Anthony Grace

9-DOF IMU Razor AHRS Code Documentation and Discussion

This document should be used as an instructional guide for the 9DOF IMU Razor_AHRS code with quaternions. The original Razor_AHRS code provided from https://www.sparkfun.com/products/10736 was a DCM (Direction Cosine Matrix) based attitude estimation. The above code was altered to implement quaternion based attitude estimation.

The structure of this document matches the structure of the code. Such as, the Razor_AHRS Tab is covered first, then the Compass, and so on. When looking through the code, you will see areas commented out saying "refer to page # for further explanation." Most discussions of the code happen in this document, but for certain sections it was easier to explain in the code with a comment.

Finally at the end of this document a discussion of position estimation is covered. An attempt to estimate the IMU position from the accelerometer readings was made. This was approached using the basic kinematic equations and filtering of the accelerometer data to remove noise.

Razor_AHRS Tab:

Accelerometer Min and Max calibration values:

First, the format of the input of the min and max values in the code look as follows:

```
#define ACCEL_X_MIN ((float) Put Value Here)
#define ACCEL_X_MAX ((float) Put Value Here)
#define ACCEL_Y_MIN ((float) Put Value Here)
#define ACCEL_Y_MAX ((float) Put Value Here)
#define ACCEL_Z_MIN ((float) Put Value Here)
#define ACCEL_Z_MAX ((float) Put Value Here)
```

Where the "Put Value Here" is, enter in the calculated values which are obtained from the below description. To obtain the Min and Max accelerometer values used for the sensor calibration several steps were taken and will be covered next.

While the IMU is connected via USB, open the serial monitor by going to tools and selecting "serial monitor". When opened the serial monitor will begin outputting the Yaw-Pitch-Roll angles. In the command/text box above the data window, type #osrt, this will switch the data output to the raw sensor data for each sensor and each axis (this results in a total of 9 readings). The data output looks as follows:

```
#A-R=-59.00,3.00,-251.00
#M-R=273.00,-331.00,-280.00
#G-R=-52.00,75.00,32.00
#A-R=-64.00,9.00,-245.00
#M-R=277.00,-329.00,-282.00
#G-R=-45.00,67.00,-34.00
```

A-R means accelerometer output, M-R means magnetometer output, and G-R means Gyro output.

We are only concerned with the accelerometer readings. Now do the following steps once the raw sensor data has begun streaming:

- Hold the accelerometer with the board parallel to the ground (gravity will be pointing down through the z-axis of the IMU). This will be the positive z-axis pointing up, hold it there for 15-30 seconds.

- In the command/text box type #o0 to stop the continuous streaming.

- The data then should be copied from the serial monitor and put into the desired program to calculate the average (Excel, Matlab, etc.).

- Once in desired program, take the average of only the z-axis values, this will be the ACCEL_Z_MAX value.

- To obtain the minimum value, redo the above steps but with the positive z-axis pointing down (IMU is rotated 180 degrees about the y-axis given on the board).

- Repeat the same process for the x-axis and y-axis.

It is important to note that in this code, the x-axis and y-axis of the board are swapped, the positive x-axis is pointing in the direction of the positive y-axis defined on the board, and the positive y-axis is pointing in the positive x-axis defined on the board. For more accurate min and max values hold the board in described positions longer for more data resulting in a better average.

If noticed, the accelerometer readings are not near ±1g, as would be expected, if not moving. The value of 1g used in the code is 256.0 (why? Still need to figure this out if possible). So the accelerometer data would need to be divided by 256 to get the raw data readings in "g".

Other Sensor Calibration values:

For the other two sensors, gyro and magnetometer, the tutorial link provided in the pre-developed code was used. The tutorial link provided in the code is: http://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs.

The above tutorial was used for the accelerometer calibration originally as well, but it proved to be difficult to get good values. If the accelerometer min and max values are not accurate, it will result in drifting and bad attitude estimation.

reset_sensor_fusion() Explanation:

The first two operations after the local variables are defined are: read_sensors() and timestamp = millis(). The "read_sensors()" function calls out for the sensor data from the gyro, accelerometer, and magnetometer. The time is then recorded to the variable timestamp using the function "millis()". The function "millis" is a built in function of the arduino program.

Next is the operation to get the Euler angle of pitch, which is the rotation around the y-axis. Recall, as discussed above, the x-axis of the board is defined to be the y-axis and vice versa. It can be shown that the rotation matrix, also known as DCM (Direction Cosine Matrix), is from inertial frame to the body frame is given as:

$$DCM_I^b = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\theta\sin\phi \\ \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix} \quad (1)$$

Also, the transpose of DCM_I^b equal to the rotation matrix from the body frame to the inertial frame shown below as:

$$DCM_b^I = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \quad (2)$$

Now for the initial calculation of the pitch angle it's assumed there is no acceleration other than gravity on the IMU. This gives us that that acceleration in the inertial frame is:

$$\vec{a}^I = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Next, rotate this vector into the body frame using the rotation matrix DCM_I^b as follows:

$$\frac{\vec{a}^b}{\|\vec{a}^b\|} = C_I^b\vec{a}^I = \begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} \quad (3)$$

From Eq. 3 each element of $\vec{a}^b$ is known. The $\vec{a}^b$ elements are the accelerometer outputs. Thus we can obtain the pitch angle "θ" from:

$$\sin\theta = -\frac{a_x^b}{\|\vec{a}^b\|} \tag{4}$$

$$\theta = -\sin^{-1}\left(\frac{\vec{a}_x^b}{\|\vec{a}^b\|}\right) \tag{5}$$

It can be further shown that:

$$\theta = -\tan^{-1}\left(\frac{a_x^b}{\sqrt{\left(a_y^b\right)^2 + \left(a_z^b\right)^2}}\right) \tag{6}$$

This result in Eq. 6 comes from the following steps:

1)

$$\frac{a_y^b}{a_z^b} = \frac{\cos\theta\sin\phi}{\cos\theta\cos\phi} = \tan\phi$$

2)

$$\frac{a_y^b}{\|\vec{a}^b\|} = \cos\theta\sin\phi = \cos\theta\sin\left(\tan^{-1}\left(\frac{a_y^b}{a_z^b}\right)\right)$$

3)

$$\frac{a_y^b}{\|\vec{a}^b\|} = \frac{a_y^b\cos\theta}{a_z^b\sqrt{\frac{\left(a_y^b\right)^2}{\left(a_z^b\right)^2}+1}} = \frac{a_y^b\cos\theta}{\sqrt{\left(a_y^b\right)^2+\left(a_z^b\right)^2}}$$

4)

$$\frac{1}{\|\vec{a}^b\|} = \frac{\cos\theta}{\sqrt{\left(a_y^b\right)^2+\left(a_z^b\right)^2}}$$

5)

$$\sin\theta = -\frac{a_x^b}{\|\vec{a}^b\|} \rightarrow \|\vec{a}^b\| = -\frac{a_x^b}{\sin\theta}$$

6)

$$\frac{-\sin\theta}{a_x^b} = \frac{\cos\theta}{\sqrt{\left(a_y^b\right)^2 + \left(a_z^b\right)^2}} \rightarrow \theta = -\tan^{-1}\left(\frac{a_x^b}{\sqrt{\left(a_y^b\right)^2 + \left(a_z^b\right)^2}}\right)$$

The initial roll, "ϕ", is than calculated by projecting the acceleration vector of the body into the y-z plane. This is done by taking the cross product of the body acceleration with the x-axis and than the cross product of the x-axis with the resulting vector, shown as follows:

$$\vec{a}_{y-z\,plane} = \hat{\imath}\times\left(\vec{a}^b\times\hat{\imath}\right) = \begin{bmatrix} 0 \\ a_y^b \\ a_z^b \end{bmatrix} \tag{7}$$

Phi is then obtained as:

$$\phi = \tan^{-1}\left(\frac{a_y^b}{a_z^b}\right) \tag{8}$$

The yaw is the final initial Euler angle to be calculated given as "ψ". To calculate the yaw angle the magnetometer must be used. It can be seen that a function is used called "Compass_Heading()". This function can be found by selecting the tab called "Compass" at the top of the code window. Refer to page # for an explanation of this function.

Once the initial Euler Angles are found the initial DCM (Direction Cosine Matrix) is constructed, also known as a rotation matrix. From the initial DCM, the initial quaternions are found. These operations are done using "init_rotation_matrix(DCM_Matrix, yaw, pitch, roll)" and "Quaternion_init()". The init_rotation_matrix and Quaternion_init functions are discussed under the Math Tab section starting on page 17.

compensate_sensor_errors Explanation:

I want to note that this explanation is given with my best understanding of this function and needs to be doubled checked. For the accelerometer error compensation, a linear model of the sensor is given as:

$$\vec{a}^b = K\left[T_b^p\right]^{-1}\vec{a}^p_{corrected} + \vec{a}^b_{offset} \tag{9}$$

The variables used in Eq.9 above are defined as follows:

$$K = \begin{bmatrix} \frac{1}{a_{x_{scale}}} & 0 & 0 \\ 0 & \frac{1}{a_{y_{scale}}} & 0 \\ 0 & 0 & \frac{1}{a_{z_{scale}}} \end{bmatrix} \tag{10}$$

$$\vec{a}_{scale} = \begin{bmatrix} \frac{Gravity}{a_{x_{max}} - a_{x_{offset}}} \\ \frac{Gravity}{a_{y_{max}} - a_{y_{offset}}} \\ \frac{Gravity}{a_{z_{max}} - a_{z_{offset}}} \end{bmatrix} \tag{11}$$

$$\vec{a}_{offset} = \begin{bmatrix} \frac{a_{x_{min}} + a_{x_{max}}}{2} \\ \frac{a_{y_{min}} + a_{y_{max}}}{2} \\ \frac{a_{z_{min}} + a_{z_{max}}}{2} \end{bmatrix} \tag{12}$$

T_b^p is the rotation matrix from the body to the platform. The platform seems to be the actual alignment of the accelerometer axes compared to the body axes. Manufacturing inaccuracies cause this difference/error. The matrix is defined as:

$$T_b^p = \begin{bmatrix} 1 & -\alpha_{yz} & \alpha_{zy} \\ \alpha_{xz} & 1 & -\alpha_{zx} \\ -\alpha_{xy} & \alpha_{yx} & 1 \end{bmatrix} \tag{13}$$

Each off diagonal component describes the inter-axis misalignments. For the code these values appear to be assumed as zero, giving an Identity matrix for T_b^p. Now solving Eq. 9 for:

$$\vec{a}^b_{corrected} = \left[T_b^p\right]^{-1} \vec{a}^p_{corrected} \tag{14}$$

The result is:

$$\vec{a}^b_{corrected} = \left[T_b^p\right]^{-1} \vec{a}^p_{corrected} = K^{-1}\left[\vec{a}^b - \vec{a}^b_{offset}\right] \tag{15}$$

$$\vec{a}^b_{corrected} = \begin{bmatrix} a_{x_{scale}} & 0 & 0 \\ 0 & a_{y_{scale}} & 0 \\ 0 & 0 & a_{z_{scale}} \end{bmatrix} \left[\vec{a}^b - \vec{a}^b_{offset}\right] \tag{16}$$

Eq. 16 is exactly what's implemented in this function for accelerometers. Two good resources used to understand the above discussion are the following two links:

http://www.diva-portal.org/smash/get/diva2:450820/FULLTEXT01
http://www.iaeng.org/publication/WCE2011/WCE2011_pp2164-2167

For the magnetometer, the extended magnetometer calibration was used. So in this section the statement, #if CALIBRATION__MAGN_USE_EXTENDED == true, is true; therefore the else statement is ignored and the extended method is used. This subject was not thoroughly looked into for how the extended calibration was completed and sensor error was compensated.

To compensate the gyro offset, the average offset given as Eq. 12 is subtracted from the raw gyro data.

Run quaternion algorithm Explanation:

This is where the algorithms are called to correct and update the quaternions for the attitude estimation of the IMU board. Each function is called for different steps of the algorithm. As seen in the code, the functions are called in the order below:

```
Compass_Heading();
Quaternion_update();
Drift_correction();
Euler_angles();
```

First is the "Compass_Heading()" function, which calculates the magnetic heading and measured yaw angle. This function can be seen under the Compass Tab of the code and is discussed further on below under Compass Tab.

The next function is the "Quaternion_update()". The function propagates the quaternions forward in time, allowing for the attitude estimation to occur. This function is discussed starting on page 14 under the DCM Tab discussion.

The third command is the "Drift_correction" function, which is applied to remove the errors that occur in the quaternion update and sensor readings. Proportional-Integral (PI) feedback is used to apply the drift correction. This function is also explained further, on page 11 under the DCM Tab topic.

Finally the Euler angles are calculated at the current time step from the quaternions using the function "Euler_angles". The Euler_angles function is discussed starting on page 15 under DCM Tab as well.

Compass Tab:

Compass_Heading() Explanation:

The purpose of this function is to obtain the magnetic heading along with its x and y components from the magnetometer readings. The magnetic heading and its x and y components are used to obtain the measured yaw angle, which is than used in the "Razor_AHRS" tab for the initial yaw angle, and in the "DCM" tab for correcting the yaw error.

Since the magnetometer sensor values are known in the body frame, we need to calculate the heading in the inertial frame. To calculate the magnetic heading in the inertial frame, the rotation matrix DCM_b^I given in Eq. 2 is used. Now the yaw angle is set to zero in DCM_b^I to get:

$$DCM_b^I(\psi = 0) = \begin{bmatrix} \cos\theta & \sin\theta\sin\phi & \cos\phi\sin\theta \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{17}$$

The reason we set the yaw angle "ψ" is because it results in a rotation matrix that only rotates about the roll axis and than the pitch axis. This rotation essentially projects the x-y plane of the body into the x-y inertial frame. This is an intermediate frame between the body and inertial, to be in inertial we would have to rotate about the z-axis by the yaw angle. This is the reason to do this intermediate rotation, it allows for the rotation of the magnetic field vector in the body frame into this intermediate frame, which is offset by the yaw angle from the inertial frame. The x and y components of this rotated magnetic field vector can be used to get the yaw angle or also known as the magnetic heading. This intermediate frame is called the coarse-over-ground (COG). So we will call the rotation matrix from the body to COG as follows:

$$C_b^{COG} = C_b^I(\psi = 0) \tag{18}$$

The magnetometer reading in the body frame is defined as:

$$\vec{M}^b = \begin{bmatrix} m_x^b \\ m_y^b \\ m_z^b \end{bmatrix} \tag{19}$$

Appling the rotation form body to COG to the magnetometer reading results in:

$$\vec{M}^{COG} = C_b^{COG}\vec{M}^b = \begin{bmatrix} \cos\theta & \sin\theta\sin\phi & \cos\phi\sin\theta \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} m_x^b \\ m_y^b \\ m_z^b \end{bmatrix} \tag{20}$$

$$\vec{M}^{COG} = \begin{bmatrix} m_x^b\cos\theta + m_y^b\sin\theta\sin\phi + m_z^b\cos\phi\sin\theta \\ m_y^b\cos\phi - m_z^b\sin\phi \\ -m_x^b\sin\theta + m_y^b\cos\theta\sin\phi + m_z^b\cos\theta\cos\phi \end{bmatrix} \tag{21}$$

To obtain the magnetic heading, the x and y components of $\vec{M}^{COG}$ are used:

$$Magnetic\ heading = \psi_{measured} = \tan^{-1}\left(\frac{-M_y^{COG}}{M_x^{COG}}\right) \tag{22}$$

The negative compensates for defining the z-axis of the board down and making the y-axis of the board equal to the x-axis of the board and vice versa (I was never able to confirm this logic but I believe it is correct).

It should be noted that this yaw angle calculated above is not only for the initial value of the yaw but also will be used as the measured yaw value "$\psi_{measured}$". This is important for removing drifting errors from the yaw angle and will be discussed in further detail on page 12 under the DCM Tab explanation.

DCM Tab:

Drift_correction(void) Explanation:

To begin, the paper "Direction Cosine Matrix IMU: Theory" by William Premerlani and Paul Bizard was used to understand the structure and math of the drift correction applied in this function. First lets describe the overall concept of this implementation. The accelerometers will be used to correct drift of the roll and pitch angles, while the magnetometer will be used to correct yaw drifting. Two vectors will be obtained, the first will relate to the error in the roll and pitch and the second will relate to the error in the yaw. These vectors are than multiplied by a weighting factor and ran through a proportional plus integral (PI) feedback controller. The output of the PI feedback controller is the angular velocity correction. This correction is than added to the gyro vector to remove drift.

Now, lets discuss the relation between quaternions and DCM. The DCM elements can be defined based on the quaternion elements. This relation is given below:

$$\vec{q}_b^I = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = q_1\hat{\imath} + q_2\hat{\jmath} + q_3\hat{k} + q_4 \tag{23}$$

$$DCM_b^I = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix} \tag{24}$$

$$DCM_b^I = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_1q_4 + q_2q_3) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{25}$$

So if the DCM or elements, columns, or rows need to be calculated they can be constructed from the quaternions. This will be important and is used in the first part of the drift correction function to construct the 3rd row of the DCM_b^I as follows:

$$\vec{g}_I^b = \begin{bmatrix} R_{zx} \\ R_{zy} \\ R_{zz} \end{bmatrix} = \begin{bmatrix} 2(q_1q_3 - q_2q_4) \\ 2(q_1q_4 + q_2q_3) \\ -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{26}$$

The 3rd row of the DCM_b^I corresponds to the estimated z-axis of the inertial frame expressed in the body frame. Since gravity in the inertial frame is along the z-axis and has a value of 1g, the 3rd row also expresses the estimated gravity vector of the inertial frame expressed in the body frame. This is an important concept, because ideally, if the propagation of the quaternions had no errors, which will be discussed later, the estimated gravity vector in the body frame should be along the accelerometer vector. Due to the errors

that occur in the quaternion update, the gravity vector expressed in the body frame does not line up with the accelerometer vector.

By taking the cross product of the 3rd row of the DCM_b^I with the accelerometer vector, we will get the error between the estimated gravity vector in the body frame and the accelerometer vector. This is because of the relation between the cross product of the two vectors and the angle between them "σ" shown below:

$$errorRollPitch = \vec{a}^b \times \vec{g}^b = \left\|\vec{a}^b\right\| \left\|\vec{g}^b\right\| \sin(\sigma)\, \hat{k}^b \tag{27}$$

Next the yaw drift error needs to be corrected. This correction relies on using the magnetic heading (also known as $\psi_{measured}$) described previously starting on page 9 of the Compass Tab section. Using the magnetic heading, the course over ground can be constructed as follows:

$$\vec{M}^{COG} = \begin{bmatrix} M_x^{COG} \\ M_y^{COG} \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\left(magnetic\ heading\right) \\ \sin\left(magnetic\ heading\right) \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\left(\psi_{measured}\right) \\ \sin\left(\psi_{measured}\right) \\ 0 \end{bmatrix} \tag{28}$$

Eq. 28 gives the coarse over ground of the IMU and to further understand this please read through the Compass Tab section of this paper. For a review the coarse over ground is the rotation of the magnetometer vector into the x-y plane (with no rotation about the z-axis applied). Now if the estimated x-axis of the body frame is expressed in the inertial frame, the cross product can be taken to achieve the yaw error. The x-axis of the body frame expressed in the inertial frame is the 1st column of the DCM_b^I. This 1st column expressed in quaternions is given below:

$$\vec{x}_b^I = \begin{bmatrix} R_{xx} \\ R_{yx} \\ R_{zx} \end{bmatrix} = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 \\ 2(q_1 q_2 + q_3 q_4) \\ 2(q_1 q_3 - q_2 q_4) \end{bmatrix} \tag{29}$$

Next, the z-component of the cross product is:

$$yaw\ error = \left(\vec{x}_b^I \times \vec{M}^{COG}\right)_z = \left((q_1^2 - q_2^2 - q_3^2 + q_4^2) M_y^{COG} - 2(q_1 q_2 + q_3 q_4) M_x^{COG}\right)\hat{k} \tag{30}$$

This yaw error is in the inertial frame but we need it to be expressed in the body frame. So the rotation of the yaw error vector into the body frame is:

$$errorYaw = \|yaw\ error\| \begin{bmatrix} R_{zx} \\ R_{zy} \\ R_{zz} \end{bmatrix} = \|yaw\ error\| \begin{bmatrix} 2(q_1 q_3 - q_2 q_4) \\ 2(q_1 q_4 + q_2 q_3) \\ -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{31}$$

Now that the yaw and roll-pitch errors have been calculated, they can be put through the PI feedback controller to get the angular velocity correction. The setup of the feedback controller is as follows:

$$\omega_P = K_p(Accel\_weight * errorRollPitch + Mag\_weight * errorYaw)$$

$$\omega_{I\_new} = \omega_{I\_old} + K_I \Delta t(Accel\_weight * errorRollPitch + Mag\_weight * errorYaw)$$

$$\omega_{correction} = \omega_P + \omega_{I\_new} \tag{32}$$

The angular velocity correction calculated above is now added to the gyro angular velocity. This is done under the quaternion update of the code discussed on page 14.

I would like to address a couple of the values used above in the feedback control equations given above. The "Accel_weight" is calculated by constraining the max and min values of the following equation to zero and one:

$$Accel\_weight = constrain(1 - 2\left|1 - \left\|\vec{a}^b\right\|\right|, 0,1) \tag{33}$$

The function "constrain" in this case will set any value less than zero to zero and any value greater than one to one. Any weight between zero and one are left as the calculated value. The "Mag_weight" in this code is equal to one, so there is no weighting there. Next is the proportional and integral gain constants that were set to "Kp_ROLLPITCH" and "Ki_ROLLPITCH". These gain values were pre selected in this code and were not altered for the quaternion implementation. Finally, Δt is the time step of the main loop in the code. In the code Δt is called "G_Dt".

Quaternion_update(void):

This quaternion update is what replaced the DCM update previously implemented. The quaternion update is the main part for the attitude estimation. The quaternion update equation is given as:

$$\dot{\vec{q}}(t) = \frac{1}{2}\Omega\big(\vec{\omega}(t)\big)\vec{q}(t) \tag{34}$$

In Eq. 34, $\Omega(\vec{\omega})$ is a 4x4 square matrix and $\vec{q}$ is the quaternion. The definition of $\Omega(\vec{\omega})$ is given below with $[\vec{\omega}\times]$ being a skew symmetric matrix:

$$\Omega(\vec{\omega}) = \begin{bmatrix} -[\vec{\omega}\times] & \vec{\omega} \\ -\vec{\omega}^T & 0 \end{bmatrix} \tag{35}$$

$$[\vec{\omega}\times] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_3 & 0 \end{bmatrix} \tag{36}$$

The errors that occur in the quaternion update equation are due to using a first order numerical integration. In order to solve the quaternion update equation the following discretization is used:

$$\dot{\vec{q}} \approx \frac{\vec{q}(t+\Delta t) - \vec{q}(t)}{\Delta t} = \frac{1}{2}\Omega\big(\vec{\omega}(t)\big)\vec{q}(t) \tag{37}$$

$$\vec{q}(t+\Delta t) = \vec{q}(t) + \frac{1}{2}\Omega\big(\vec{\omega}(t)\big)\vec{q}(t)\Delta t \tag{38}$$

As can be seen above, the smaller the Δt is, the better our numerical integration is. For a smaller Δt less numerical errors occur. This is why it is important to minimize the main loop time to minimize Δt.

The first order numerical integration still leads to errors. These errors can be fixed by checking a property of quaternions. This property is:

$$\vec{q}^T\vec{q} = \vec{q}\cdot\vec{q} = q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{39}$$

Due to the numerical error, the dot product will not be equal exactly to one after the update occurs. This is fixed by renormalizing the quaternion:

$$\vec{q}_{renormalized} = \frac{\vec{q}}{\|\vec{q}\|} \tag{40}$$

From the updated quaternion, the next step is to get the Euler angles and is discussed below.

Euler_angles(void):

To convert the quaternions to Euler angles we need to recall what the DCM is from body to inertial given by Eq. 2 as:

$$DCM_b^I = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

To simplify the calculations we will also recall Eq. 24:

$$DCM_b^I = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix}$$

Now by looking at the above rotation matrix we can see that the Euler angles can be obtained by:

$$-\sin\theta = R_{zx} \rightarrow R_{zx} = -\sin^{-1}(R_{zx}) \tag{41}$$

$$\tan\phi = \frac{\cos\theta\sin\phi}{\cos\theta\cos\phi} = \frac{R_{zy}}{R_{zz}} \rightarrow \phi = \tan^{-1}\left(\frac{R_{zy}}{R_{zz}}\right) \tag{42}$$

$$\tan\psi = \frac{\cos\theta\sin\psi}{\cos\theta\cos\psi} = \frac{R_{yx}}{R_{xx}} \rightarrow \psi = \tan^{-1}\left(\frac{R_{yx}}{R_{xx}}\right) \tag{43}$$

Now that the relation between Euler angles and DCM elements are known, lets recall the relation between DCM and quaternion elements given by Eq. 24 and Eq. 25:

$$DCM_b^I = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix}$$

$$DCM_b^l = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_1q_4 + q_2q_3) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}$$

Now we can see that the conversion from quaternion elements to Euler angles is:

$$-\sin\theta = R_{zx} \rightarrow R_{zx} = -\sin^{-1}(2(q_1q_3 - q_2q_4)) \tag{44}$$

$$\tan\phi = \frac{\cos\theta\sin\phi}{\cos\theta\cos\phi} = \frac{R_{zy}}{R_{zz}} \rightarrow \phi = \tan^{-1}\left(\frac{2(q_1q_4 + q_2q_3)}{-q_1^2 - q_2^2 + q_3^2 + q_4^2}\right) \tag{45}$$

$$\tan\psi = \frac{\cos\theta\sin\psi}{\cos\theta\cos\psi} = \frac{R_{yx}}{R_{xx}} \rightarrow \psi = \tan^{-1}\left(\frac{2(q_1q_2 + q_3q_4)}{q_1^2 - q_2^2 - q_3^2 + q_4^2}\right) \tag{46}$$

DCM Tab wrap up:

For a wrap up of the DCM Tab, recall the main algorithm covered under the topic "Run quaternion algorithm" on page 8. The process was given as follows:

```
Compass_Heading();
Quaternion_update();
Drift_correction();
Euler_angles();
```

The compass heading is called for the first loop to initialize the DCM and quaternions. Every loop after it is used to correct the yaw drift. Next the quaternion update is called on. This progresses the quaternions to the next time step. The drift correction is then run to calculate the angular velocity correction to apply. Finally the Euler angles are calculated for the newly updated quaternions.

Math Tab:

There only two operations that I want to clarify under the Math Tab in the code. The two functions are to initialize the DCM and quaternions and need some extra explanation. The rest of the code in the Math Tab is defining all the needed math operations such as dot products and cross products. The reason to build these basic math operations instead of importing a math library is to conserve space.

init_rotation_matrix(float m[3][3], float yaw, float pitch, float roll) explanation:

This function is used to initialize the DCM. This is done by calculating the initial roll, pitch, and yaw discussed in the Razor_AHRS Tab section starting on page 4. This function takes in the yaw, pitch, and roll angles and constructs the DCM and outputs it under the name given for m[3][3]. Under the Razor_AHRS tab this looks like:

- init_rotation_matrix(DCM_Matrix, yaw, pitch, roll);

Where the yaw, pitch, and roll are the initial Euler angles calculated and the DCM outputted by the function is saved to DCM_Matrix.

The initial DCM is constructed by simply plugging in the initial values of roll, pitch, and yaw into the previously defined rotation matrix from Eq. 2 shown below:

$$DCM_b^I = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

The purpose of getting the initial DCM is to calculate the initial quaternion elements.

Quaternion_init(void) Explanation:

The Quaternion_init function calculates the four initial values of the quaternion. The elements of the quaternions are calculated using a relationship between the Direction Cosine Matrix and quaternion elements. Recall from Eq. 23 the quaternions are given as:

$$\vec{q}_b^I = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = q_1\hat{\imath} + q_2\hat{\jmath} + q_3\hat{k} + q_4$$

The fourth element of this quaternion is the real part and related to the rotation applied while the other three elements are the imaginary parts. The relation of each element with the Direction Cosine Matrix is shown as followed:

First lets recall Eq. 24 and 25:

$$DCM_b^l = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix}$$

$$DCM_b^l = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_1q_4 + q_2q_3) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}$$

Using the above notation it can be shown:

$$q_1 = \frac{R_{zy} - R_{yz}}{4q_4} \tag{47}$$

$$q_2 = \frac{R_{xz} - R_{zx}}{4q_4} \tag{48}$$

$$q_3 = \frac{R_{yx} - R_{xy}}{4q_4} \tag{49}$$

$$q_4 = \frac{1}{2}\sqrt{1 + R_{xx} + R_{yy} + R_{zz}} \tag{50}$$

The initializing of the quaternion values is for propagating them forward in time for attitude estimation, which is discussed starting on page 14.

Position Estimation With Accelerometers:

An attempt was made to get the best possible position estimation from the accelerometers. The issue with using just accelerometers to estimate position is the inherent high frequency noise. This noise causes errors that are amplified as time progresses in the velocity and position of the IMU.

The issues arise from the accelerometer noise when subtracting gravity from the accelerometer to achieve the pure acceleration. The pure acceleration is what's needed to calculate changes in velocity and position. Ideally when the accelerometer is not moving, it should read 0g. Due to the noise this does not happen. To show this lets say we have the accelerometer output $\vec{a}$ in the Inertial frame and the IMU is not moving:

$$\vec{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{1}$$

If the accelerometers did not have noise than the accelerometer in the inertial frame would read 1g along the z-axis. Since it does have the extra noise lets model it as:

$$\vec{a} + \vec{\varepsilon} = \begin{bmatrix} a_x + \varepsilon_x \\ a_y + \varepsilon_y \\ a_z + \varepsilon_z \end{bmatrix} \tag{2}$$

$$\vec{a} + \vec{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ 1 + \varepsilon_z \end{bmatrix} \tag{3}$$

Where ε_x and ε_y are the error from 0g for the x and y-axes and ε_z is the error from the 1g along the z-axis. Now, if the gravity vector is assumed to be constant and is downward along the z-axis of the inertial frame we get:

$$\vec{g} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4}$$

To get the pure acceleration Eq. 4 is subtracted from Eq. 3 as follows:

$$\vec{a}_{pure} = \vec{a} + \vec{\varepsilon} - \vec{g} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ 1 + \varepsilon_z \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{bmatrix} = \vec{\varepsilon}$$

$$\vec{a}_{pure} = \vec{\varepsilon} \tag{5}$$

To obtain velocity Eq. 5 is integrated to obtain:

$$\vec{a}_{pure} = \frac{d\vec{V}}{dt} = \vec{\varepsilon} \rightarrow d\vec{V} = \vec{\varepsilon} dt$$

$$\int_{V_o}^{V_f} d\vec{V} = \int_0^t \vec{\varepsilon} dt$$

$$\vec{V}(t) = \vec{V}_o + \vec{\varepsilon} t \tag{6}$$

Looking at Eq. 6 it can be seen that the error on all three axes grow proportionally with time. Now if Eq. 6 is integrated we obtain the position "P":

$$\vec{V}(t) = \frac{d\vec{P}}{dt} = \vec{V}_o + \vec{\varepsilon} t \rightarrow d\vec{P} = (\vec{V}_o + \vec{\varepsilon} t) dt$$

$$\int_{P_o}^{P_f} d\vec{P} = \int_0^t (\vec{V}_o + \vec{\varepsilon} t) dt$$

$$\vec{P}(t) = \vec{P}_o + \vec{V}_o t + \frac{1}{2} \vec{\varepsilon} t^2 \tag{7}$$

Looking at Eq. 7 it can be seen that the error on all three axes grow quadraticly with time. It was found that with the addition of a low pass filter, for the accelerometer, would help reduce the error but not enough to make the results usable.

For the reasons seen in Eq. 6 and 7, accelerometers are a highly inaccurate way to measure velocity and position. For velocity and position estimation, it would be recommended to integrate a GPS into the system.

Reference Material Used:

"Direction Cosine Matrix IMU: Theory" by William Premerlani and Paul Bizard
http://www.diva-portal.org/smash/get/diva2:450820/FULLTEXT01
http://www.iaeng.org/publication/WCE2011/WCE2011_pp2164-2167