**Lab 5: Calculating Orientation using an AHRS Algorithm**

**Instructor: Prof. Yifeng Zhu**

**Fall 2019**

## Goals:

1. Understanding how the position and orientation of an object can be mathematically represented and calculated using accelerometer and gyroscope data
2. Understanding the role that the AHRS module plays in the STEVAL’s firmware

## Summary of Tasks:

1. Read the contents of the ‘Background’ section
2. Complete the pre-lab assignment
3. Complete the lab assignment
	1. Perform the required tasks using the provided code
	2. Record results and answers in this lab handout

**Background:**

This lab focuses on the firmware modules used by the STEVAL to estimate its current position and orientation. Specifically, this lab covers the MEMS Sensor setup, the AHRS algorithm, and the Euler angle conversion. A block diagram of the STEVAL’s firmware architecture is shown below.



Figure 1. STEVAL Firmware Block Diagram (Source: ST)

The previously mentioned modules are located in section 2 of the above diagram, and, as the image shows, these modules take data from the Quadcopter system to produce output values for roll, pitch, and yaw. These values are used by the PID Control module to determine the motor speeds required for the STEVAL to hover and correctly maneuver in response to the user input received via section 1. As with any digital measurements, the MEMS sensor measurements are affected by noise. To reduce this noise, the inputs from the MEMS sensors are filtered by software before reaching the AHRS module. The PID Control module will be discussed further in Lab 6 and the contents of section 1 will be discussed further in Lab 7. The noise inherent to these sensor measurements, as well as the sensor input filters used to mitigate it, are discussed in Appendix B.

The main focus of this lab will be the AHRS (Attitude and Heading Reference System), which the STEVAL uses to determine its orientation via data readings from the accelerometer and gyroscope, connected over the SPI2 interface.

**Background: Ways to Represent Orientation:**

Before describing the operation of the AHRS, it is important to first understand how the position and orientation of an object can be represented and calculated mathematically. A variety of representations exist, each with their own benefits and drawbacks. The first to be discussed here is the axis-angle representation. Axis-angle, as its name implies, describes an orientation using an axis, represented by the vector , and an angle θ. The vector e is used to determine the object’s ‘pitch’ and ‘yaw’, while the angle θ is used to determine the object’s ‘roll’. A diagram describing the representation is shown below.

Figure 2. Axis Angle Representation (Source: Wikipedia)

As shown in the above image, because the vector e can point in any direction in 3d space, only the additional rotation θ about e is required for the axis-angle representation to be capable of describing every possible orientation in 3d space. The main benefit of this representation lies in its intuitiveness and the ease with which it can describe any given orientation. With axis-angle, however, it is impossible to mathematically perform a rotation to an orientation directly. To solve this problem, an initial axis-angle orientation is often converted to either a rotation matrix or a quaternion.

Quaternions function similarly to the axis-angle representation, and can be interpreted as such, however, the representation itself is comparatively less intuitive, in exchange for better mathematical usability. A quaternion is simply vector of length 4, containing the following elements:

.

where q is the quaternion, is the desired rotation, and describe the axis of rotation.

One major benefit when using this representation is that the final positon of an object after a sequence of rotations can be found simply by interpolating the quaternions representing each successive rotation.

Rotation matrices are yet another way to represent an object’s orientation. The main difference between the previous representations and rotation matrices are that a rotation matrix describes an orientation as a combination of rotations about the x, y, and z axes, while axis-angle and quaternion representations use an arbitrary set of axes. Another important difference to note is that rotation matrices are relatively inefficient for memory usage and computation, as a 3x3 rotation matrix, containing 9 values, is used in all operations, compared to the 4 values used by the other systems. Because of this drawback, rotation matrices are often only used when some information related to the original x, y, and z axes is required. Any orientation can be achieved using the product of three rotation matrices, , which are defined as follows:

These matrices are equivalent to a single rotation about the chosen axis by the angle . The product of these three matrices yields a matrix that represents an object’s orientation. In the event that an equivalent rotation matrix must be generated from a quaternion, the following equation is used.

Finally, the last important representation used by the AHRS module is the Euler angle system. Euler angles are a series of three angles (*ψ*, *θ*, *Ф*) that represent rotation around three axes, in sequence. A variety of Euler angle systems exist, with the difference between each being the order in which the rotations are performed. For the STEVAL, as well as most aerospace applications, a type of Euler angles known as Tait-Bryan angles are used. This system rotates first about the z axis by the angle *ψ,* then about the y axis by *θ,* and finally about the x axis by *Ф.* These rotations are equivalent to an airplane’s yaw, pitch, and roll, in that order. The below images illustrate this rotation.

 

Figure 3. Tait-Bryan Sequence of Rotations (Source: Wikipedia)

In the first image, above, the original axes (x,y,z), in blue, are rotated to produce the new axes (X,Y,Z), after rotating in the Tait-Bryan sequence. The second image shows an equivalent rotation, but with an airplane representing the final position, rather than another set of axes. This system is used to represent orientation in the PID Control module, and therefore must be the final output of the modules discussed in this lab. Until the final output is calculated, the STEVAL operates by using quaternions, requiring the final output quaternion to be converted to a set of Euler angles using the following equations.

where , and *ψ*/*θ*/*Ф* are the yaw/pitch/roll, respectively.

**Background: AHRS Algorithm Flow:**

With this in mind, we can now discuss the operation of the STEVAL’s AHRS module. Code snippets will be used here, and the code in its entirety can be found in Appendix A. An overview of the process is as follows:

* The main function calls the AHRS function to take accelerometer and gyroscope data to produce a quaternion
* Output from the AHRS function is converted from a quaternion to produce equivalent Euler angles
* Euler angles are sent to and used by the PID controller alongside user input to determine the motor speeds required to modify or maintain the drone’s trajectory

Now, the above process will be broken down into various blocks of code, which will be described individually. The AHRS program begins by collecting data from the accelerometer and gyroscope, converting the gyroscope data from mdps to rad/s, and normalizing the accelerometer data.

* Data from accelerometer is collected

 axf = acc->AXIS\_X;

 ayf = acc->AXIS\_Y;

 azf = acc->AXIS\_Z;

* Data from gyroscope is collected and converted from mdps to rad/s

 gxf = ((**float**)gyro->AXIS\_X) \* ((**float**)COE\_MDPS\_TO\_RADPS);

 gyf = ((**float**)gyro->AXIS\_Y) \* ((**float**)COE\_MDPS\_TO\_RADPS);

 gzf = ((**float**)gyro->AXIS\_Z) \* ((**float**)COE\_MDPS\_TO\_RADPS);

* Accelerometer measurement is normalized

 norm = **invSqrt**(axf\*axf+ayf\*ayf+azf\*azf);

 axf = axf \* norm;

 ayf = ayf \* norm;

 azf = azf \* norm;

Next, the direction of gravity relative to the drone’s current positon is estimated. The equations to do this are pulled from the third row of the rotation matrix representing a quaternion, as this third row relates the direction of the world’s z-axis to the drone’s x, y, and z axes.

* The direction of gravity is estimated by calculating the z component of the output quaternion from the last iteration of the AHRS algorithm

 vx = 2\*(q1\*q3 - q0\*q2);

 vy = 2\*(q0\*q1 + q2\*q3);

 vz = q0\*q0 - q1\*q1 - q2\*q2 + q3\*q3;

After the direction of gravity is calculated, a cross product is taken between the acceleration measurement vector and the gravity vector. This produces a vector representing the flux of the linear acceleration.

* Flux is calculated by taking the cross product of the drone’s acceleration and the gravity vector

 ex = (ayf\*vz - azf\*vy);

 ey = (azf\*vx - axf\*vz);

 ez = (axf\*vy - ayf\*vx);

This flux value is used to determine the position error adjustment for the gyroscope measurement, which is calculated by the code below. For more information on the position error adjustment, please see the link in Appendix C.

* The integral portion of the position error adjustment is calculated

 exInt = exInt + ex\*0.1f\*0.00625f;

 eyInt = eyInt + ey\*0.1f\*0.00625f;

 ezInt = ezInt + ez\*0.1f\*0.00625f;

* The integral and proportional position error are added to the gyroscope measurement to compensate for the gyroscope’s finite sampling rate

 gxf = gxf + ahrs\_kp\*ex + exInt;

 gyf = gyf + ahrs\_kp\*ey + eyInt;

 gzf = gzf + ahrs\_kp\*ez + ezInt;

The final lines of code in the AHRS algorithm adjust the position estimation using the velocity values previously calculated. These values are scaled by the sampling time (6.25ms) and adjusted to correctly account for changes in velocity between measurements.

* Output quaternion is updated

 q0 = q0 + (-q1\*gxf - q2\*gyf - q3\*gzf)\*0.5f\*0.00625f;

 q1 = q1 + (q0\*gxf + q2\*gzf - q3\*gyf)\*0.5f\*0.00625f;

 q2 = q2 + (q0\*gyf - q1\*gzf + q3\*gxf)\*0.5f\*0.00625f;

 q3 = q3 + (q0\*gzf + q1\*gyf - q2\*gxf)\*0.5f\*0.00625f;

* New output quaternion is normalized

 norm = **invSqrt**(q0\*q0 + q1\*q1 + q2\*q2 + q3\*q3);

 q0 \*= norm;

 q1 \*= norm;

 q2 \*= norm;

 q3 \*= norm;

Finally, the output quaternion is converted to a set of yaw/pitch/roll values for use by the PID Control module.

* Convert quaternion to Euler Angles

 ea->thx = **atan2**(2\*q0\*q1+2\*q2\*q3, q0\*q0+q3\*q3-q1\*q1-q2\*q2);

 ea->thy = **asin**(2\*q0\*q2-2\*q1\*q3);

 ea->thz = **atan2**(2\*q1\*q2+2\*q0\*q3, q0\*q0+q1\*q1-q2\*q2-q3\*q3);

This algorithm is run consistently as part of the program’s main function, continually updating the position estimation using past estimates alongside new data from the gyroscope and accelerometer.

## Prelab:

Using the information from the previous section, please answer the following questions:

1. You have an orientation described by the following axis-angle values. Convert this to an equivalent quaternion:

a)

b)

c)

2. Convert the following quaternion into a set of Tait-Bryan angles:

a) The quaternion you calculated in 1a)

b)The quaternion you calculated in 1c)

## Lab Assignment:

As the AHRS algorithm is complex, both conceptually and in its implementation, rather than being asked to write code as you have done in previous labs, you will instead work with an existing project. First, download and compile the AHRS code, which can be found in the folder for this lab assignment, then follow the procedures below.

1. The code you have been given simulates the output of the STEVAL’s AHRS module based on various inputs from you, the user. A number of parameters, present at the beginning of the program, control the program’s behavior. First, adjust the gyro vector so that it has the value [1000, 2000, 4000] and adjust the acceleration vector to have the value [0, 0, 0], then compile and run the code. The code should produce outputs for 100 iterations of the AHRS algorithm. Collect and graph the program’s output, using Excel, MATLAB, or another similar program, then record the final results below. This configuration will be used as the baseline for your analysis in the following steps.

eax: eay: eaz:

2. Now, attempt to produce results that differ from those produced in the previous step by modifying the parameters under “part 1” and “part 2”. For at least three different configurations, record the changes you made to the parameters, and what the effects of those changes appear to be.

a)

b)

c)

3. Choose one or more of your previous configurations, and change the values of the noise controllers under “part 3” of the parameter list to 1. Enabling these parameters cause noise equivalent to the typical noise for both the gyroscope and accelerometer measurements to be added to the calculations, imitating a version of data collection without any hardware filters enabled. Rerun the program, and note the changes that occur, if any, in response to the additional noise being added. If changes do occur, is one type of noise more significant than the other? If so, which one?

4. By this point, you have completed all of the required tasks for Lab 5. With your remaining lab time, you are encouraged to continue testing and working with the provided code. Feel free to modify the code in any way you see fit, and try to produce some interesting outputs while further improving your understanding of the AHRS algorithm.

## Appendix A: AHRS Module Code:

Global variables for ahrs function:

**float** q0 = 1, q1 = 0, q2 = 0, q3 = 0;

**float** exInt = 0, eyInt = 0, ezInt = 0;

**extern** int16\_t gTHR;

**float** ahrs\_kp;

ahrs function:

**void** **ahrs\_fusion\_ag**(AxesRaw\_TypeDef\_Float \*acc, AxesRaw\_TypeDef\_Float \*gyro, AHRS\_State\_TypeDef \*ahrs)

{

 **float** axf,ayf,azf,gxf,gyf,gzf;

 **float** norm;

 **float** vx, vy, vz;

 **float** ex, ey, ez;

 **if**(gTHR<MIN\_THR) ahrs\_kp = 10;

 **else** ahrs\_kp = 0.4;

 axf = acc->AXIS\_X;

 ayf = acc->AXIS\_Y;

 azf = acc->AXIS\_Z;

 // mdps convert to rad/s

 gxf = ((**float**)gyro->AXIS\_X) \* ((**float**)COE\_MDPS\_TO\_RADPS);

 gyf = ((**float**)gyro->AXIS\_Y) \* ((**float**)COE\_MDPS\_TO\_RADPS);

 gzf = ((**float**)gyro->AXIS\_Z) \* ((**float**)COE\_MDPS\_TO\_RADPS);

 // normalize the accelerometer measurement

 norm = **invSqrt**(axf\*axf+ayf\*ayf+azf\*azf);

 axf = axf \* norm;

 ayf = ayf \* norm;

 azf = azf \* norm;

 //calculates unit vector for z axis relative direction

 vx = 2\*(q1\*q3 - q0\*q2);

 vy = 2\*(q0\*q1 + q2\*q3);

 vz = q0\*q0 - q1\*q1 - q2\*q2 + q3\*q3;

 //cross product of acceleration and gravity unit vectors to estimate flux

 ex = (ayf\*vz - azf\*vy);

 ey = (azf\*vx - axf\*vz);

 ez = (axf\*vy - ayf\*vx);

 //Int(t) = Int(t-1) + position error \* integral gain \* time period

 exInt = exInt + ex\*0.1f\*0.00625f;

 eyInt = eyInt + ey\*0.1f\*0.00625f;

 ezInt = ezInt + ez\*0.1f\*0.00625f;

 // adjusted gyroscope measurements

 gxf = gxf + ahrs\_kp\*ex + exInt;

 gyf = gyf + ahrs\_kp\*ey + eyInt;

 gzf = gzf + ahrs\_kp\*ez + ezInt;

 //ahrs\_kp = proportional gain, set to either 0.4 or 10 at start of function

 // integrate quaternion

 q0 = q0 + (-q1\*gxf - q2\*gyf - q3\*gzf)\*0.5f\*0.00625f;

 q1 = q1 + (q0\*gxf + q2\*gzf - q3\*gyf)\*0.5f\*0.00625f;

 q2 = q2 + (q0\*gyf - q1\*gzf + q3\*gxf)\*0.5f\*0.00625f;

 q3 = q3 + (q0\*gzf + q1\*gyf - q2\*gxf)\*0.5f\*0.00625f;

 // Normalize quaternion

 norm = **invSqrt**(q0\*q0 + q1\*q1 + q2\*q2 + q3\*q3);

 q0 \*= norm;

 q1 \*= norm;

 q2 \*= norm;

 q3 \*= norm;

 ahrs->q.q0 = q0;

 ahrs->q.q1 = q1;

 ahrs->q.q2 = q2;

 ahrs->q.q3 = q3;

}

Quaternion to Euler conversion function:

**void** **QuaternionToEuler**(QuaternionTypeDef \*qr, EulerAngleTypeDef \*ea)

{

 **float** q0, q1, q2, q3;

 q0 = qr->q0;

 q1 = qr->q1;

 q2 = qr->q2;

 q3 = qr->q3;

 ea->thx = **atan2**(2\*q0\*q1+2\*q2\*q3, q0\*q0+q3\*q3-q1\*q1-q2\*q2);

 ea->thy = **asin**(2\*q0\*q2-2\*q1\*q3);

 ea->thz = **atan2**(2\*q1\*q2+2\*q0\*q3, q0\*q0+q1\*q1-q2\*q2-q3\*q3);

}

## Appendix B: Noise, Error, and Filtering of Input Data:

Multiple sources of error inherently exist within the AHRS algorithm’s calculations. Among these, the two most significant sources are the error caused by the finite sampling rate of the LSM6DSL, and noise within the measurement data itself. The STEVAL accounts for these sources of error in various ways, which will be described below.

As with any sensor, the measurements gathered using the LSM6DSL’s gyroscope and accelerometer contain some amount of noise. Specifically, the gyroscope measurements are typically prone to approximately 75mdps of inaccuracy during normal operation. The accelerometer, on the other hand, can produce significantly different quantities of noise depending on the measurement range that the sensor is configured to have. As the measurement range increases, the amount of noise generated also increases. In the STEVAL’s case, the LSM6DSL’s accelerometer is configured to operate within the range of ±4g, producing a typical noise of approximately 2mg in the process.

The LSM6DSL is able to partially account for the noise produced by these sensors via a variety of configurable hardware filters. The gyroscope is configured by the STEVAL to use a narrow bandwidth low-pass filter, while the accelerometer is configured to use an anti-aliasing analog low-pass filter, a digital low-pass filter, and a composite filter to ensure a clean output. Specifics regarding these filters can be found in the LSM6DSL datasheet, located in Appendix C below.

On top of the hardware filtering performed by the LSM6DSL, the STEVAL also performs software filtering on the gyroscope data as it is read in alongisde the accelerometer data. Specifically, the TIM9 interrupt routine is used to collect data from the accelerometer and gyroscope every 1.25ms, and, during this routine, the incoming gyroscope measurement is modified by a second order IIR filter. This second order IIR filter uses weighted values of the current gyroscopic data as well as the previous two inputs and outputs from the routine to calculate each new output. This filtering is done with the same coefficients for each of the three gyroscopic axes. For completeness, the IIR equation for any given axis, as well as the values of the five filter coefficients, are provided below.

 a1 = 0.94280904158206336, a2 = -0.33333333333333343

b0 = 0.09763107293781749, b1 = 0.19526214587563498, b2 = 0.09763107293781749

These hardware and software input data filters allow for the AHRS algorithm to operate without any significant concern for noise in the initial data set, allowing for other sources of error to be focused on in the algorithm itself.

The finite sampling rate of the LSM6DSL causes error as a result of the equations used by the AHRS algorithm. Many of the operations in the above code are only capable of producing completely accurate results when data is operated on continuously. Since such a condition is impossible to achieve in a real-world environment, the accuracy of these operations is limited by the period over which data is collected.

The STEVAL’s AHRS algorithm takes steps to mitigate the effects of the LSM6DSL’s finite sampling rate by performing linear interpolation on the data it collects during the calculation, to produce more recent estimates than would otherwise be available. This procedure allows for the effective sampling rate of the LSM6DSL to be doubled, reducing error in the AHRS algorithm’s results with only a relatively minor increase to the computational overhead.

## Appendix C: Additional Reading Material:

Wikipedia Links:

<https://en.wikipedia.org/wiki/Rotation_matrix>
<https://en.wikipedia.org/wiki/Euler_angles>

<https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation>

In-depth tutorial on quaternions and Euler angles:

<http://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf>

Additional information on integral gain value:

<https://deltamotion.com/support/webhelp/rmctools/Registers/Register_Descriptions/Axis_Parameter_Registers/Position_Velocity_Control/Param_Integral_Gain.htm>

LSM6DSL Datasheet:
<https://www.st.com/resource/en/datasheet/lsm6dsl.pdf>