The L3GD20 3-Axis Gyro

Table of Contents

1 Overview 1

1.1 Terminology 2

2 Reading and Downloads 3

3 Lab Checklist 3

4 How it Works 3

5 Wire the Gyro 4

6 Install the L3G Arduino Library 4

7 Create the GyroTest Program 5

8 Calibrate the Gyro 6

8.1 Measure Gyro Offset at Rest (Zero-rate Level) 6

8.2 Measure Gyro Noise Level at Rest 7

9 Measure Rotational Velocity 8

10 Measure Angle 9

11 Design Considerations 10

1 Overview

This is the first of two labs that make use of the gyroscope. In this lab you will use the Arduino microcontroller board to read the gyro and process its data into a usable form. In the PID lab you will use your gyro to detect when your rover drifts from a straight trajectory.

The L3GD20 or older L3G4200D is a carrier/breakout board from Pololu which includes a voltage regulator and a high-precision ST L3GD20 (or ST L3G4200D) 3-axis gyroscope. The ST gyro measures the angular rates of rotation (velocity) about the pitch (x), roll (y), and yaw (z)

axes with a configurable sensitivity of ±250°/s, ±500°/s, or ±2000°/s. You will be using the z-axis sensor if your gyro board lies parallel to the ground. This lab assumes use of the z-axis sensor with the board inverted, so you can read the pin labels from above. Otherwise you can determine your axis of rotation, as well as the "positive" direction of rotation, by referring to the axis diagram printed on the gyroscope breakout board.

The ST gyro communicates with our microcontroller over an I2C serial interface. The board includes a 3.3 V linear regulator and integrated level-shifters that allows it to work over an input voltage range of 2.5–5.5 V. The carrier/breakout board uses a standard .1" header allowing it to plug into our solderless breadboard.

As the L3GD20 is an upgrade to the L3G4200D, you may have this older version of the ST gyroscope. If this is the case, you must make three adjustments to this lab:

1. In place of the L3G Arduino library, the L3G4200D library must be added. This library can be found on the Pololu product page for the L3G4200D.
2. In the GyroTest program, the L3G4200D library must be imported in place of the L3G library.
3. The line "while(!gyro.init());" in GyroTest may be deleted or commented out. *At this time including this line with a L3G4200D has not been tested.*

1.1 Terminology

Sensitivity

An angular rate gyroscope is a device that produces a positive-going digital output for counterclockwise rotation around the sensitive axis considered. Sensitivity describes the gain of the sensor and can be determined by applying a defined angular velocity to it. This value changes very little over temperature and time. Note: *Our gyros are mounted upside down on the breadboard so a clockwise rotation generates a positive output.*

Zero-rate level

Zero-rate level describes the actual output signal if there is no angular rate present. The zero-rate level of our precise MEMS (Microelectromechanical system) gyro sensor is, to some extent, a result of stress to the sensor and, therefore, the zero-rate level can slightly change after mounting the sensor onto a printed circuit board or after exposing it to extensive mechanical stress. This value changes very little over temperature and time.

Stability over temperature and time

The single driving mass design of the ST gyroscopes matches the MEMS mechanical mass and the ASIC interface, delivers a high level of stability over temperature and time.

2 Reading and Downloads

1. L3GD20 MEMS motion sensor: ultra-stable three-axis digital output gyroscope
   http://www.pololu.com/catalog/product/2125
2. Arduino library for the L3GD20
   https://github.com/pololu/L3G
3. STMicroelectronics L3GD20 data sheet
4. http://www.st.com/web/catalog/sense_power/FM89/SC1288/PF252443
5. Read about Gyroscopes: http://tom.pycke.be/mav/70/gyroscope-to-roll-pitch-and-yaw and Kalman Filters (optional): http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data

3 Lab Checklist

Here is your checklist for this lab.

- ☐ Learn how to connect and sample Gyro data
- ☐ Measure Gyro noise and drift at rest
- ☐ Compute angle as the Gyro is rotated
- ☐ Explore data filtering techniques

4 How it Works

The purpose of a gyro is to maintain a sense of direction. A gyro does this by sensing changes in its angular motion (angular velocity), and outputting a digital number that is proportional to its rotating speed. This angular velocity can then be integrated over time to give you the offset, in degrees, of the gyro from its zeroed position. In other words, it will tell you the net rotation the gyro is experiencing over time.

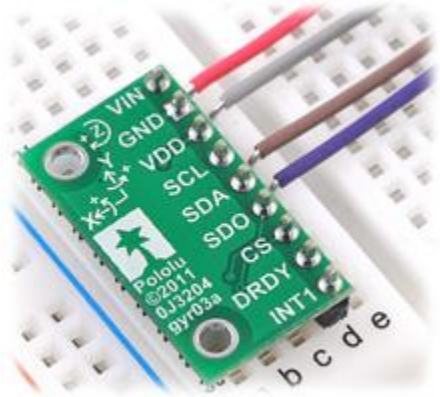
Our gyro can measure three axes referred to as roll (x axis), pitch (y axis), and yaw (z axis). For our rover application, we are only concerned with the yaw (z axis). In terms of the L3GD20, angular velocity is measured in degrees per second. Our gyro returns a positive number for clockwise rotation and a negative number for counterclockwise (note: L3GD20 MIMS chip is inverted)

A parameter of interest for a gyro is the sensitivity, which is found in the sensor's data sheet. It is the parameter linking your DN to your angular velocity (degrees/s).

$$V_{\omega} = \frac{\text{Digital Number}}{\text{Resolution}} \times \text{Sensitivity} \left(\frac{\text{degrees}}{\text{s}}\right) \quad \textbf{EQ 1-1}$$

Our gyro has three sensitivity range settings ±250°/s, ±500°/s, or ±2000°/s. Our rover does not change direction very quickly, so we choose the lowest scale due to its higher level of precision.

5 Wire the Gyro



Install the breakout in the breadboard as shown in the figure above (chip inverted). When the sensor is installed upside down, it will give a positive value for clockwise rotation and a negative value for counterclockwise rotation about the z-axis.

Make the following connections with wires between the Arduino and the L3G4200D:

Arduino Uno/Duemilanove		L3G4200D Carrier
5V	⇨	VIN
GND	⇨	GND
Analog Pin 5	⇨	SCL
Analog Pin 4	⇨	SDA

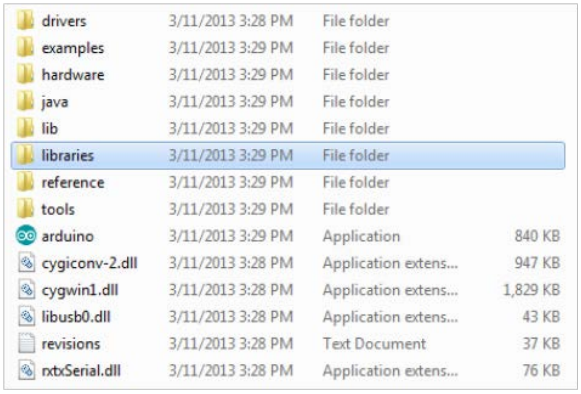
6 Install the L3G Arduino Library

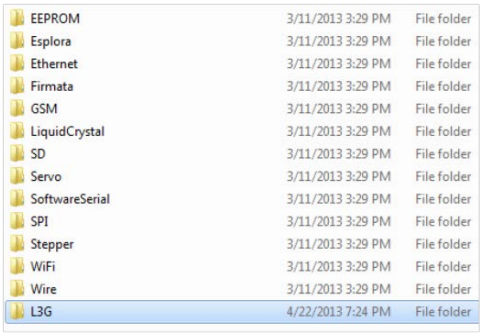
Step 1: Download the library

The L3G library can be found at https://github.com/pololu/L3G.

Note: If you are using the L3G4200D gyro, you must use the L3G4200D library instead.

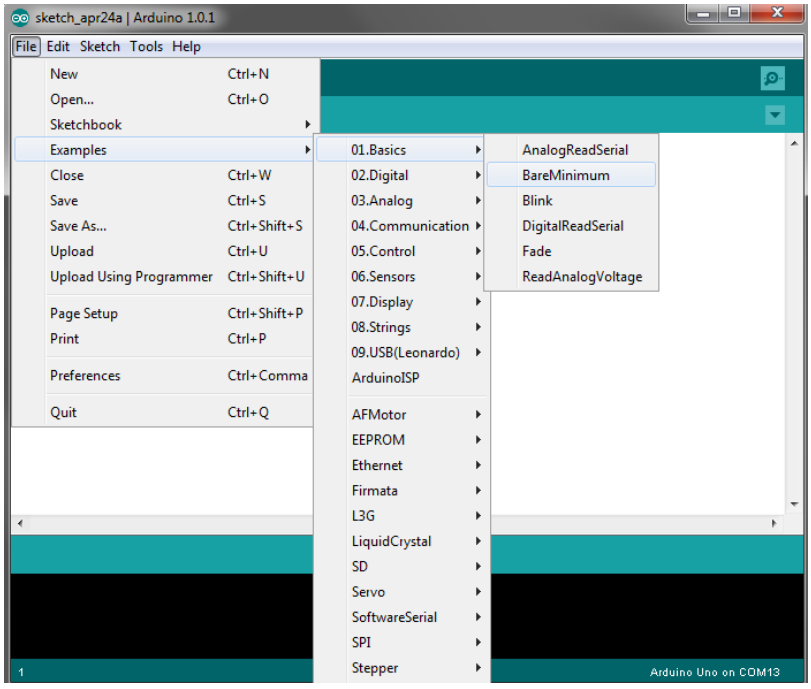
Step 2: Place the whole L3G library folder in your Arduino libraries directory





7 Create the GyroTest Program

Open the Arduino IDE and create a new sketch using the BareMinimum template. This sets up the two loops necessary to all Arduino programs. Save this new sketch under the new name "GyroTest" before adding code.



Add the following code above the "void setup()" block. It imports the L3G gyro and Arduino I2C libraries, and creates an L3G gyro object.

```
#include <Wire.h>
#include <L3G.h>

L3G gyro;
```

Note: L3G4200D users must use

```
#include <L3G4200D.h>
```

instead of

```
#include <L3G.h>
```

Inside the `setup()` block, place the following code. This sets up serial communication for the Arduino, and initializes the gyro sensor.

```
Serial.begin(115200);
Wire.begin();

while (!gyro.init());

gyro.enableDefault();
```

Note: L3G4200D users should delete or comment out the "`while (!gyro.init());`" line.

8 Calibrate the Gyro

Ideally the gyro should output an angular velocity of 0 when it is at rest. In reality the output tends to have a non-zero average value. As we must integrate the angular velocity (degrees/sec) output by the sensor to measure the angle (degrees) of our rover, any DC offset can accumulate and lead to the gyro angle drifting while the rover is sitting still. To fix this problem, we must calculate the zero-rate level of the gyro output on startup and subtract it from subsequent readings.

Noise can also cause inaccurate readings, so it's useful to calculate the noise level of the gyro output at rest. This can help us determine whether or not to set a noise threshold for our sensor readings.

8.1 Measure Gyro Offset at Rest (Zero-rate Level)

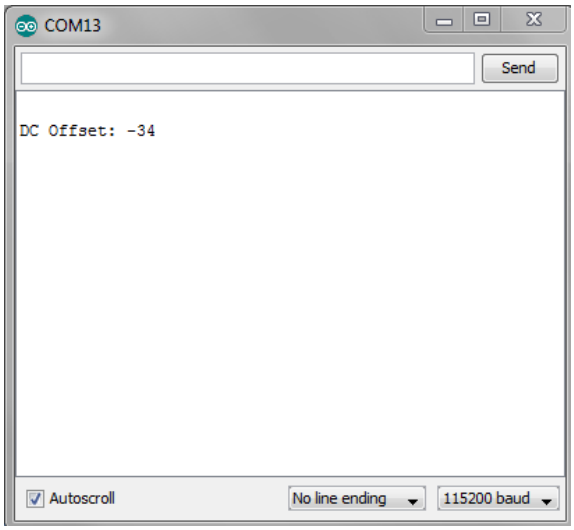
In your GyroTest program, add the following variable declarations above the `setup()` block.

```
int sampleNum=500;
int dc_offset=0;
```

Then, at the bottom of the `setup()` block, add the code to calculate the gyro sensor's DC offset. This code works by taking 500 sample readings and calculating the average value. You can change the number of samples to improve performance: a lower number will reduce calibration time while a larger number will increase precision.

```
//Calculate initial DC offset and noise level of gyro

for(int n=0;n<sampleNum;n++){
  gyro.read();
  dc_offset+=(int)gyro.g.z;
}
dc_offset=dc_offset/sampleNum;

//print dc offset and noise level
Serial.println();
Serial.print("DC Offset: ");
Serial.print(dc_offset);
Serial.println();
```

Now save the program and upload it to your Arduino, making sure to keep the gyro sensor completely still. Check the serial monitor. It should display an integer representing your gyro's current zero-rate level.



8.2 Measure Gyro Noise Level at Rest

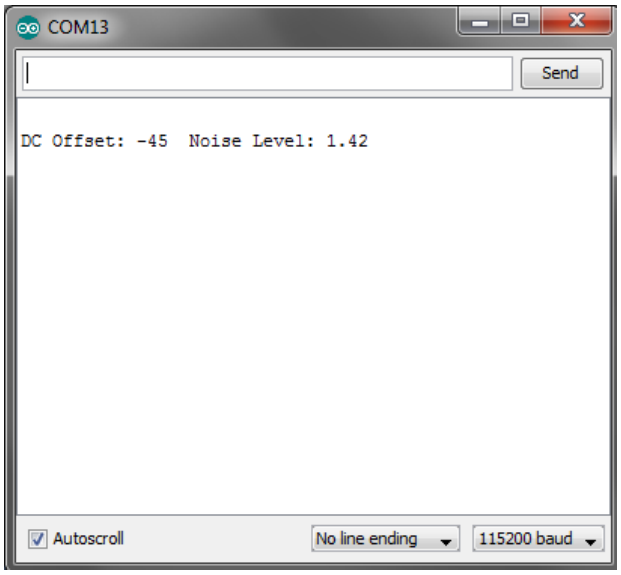
Add the following variable declaration above the `setup()` block.

```
double noise=0;
```

Add the highlighted code to your existing `setup()` block. This code calculates the noise level by finding the reading with the largest absolute value. Notice that these readings take the DC offset obtained in the previous section into account.

```
//Calculate initial DC offset and noise level of gyro

for(int n=0;n<sampleNum;n++){
  gyro.read();
  dc_offset+=(int)gyro.g.z;
}
dc_offset=dc_offset/sampleNum;

for(int n=0;n<sampleNum;n++){
  gyro.read();
  if((int)gyro.g.z-dc_offset>noise)
    noise=(int)gyro.g.z-dc_offset;
  else if((int)gyro.g.z-dc_offset<-noise)
    noise=-(int)gyro.g.z-dc_offset;
}
noise=noise/100; //gyro returns hundredths of degrees/sec

//print dc offset and noise level
Serial.println();
Serial.print("DC Offset: ");
Serial.print(dc_offset);
Serial.print("\tNoise Level: ");
Serial.print(noise);
Serial.println();
```

Now save and run the program again, making sure to keep the gyro sensor completely still, and you should get a reading that includes both the DC offset and noise level. The noise should be around 1 or 2. If it is significantly higher, a noise threshold may be needed for accurate measurement.



9 Measure Rotational Velocity

The ST gyro returns an integer value proportional to the sensor's rate of rotation in degrees/second. To display this data, first add the following variable declarations above the `setup()` block of your GyroTest program.

```
unsigned long time;
int sampleTime=10;
int rate;
```

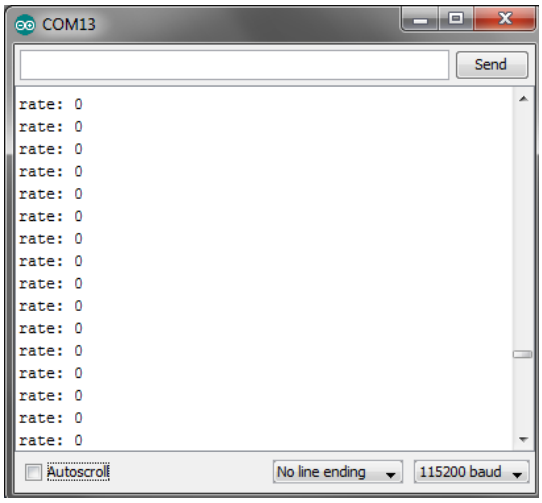
Then, add the following code to the `loop()` block. The `millis()` function returns the number of milliseconds that have passed since the Arduino was reset, so it is used to record time differences between events.

```
// Every 10 ms take a sample from the gyro
if(millis() - time > sampleTime)
{
  time = millis();  // update the time to get the next sample
  gyro.read();
  rate=((int)gyro.g.z-dc_offset)/100;
  Serial.print("rate: ");
  Serial.println(rate);
}
```

Since the velocity readings will quickly take over the whole window, add a time delay at the end of the `setup()` block to give you time to read the DC offset and noise level values.

```
delay(7000);
```

Save and run GyroTest again, and this time you should see a rapid stream of integers corresponding to the rotational velocity of the sensor. Try turning the rover and watching the values change.



10 Measure Angle

To obtain the angular position of the sensor, the rate of rotation returned by the sensor must be integrated with respect to time. A simple and fairly accurate method of numerical integration is the trapezoid method. This algorithm takes the mean average of the current and previous reading, and multiplies the amount of time between those two readings. To integrate the sensor output, first add the following variable declarations above the `setup()` block.

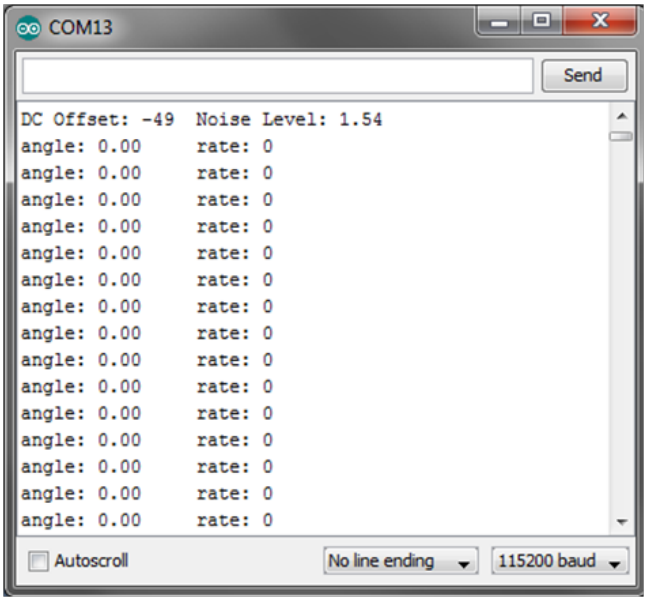
```
int prev_rate=0;
double angle=0;
```

Then, add the highlighted code below to the `loop()` block.

```
void loop() {
  // Every 10 ms take a sample from the gyro
  if(millis() - time > sampleTime)
  {
    time = millis();  // update the time to get the next sample
    gyro.read();
    rate=((int)gyro.g.z-dc_offset)/100;

    angle += ((double)(prev_rate + rate) * sampleTime) / 2000;

    // remember the current speed for the next loop rate integration.
    prev_rate = rate;

    // Keep our angle between 0-359 degrees
    if (angle < 0)
```

```
      angle += 360;
    else if (angle >= 360)
      angle -= 360;

    Serial.print("angle: ");
    Serial.print(angle);
    Serial.print("\trate: ");
    Serial.println(rate);
  }
}
```

To test this program, print out the compass drawing at the end of this document and place it on a solid floor or table, then place your rover on top of it facing zero degrees. Save and upload GyroTest to your rover and open the serial monitor. Once the gyro readings begin appearing in the window, turn the rover and make sure the program measures the same angle as the compass. The output must be divided by an integer value to obtain proper degrees per second. I have used 100, but this number may need to be adjusted to get correct angle readings.



11 Design Considerations

Errors in measurement can be caused by a gradual drift in the DC offset value of the sensor output. This can be corrected by adjusting calibration whenever the rover is motionless using the following code. This has the obvious drawback that the rover must stop to recalibrate during its mission.

```
//Code to correct for gyro drift when rover is motionless
dc_offset = (sampleNum-1)*dc_offset+(1/sampleNum)*(int)gyro.g.z;
```

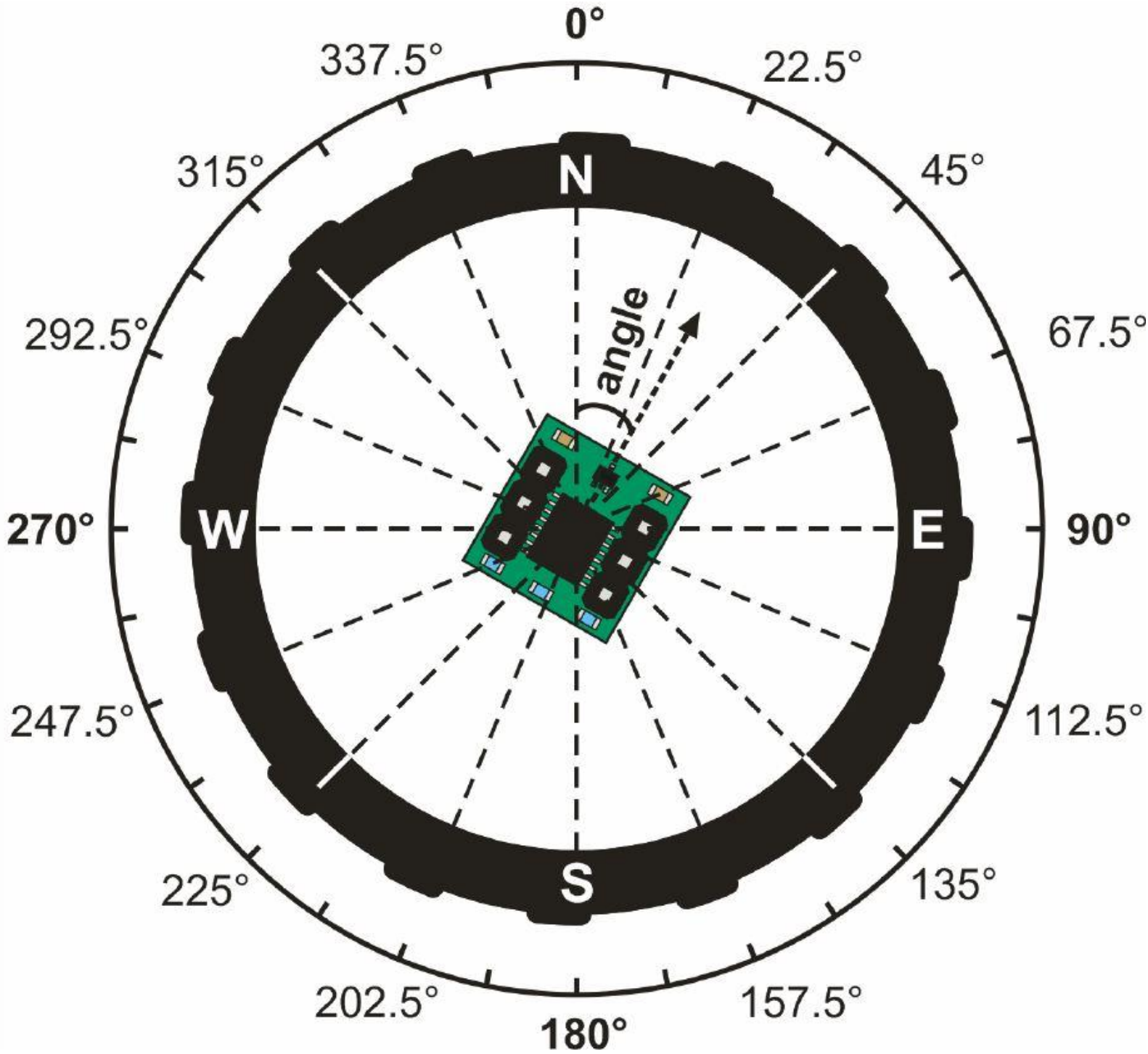
High levels of noise can potentially interfere with gyro operation. If the gyro output tends to oscillate rather than giving steady readings, a noise threshold can be set by adding the following

highlighted code. Note that setting a threshold will cause low-speed rotation to be ignored, which can cause angle readings to become increasingly inaccurate over time.

```
// Ignore the gyro if our angular velocity does not meet our threshold
if(rate >= noise || rate <= -noise)
  angle += ((double)(prev_rate + rate) * sampleTime) / 2000;
```

Also note that since GyroTest uses the polling method to determine when the appropriate sample time has passed, the readings may become inaccurate if this code is added to a program that incorporates large time delays. This can be fixed by setting an interrupt to take a gyro reading every time the appropriate sample time has passed regardless of what the rest of the program is doing.

Appendix: Compass Drawing


0°
337.5°
22.5°
315°
45°
N
angle
292.5°
67.5°
270°
W
E
90°
247.5°
112.5°
S
225°
135°
202.5°
157.5°
180°