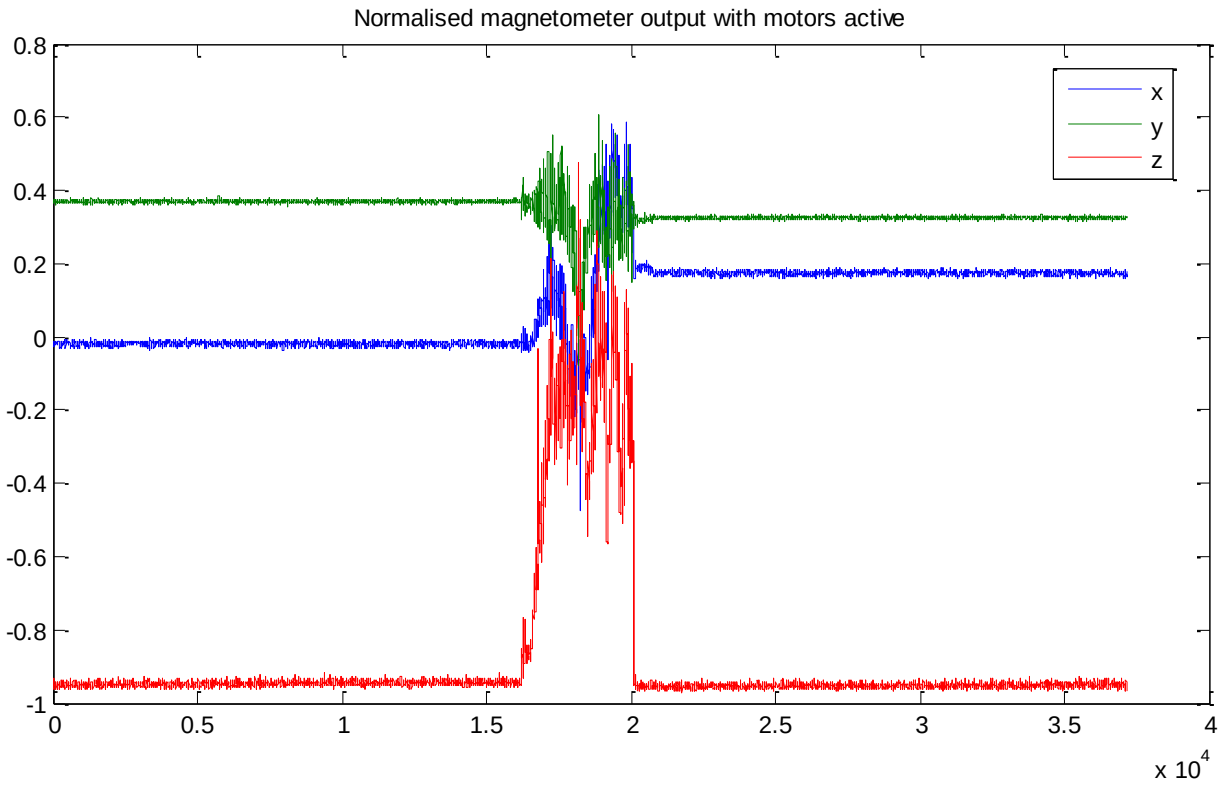

Figure 24 - Normalised magnetometer output with motors active

7 Control

It is the control scheme's job to translate the pilot's control demands into motor speeds. As mentioned in section 2.2, this is achieved using two PID layers for the pitch and roll axes, with a single layer for the yaw axis.

The first layer controls the quadrotor's angular velocity using a PI controller, taking either direct pilot commands or the output of the second control layer as the setpoint, and the raw gyroscope angular rate outputs as process variables. This layer is responsible for the quadrotor's stability, resisting all unwanted angular motion.

The second layer is a full PID loop, taking the pilots demanded attitudes as a setpoint and outputting a desired angular rate to the inner loop. The P and I terms take the AHRS's attitude as the process variable, with the D term being operated as derivative on measurement, as opposed to the usual derivative on error. This means the derivative of the angle estimate is taken, instead of the derivative of the error between this and the setpoint. This has the advantage that the derivative of the angle estimate is already available in the form of the raw gyroscope angular rates, and without any of the noise introduced by the EKF. Using this allows the D gain to be increased without resulting in large output spikes during setpoint changes, and reduces the amplification effect of the noise in the AHRS estimate. This is better described in Figure 25.

These loops are tuned by hand by adjusting the five gains until the desired response is achieved, with the same set of coefficients being used for both the pitch and roll axes due to the symmetry of the platform. There is no definite 'correct' tuning; it is simply a matter of achieving a response that meets the desired flight characteristics. For standard one degree of freedom PID controllers this generally means a trade-off between disturbance rejection and command tracking, so different tuning coefficients can be used for different applications. For example, a system with high disturbance rejection would be better suited to a low speed hovering application where quick dynamic movements are not expected, whereas a vehicle aiming to attempt aerial acrobatics would definitely be better suited to a command tracking tuning.

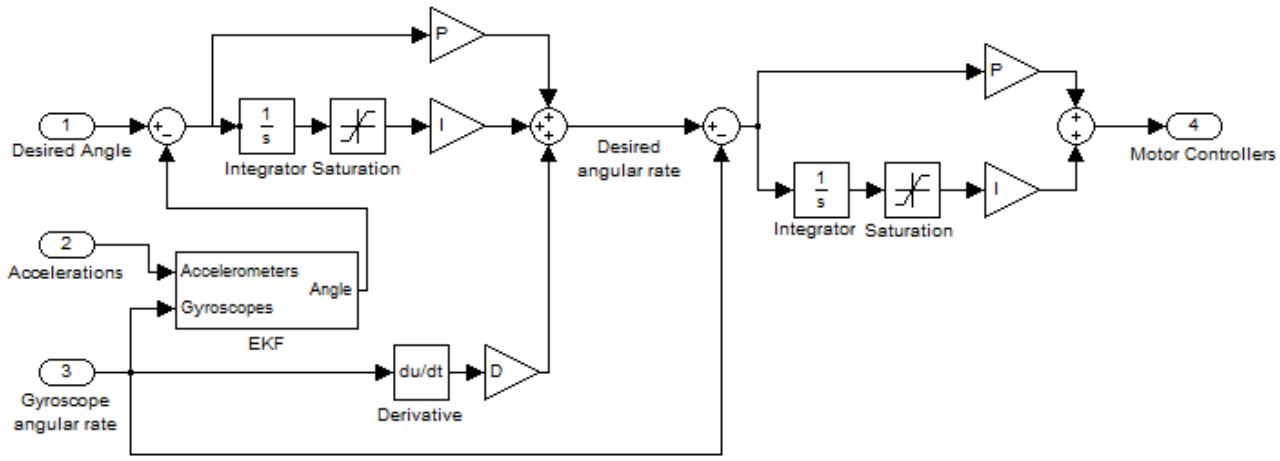

Figure 25 - Attitude control scheme

8 Conclusion

The quadrotor has proven as stable as a majority of commercial flight controllers, a lot more so than the lower end models. This was measured through experimentally evaluating its flight characteristics, with emphasis on how quickly the platform accelerates laterally when hovering with no pilot input. This stability may possibly be improved even further by the use of a better control scheme, but this project's focus on the AHRS development didn't allow the time for this to be explored.

The AHRS has also proven to fully function in any orientation, meeting another main criterion in the specification.

There exist a large number of changes and improvements I would make if I could do further work on this project:

8.1 Vibration Reduction

This quadrotor appears to suffer from vibration related problems more than the top commercially available models such as the AutoQuad flight controller. I believe this stems from two stark differences between this platform and the AutoQuad.

First, most implementations of the AutoQuad flight controller use a frame comprised of carbon fibre tubing, whereas I chose to use aluminium square tubes for arms. I've since found these to be much more transmissive to the high frequency vibrations from the motors than CF. This was then exasperated by the use of much cheaper motors and propellers than a typical AutoQuad setup, adding additional vibrations.

Second, I initially believed the 1KHz sampling frequency of the MPU6050 gyroscope/accelerometer would provide enough bandwidth to accurately capture the vehicles dynamics. However, it has become clear that a lot of the audible noise present during flight exists well above the 500Hz Nyquist frequency, which if measured by the inertial sensors will appear aliased as low frequency noise. The AutoQuad FC solves this problem by using only analogue sensors, performing the analogue to digital conversion using a separate dedicated IC. This allows the designer to specify their own sampling frequency, which for the AutoQuad is 262.4 KHz. This represents a huge degree of oversampling, which drives down the noise floor on both sensors, which in turn allows larger derivative coefficients to be used in attitude PID layer to increase stability.

8.2 Control Improvements

Since this project focused mostly on the development of the quadrotor's AHRS, only a short amount of time was available to be dedicated to developing the attitude control scheme. Given more time I would like to try and address the disturbance rejection vs. command tracking trade-off, as well as implementing velocity and position control layers.

8.3 Additional Sensor Integration

The addition of a GPS module would provide a position and velocity measurement. This allows the EKF to be extended from estimating just the attitude quaternion and gyroscope biases to also estimating the quadrotor's position, its velocity, and the bias of its accelerometers. This will also improve the accuracy of the original quaternion estimate, as the Kalman filter allows every

measurement to influence every state estimate if there is determined to be a covariance between the two.

8.4 Estimation Algorithm Improvement

An improvement to the non-linear Kalman filter was proposed in 1997 that allowed non-linear systems to be estimated without first performing a first order linearization of the model as in the extended Kalman filter (17). This became known as the unscented Kalman filter, which works by passing a number of points around the mean through the non-linear system, after which the true distorted mean and covariance can be recovered using the unscented transform. This completely solves the divergence issues inherent with the extended Kalman filter, and is actually computationally more efficient as a matrix inversion and two Jacobian calculations are omitted.

With more time I would like to attempt to implement these methods along with the addition of the position and velocity estimates. In the commercial world this type of filter has only been implemented in the AutoQuad controller, and is one of the main contributors to its exceptional performance.

9 Works Cited

1. *MultiWii repository.* [Online] http://code.google.com/p/multiwii/.

2. *Arducopter repository.* [Online] http://code.google.com/p/arducopter/.

3. *Autoquad.* [Online] http://autoquad.org/about/.

4. **K. Nomami, F. Kendoul, S. Suzuki, W. Wang, D. Nakazawa.** *Autonomous Flying Robots.* s.l. : Springer, 2010.

5. Quadrocopter yaw diagram. *Wikipedia.* [Online] http://upload.wikimedia.org/wikipedia/commons/2/2a/Quadrotor_yaw_torque.png.

6. Turnigy L2215J-900. *HobbyKing.* [Online] http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=14737.

7. Flightmax 5000mAh 20C LiPo. *HobbyKing.* [Online] http://www.hobbyking.com/hobbyking/store/__8579__ZIPPY_Flightmax_5000mAh_3S1P_20C.html.

8. **Invensense.** MPU6050 datasheet. [Online] www.invensense.com/mems/gyro/documents/PS-MPU-6000A.pdf.

9. MS5611-01BA Datasheet. [Online] www.meas-spec.com/downloads/MS5611-01BA03.pdf.

10. Tait-Bryan diagram. *Wikipedia.* [Online] http://upload.wikimedia.org/wikipedia/commons/6/67/Plane.svg.

11. **Hoag, David.** *Apollo Guidance and Navigation Considerations of Apollo IMU Gimbal Lock.* 1963.

12. **Diebel, James.** *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors.* s.l. : Stanford University, 2006.

13. **Solà, Joan.** *Quaternion kinematics for the error-state KF.* 2012.

14. *A Comparison of Complementary and Kalman Filtering.* **HIGGINS, WALTER T.** s.l. : IEEE, 1974.

15. **G. Minkler, J. Minkler.** *Theory and Application of Kalman Filtering.* 1993.

16. **Madgwick, Sebastian O.H.** *An efficient orientation filter for inertial and inertial/magnetic sensor arrays.* 2010.

17. *A New Extension of the Kalman Filter to Nonlinear Systems.* **Simon J. Julier, Jeffrey K. Uhlmann.** s.l. : The Robotics Research Group, Department of Engineering Science, The University of Oxford, 1997.

18. **Barraud, Alain.** MgnCalibration.m. *mathworks.co.uk.* [Online] http://www.mathworks.co.uk/matlabcentral/fileexchange/23398-magnetometers-calibration/content/MgnCalibration.m.

10 Appendices

10.1 MgnCalibration.m (10)

```
function [U,c] = MgnCalibration(X)
% performs magnetometer calibration from a set of data
% using Merayo technique with a non iterative algoritm
% J.Merayo et al. "Scalar calibration of vector magnemoters"
% Meas. Sci. Technol. 11 (2000) 120-132.
%
%   X      : a Nx3 (or 3xN) data matrix
%              each row (columns) contains x, y, z measurements
%              N must be such that the data set describes
%              as completely as possible the 3D space
%              In any case N > 10
%
%    The calibration tries to find the best 3D ellipsoid that fits the data
set
%    and returns the parameters of this ellipsoid
%
%    U     :  shape ellipsoid parameter, (3x3) upper triangular matrix
%    c      : ellipsoid center, (3x1) vector
%
%    Ellipsoid equation : (v-c)'*(U'*U)(v-c) = 1
%    with v a rough triaxes magnetometer  measurement
%
%    calibrated measurement w = U*(v-c)
%
%   author : Alain Barraud, Suzanne Lesecq 2008
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[N,m] = size(X);
if m>3&&N==3,X = X';N = m;m = 3;end;%check that X is not transposed
if N<=10,U = [];c = [];return;end;%not enough data no calibration !!
% write  the ellipsoid equation as D*p=0
% the best parameter is the solution of min||D*p|| with ||p||=1;
% form D matrix from X measurements
x = X(:,1); y = X(:,2); z = X(:,3);
D = [x.^2, y.^2, z.^2, x.*y, x.*z, y.*z, x, y, z, ones(N,1)];
D=triu(qr(D));%avoids to compute the svd of a large matrix
[U,S,V] = svd(D);%because usually N may be very large
p = V(:,end);if p(1)<0,p =-p;end;
% the following matrix A(p) must be positive definite
% The optimization done by svd does not include such a constraint
% With "good" data the constraint is allways satisfied
% With too poor data A may fail to be positive definite
% In this case the calibration fails
%
A = [p(1) p(4)/2 p(5)/2;
       p(4)/2 p(2) p(6)/2;
       p(5)/2 p(6)/2 p(3)];
[U,ok] = fchol(m,A);
if ~ok,U = [];c = [];return;end%calibration fails too poor data!!
b = [p(7);p(8);p(9)];
v = Utsolve(U,b/2,m);
d = p(10);
s = 1/sqrt(v*v'-d);
c =-Usolve(U,v,m)';%ellipsoid center
U = s*U;%shape ellipsoid parameter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [A,ok] = fchol(n,A)
% performs Cholesky factoristation
A(1,1:n) = A(1,1:n)/sqrt(A(1,1));
A(2:n,1) = 0;
for j=2:n
  A(j,j:n) = A(j,j:n) - A(1:j-1,j)'*A(1:j-1,j:n);
  if A(j,j)<=0,ok=0;break;end%A is not positive definite
  A(j,j:n) = A(j,j:n)/sqrt(A(j,j));
  A(j+1:n,j) = 0;
end
ok=1;
function x=Utsolve(U,b,n)
% solves U'*x=b
x(1) = b(1)/U(1,1);
for k=2:n
    x(k) = (b(k)-x(1:k-1)*U(1:k-1,k))/U(k,k);
end
function x=Usolve(U,b,n)
% solves U*x=b
x(n) = b(n)/U(n,n);
for k=n-1:-1:1
    x(k) = (b(k)-U(k,k+1:n)*x(k+1:n)')/U(k,k);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

10.2 ApplyMgnCalibration.m

```
function [ xnew,ynew,znew ] = ApplyMgnCalibration( U, c, x,y,z )

for (i = 1:length(x))

    temp0 = x(i) - c(1);
    temp1 = y(i) - c(2);
    temp2 = z(i) - c(3);

    xnew(i,1) = U(1,1)*temp0 + U(1,2)*temp1 + U(1,3)*temp2;
    ynew(i,1) = U(2,2)*temp1 + U(2,3)*temp2;
    znew(i,1) = U(3,3)*temp2;

end
```

10.3 LKF.m

```
function [pitch, roll, bp, bq, P_out] = LKF( p, q, accelpitch, accelroll,
dt )

pitch = zeros(length(dt),1);
roll = zeros(length(dt),1);
bp = zeros(length(dt),1);
bq = zeros(length(dt),1);
P_out = zeros(length(dt),4,4);

Q_pitch = 0.0; % Calculated from data
Q_roll = 0.0; % Calculated from data
Q_bp = 0.1;
Q_bq = 0.1;
R_pitch = 50; % Accel variance, calculated from data
R_roll = 50; % Accel variance, calculated from data

Q = [Q_pitch 0     0    0
     0     Q_roll 0    0
     0     0     Q_bp 0
     0     0     0    Q_bq];

R = [R_pitch 0     0 0
     0     R_roll 0 0
     0     0     0 0
     0     0     0 0];

P = [0 0 0 0
     0 0 0 0
     0 0 10000 0
     0 0 0 10000];

H = [1 0 0 0
     0 1 0 0
     0 0 0 0
     0 0 0 0];

I = [1 0 0 0
     0 1 0 0
     0 0 1 0
     0 0 0 1];

x = [accelpitch(1) % initial state estimate
     accelroll(1)
     0
     0];

for i = 2:1:length(dt)
    F = [1 0 -dt(i) 0
         0 1 0      -dt(i)
         0 0 1      0
         0 0 0      1];

    B = [dt(i) 0     0 0
```

```
        0     dt(i) 0 0
        0     0     0 0
        0     0     0 0];

    u = [p(i)
         q(i)
         0
         0];

    z = [accelpitch(i)
         accelroll(i)
         0
         0];

    % Predicted state estimate
    x = F*x + B*u;

    % Predicted estimate covariance
    P = F*P*F' + dt(i)*Q;

    % Measurement residual
    y = z - H*x;

    % Residual covariance
    S = H*P*H' + R;

    % Optimal Kalman gain
    K = P*H'*pinv(S);

    % Updated state estimate
    x = x + K*y;

    % Updated estimate covariance
    P = (I - K*H)*P;

    pitch(i) = x(1);
    roll(i) = x(2);
    bp(i) = x(3);
    bq(i) = x(4);
    P_out(i,1:4,1:4) = P(1:4,1:4);
end
end
```

10.4 EKFSimple_CALL.m

```
function [ quaternion, wb, pitch, roll, yaw, error ] = EKFSimple_CALL(
ax,ay,az,p,q,r,dt )

a = [ax,ay,az];
w = [p,q,r];
quaternion = zeros(length(dt),4);
wb = zeros(length(dt),3);
pitch = zeros(length(dt),1);
roll = zeros(length(dt),1);
clear EKFSimple;

for i = 1:length(dt)
    [quaternion(i,:), wb(i,:), pitch(i,:), roll(i,:), yaw(i,:), error(i,:)]
= EKFSimple(a(i,:),w(i,:),dt(i,1));
end

end
```

10.5 EKFSimple.m

```
function [ q, wb, pitch, roll, yaw, error ] = EKFSimple( a,w,dt )


persistent x P;

% Covariance matrices
Q = [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, 0, 0, 0;
    0, 0, 0, 0, 0.2, 0, 0;
    0, 0, 0, 0, 0, 0.2, 0;
    0, 0, 0, 0, 0, 0, 0.2];

R = [1e6,    0,    0;
       0,  1e6,    0;
       0,    0, 1e6];


if isempty(P)
    P = eye(7);
    P = P*10000;

    x = [1, 0, 0, 0, 0, 0, 0]';
end

q0 = x(1);
q1 = x(2);
q2 = x(3);
q3 = x(4);
wxb = x(5);
wyb = x(6);
wzb = x(7);
```

```
wx = w(1);
wy = w(2);
wz = w(3);

z = [a(1); a(2); a(3)];


%%%%%%%%% PREDICT %%%%%%%%%
% Predicted state estimate
% x = f(x,u)
x = [q0 + (dt/2) * (-q1*(wx-wxb) - q2*(wy-wyb) - q3*(wz-wzb));
     q1 + (dt/2) * ( q0*(wx-wxb) - q3*(wy-wyb) + q2*(wz-wzb));
     q2 + (dt/2) * ( q3*(wx-wxb) + q0*(wy-wyb) - q1*(wz-wzb));
     q3 + (dt/2) * (-q2*(wx-wxb) + q1*(wy-wyb) + q0*(wz-wzb));
     wxb;
     wyb;
     wzb;];

% Normalize quaternion
qnorm = sqrt(x(1)^2 + x(2)^2 + x(3)^2 + x(4)^2);
x(1) = x(1)/qnorm;
x(2) = x(2)/qnorm;
x(3) = x(3)/qnorm;
x(4) = x(4)/qnorm;

q0 = x(1);
q1 = x(2);
q2 = x(3);
q3 = x(4);

% Populate F Jacobian
F = [              1, -(dt/2)*(wx-wxb), -(dt/2)*(wy-wyb), -(dt/2)*(wz-wzb),
(dt/2)*q1,  (dt/2)*q2,  (dt/2)*q3;
     (dt/2)*(wx-wxb),                1,  (dt/2)*(wz-wzb), -(dt/2)*(wy-wyb),
-(dt/2)*q0,  (dt/2)*q3, -(dt/2)*q2;
     (dt/2)*(wy-wyb), -(dt/2)*(wz-wzb),                1,  (dt/2)*(wx-wxb),
-(dt/2)*q3, -(dt/2)*q0,  (dt/2)*q1;
     (dt/2)*(wz-wzb),  (dt/2)*(wy-wyb), -(dt/2)*(wx-wxb),                1,
(dt/2)*q2, -(dt/2)*q1, -(dt/2)*q0;
                   0,                0,                0,                0,
1,          0,          0;
                   0,                0,                0,                0,
0,          1,          0;
                   0,                0,                0,                0,
0,          0,          1;];


% Predicted covariance estimate
P = F*P*F' + Q;


%%%%%%%%%% UPDATE %%%%%%%%%%
% Normalize accelerometer and magnetometer measurements
acc_norm = sqrt(z(1)^2 + z(2)^2 + z(3)^2);
z(1) = z(1)/acc_norm;
z(2) = z(2)/acc_norm;
z(3) = z(3)/acc_norm;
```

```
h = [-2*(q1*q3 - q0*q2);
     -2*(q2*q3 + q0*q1);
     - q0^2 + q1^2 + q2^2 - q3^2;];

%Measurement residual
% y = z - h(x), where h(x) is the matrix that maps the state onto the
measurement
y = z - h;


% Populate H Jacobian
H = [ 2*q2, -2*q3,  2*q0, -2*q1, 0, 0, 0;
     -2*q1, -2*q0, -2*q3, -2*q2, 0, 0, 0;
     -2*q0,  2*q1,  2*q2, -2*q3, 0, 0, 0;];

% Residual covariance
S = H*P*H' + R;

% Calculate Kalman gain
K = P*H'/S;

% Update state estimate
x = x + K*y;

% Normalize quaternion
qnorm = sqrt(x(1)^2 + x(2)^2 + x(3)^2 + x(4)^2);
x(1) = x(1)/qnorm;
x(2) = x(2)/qnorm;
x(3) = x(3)/qnorm;
x(4) = x(4)/qnorm;
q0 = x(1);
q1 = x(2);
q2 = x(3);
q3 = x(4);


% Update estimate covariance
I = eye(length(P));
P = (I - K*H)*P;

% Generate YPR values
pitch = (180/pi) * atan2(2*(q0*q1+q2*q3),1-2*(q1^2+q2^2));
roll = (180/pi) * asin(2*(q0*q2-q3*q1));
yaw = (180/pi) * atan2(2*(q0*q3+q1*q2),1-2*(q2^2+q3^2));


q = [x(1), x(2), x(3), x(4)];
wb = [x(5), x(6), x(7)];
error = y;

end
```

10.6 EKF_CALL.m

```
function [ quaternion, wb, pitch, roll, yaw, error ] = EKF_CALL(
ax,ay,az,p,q,r,mx,my,mz,dt )

a = [ax,ay,az];
w = [p,q,r];
m = [mx,my,mz];
quaternion = zeros(length(dt),4);
wb = zeros(length(dt),3);
pitch = zeros(length(dt),1);
roll = zeros(length(dt),1);
clear EKF;

for i = 1:length(dt)
    [quaternion(i,:), wb(i,:), pitch(i,:), roll(i,:), yaw(i,:), error(i,:)]
= EKF(a(i,:),w(i,:),m(i,:),dt(i,1));
end

end
```

10.7 EKF.m

```
function [ q, wb, pitch, roll, yaw, error, b ] = EKF( a,w,m,dt )


persistent x P;

% Covariance matrices
Q = [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, 0, 0, 0;
    0, 0, 0, 0, 0.2, 0, 0;
    0, 0, 0, 0, 0, 0.2, 0;
    0, 0, 0, 0, 0, 0, 0.2];

R = [500000,0,0,0,0,0;
     0,500000,0,0,0,0;
     0,0,500000,0,0,0;
     0,0,0,10000000,0,0;
     0,0,0,0,10000000,0;
     0,0,0,0,0,10000000;];


if isempty(P)
    P = eye(7,7)*100000000; % No idea what the initial state is
    x = [1, 0, 0, 0, 0, 0, 0]';
end

q0 = x(1);
q1 = x(2);
q2 = x(3);
```

```
q3 = x(4);
wxb = x(5);
wyb = x(6);
wzb = x(7);
wx = w(1);
wy = w(2);
wz = w(3);

z = [a(1); a(2); a(3); m(1); m(2); m(3)];


%%%%%%%%% PREDICT %%%%%%%%%
% Predicted state estimate
% x = f(x,u)
x = [q0 + (dt/2) * (-q1*(wx-wxb) - q2*(wy-wyb) - q3*(wz-wzb));
     q1 + (dt/2) * ( q0*(wx-wxb) - q3*(wy-wyb) + q2*(wz-wzb));
     q2 + (dt/2) * ( q3*(wx-wxb) + q0*(wy-wyb) - q1*(wz-wzb));
     q3 + (dt/2) * (-q2*(wx-wxb) + q1*(wy-wyb) + q0*(wz-wzb));
     wxb;
     wyb;
     wzb;];

% Normalize quaternion
qnorm = sqrt(x(1)^2 + x(2)^2 + x(3)^2 + x(4)^2);
x(1) = x(1)/qnorm;
x(2) = x(2)/qnorm;
x(3) = x(3)/qnorm;
x(4) = x(4)/qnorm;

q0 = x(1);
q1 = x(2);
q2 = x(3);
q3 = x(4);

% Populate F Jacobian
F = [              1, -(dt/2)*(wx-wxb), -(dt/2)*(wy-wyb), -(dt/2)*(wz-wzb),
(dt/2)*q1,  (dt/2)*q2,  (dt/2)*q3;
     (dt/2)*(wx-wxb),                 1,  (dt/2)*(wz-wzb), -(dt/2)*(wy-wyb),
-(dt/2)*q0,  (dt/2)*q3, -(dt/2)*q2;
     (dt/2)*(wy-wyb), -(dt/2)*(wz-wzb),                 1,  (dt/2)*(wx-wxb),
-(dt/2)*q3, -(dt/2)*q0,  (dt/2)*q1;
     (dt/2)*(wz-wzb),  (dt/2)*(wy-wyb), -(dt/2)*(wx-wxb),                 1,
(dt/2)*q2, -(dt/2)*q1, -(dt/2)*q0;
                   0,                 0,                 0,                 0,
1,          0,          0;
                   0,                 0,                 0,                 0,
0,          1,          0;
                   0,                 0,                 0,                 0,
0,          0,          1;];


% Predicted covariance estimate
P = F*P*F' + Q;

%%%%%%%%%% UPDATE %%%%%%%%%%
% Normalize accelerometer and magnetometer measurements
acc_norm = sqrt(z(1)^2 + z(2)^2 + z(3)^2);
z(1) = z(1)/acc_norm;
z(2) = z(2)/acc_norm;
```

```
z(3) = z(3)/acc_norm;

mag_norm = sqrt(z(4)^2 + z(5)^2 + z(6)^2);
z(4) = z(4)/mag_norm;
z(5) = z(5)/mag_norm;
z(6) = z(6)/mag_norm;

% Reference field calculation
% Build quaternion rotation matrix
Rq = [q0^2+q1^2-q2^2-q3^2,     2*(q1*q2-q0*q3),      2*(q1*q3+q0*q2);
          2*(q1*q2+q0*q3), q0^2-q1^2+q2^2-q3^2,      2*(q2*q3-q0*q1);
          2*(q1*q3-q0*q2),     2*(q2*q3+q0*q1), q0^2-q1^2-q2^2+q3^2];
% Rotate magnetic vector into reference frame
Rm = Rq * [z(4); z(5); z(6)];
bx = sqrt(Rm(1)^2 + Rm(2)^2);
bz = Rm(3);

h = [-2*(q1*q3 - q0*q2);
     -2*(q2*q3 + q0*q1);
     - q0^2 + q1^2 + q2^2 - q3^2;
     bx*(q0^2 + q1^2 - q2^2 - q3^2) + 2*bz*(q1*q3 - q0*q2);
     2*bx*(q1*q2 - q0*q3) + 2*bz*(q2*q3 + q0*q1);
     2*bx*(q1*q3 + q0*q2) + bz*(q0^2 - q1^2 - q2^2 + q3^2);];

% Measurement residual
% y = z - h(x), where h(x) is the matrix that maps the state onto the
measurement
y = z - h;


% Populate H Jacobian
H = [ 2*q2, -2*q3,  2*q0, -2*q1, 0, 0, 0;
     -2*q1, -2*q0, -2*q3, -2*q2, 0, 0, 0;
     -2*q0,  2*q1,  2*q2, -2*q3, 0, 0, 0;
      2*( q0*bx - q2*bz), 2*( q1*bx + q3*bz), 2*(-q2*bx - q0*bz), 2*(-q3*bx
+ q1*bz), 0, 0, 0;
      2*(-q3*bx + q1*bz), 2*( q2*bx + q0*bz), 2*( q1*bx + q3*bz), 2*(-q0*bx
+ q2*bz), 0, 0, 0;
      2*( q2*bx + q0*bz), 2*( q3*bx - q1*bz), 2*( q0*bx - q2*bz), 2*( q1*bx
+ q3*bz), 0, 0, 0];

% Residual covariance
S = H*P*H' + R;

% Calculate Kalman gain
K = P*H'/S;

% Update state estimate
x = x + K*y;

% Normalize quaternion
qnorm = sqrt(x(1)^2 + x(2)^2 + x(3)^2 + x(4)^2);
x(1) = x(1)/qnorm;
x(2) = x(2)/qnorm;
x(3) = x(3)/qnorm;
x(4) = x(4)/qnorm;
q0 = x(1);
q1 = x(2);
q2 = x(3);
```

```
q3 = x(4);


% Update estimate covariance
I = eye(length(P));
P = (I - K*H)*P;

% Generate YPR values
pitch = (180/pi) * atan2(2*(q0*q1+q2*q3),1-2*(q1^2+q2^2));
roll = (180/pi) * asin(2*(q0*q2-q3*q1));
yaw = (180/pi) * atan2(2*(q0*q3+q1*q2),1-2*(q2^2+q3^2));


q = [x(1), x(2), x(3), x(4)];
wb = [x(5), x(6), x(7)];
error = y;
b = [bx, 0, bz];

end
```

10.8 FC and Microcontroller Code

The code for the flight controller and the onboard microcontroller is available at https://github.com/big5824/Picopter and https://github.com/big5824/PicopterPic. This is also included on a CD with this report. The Matlab scripts and all figures used are also available in the first repository. This code is all my own work, with exception of the previously referenced MgnCalibration.m function and the C++ Eigen matrix library, which is included in a separate directory.