**DUE: At beginning of lab 8 session**

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Names: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| LABORATORY CHECK OFFS | |
| **Section 1**  Serial plotter shows gyro rotation around X, Y, Z axis.  **Section 2**  Convert from millis() to Timer 1 overflow interrupt  TCCR1A= \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (HEX)  TCCR1B= \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (HEX)  TIMSK1= \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (HEX)  ICR1= \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (decimal)  Sensor plots correctly on Serial plotter, and scope traces show SDA and SCL lines  **Section 3**  Time for Timer 1 OVF = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ microseconds  Time for I2C communications= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ microseconds  Time without prints = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ microseconds  Time for main loop = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ microseconds  **Section 4**  Lights and motor motion change in response to gyro sensor.  **Section 5**  Yaw heading is printed and correct for 90 degrees turns in either direction. | \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_ |
| Points | \_\_\_\_ |

Prelab:

Complete the separate prelab that involves soldering the gyro sensor board.

Learning Outcomes:

By the end of this lab you will be able to:

1. Configure Timer 0 to create fast PWM without affecting the operation of the millis() timer.
2. Create an interrupt service routine for Timer 1 overflow.
3. Interface to a gyroscopic sensor to the ATMEL 328 using I2C serial communication.
4. Use a gyroscopic sensor to sense the heading of the robot.
5. Combine multiple functions into one program.

Overview:

At the end of this lab, you will have built a gesture-controlled motor system. Cool!

In this lab, you will be combining elements from previous labs (PWM and the RGB led) with a new sensor element (gyroscope) to create a system that drives a motor forward and backward. You will using your existing skills as practice for the next competency exam, and you will be learning new skills working with serial communication (I2C protocol or Two Wire interface) to interface to a sensor.

References:

Gyro sensor code: <http://playground.arduino.cc/Main/MPU-6050>

Deep dive into gyros: ST micro everything about gyros.pdf on MyCourses

Motor driver: <https://arduinoinfo.mywikis.net/wiki/MotorDrivers>

How to Succeed With This Lab:

Follow the wiring color code exactly as shown in the pictures. Your lab instructor will ask you to re-wire anything that doesn’t match the color code.

Check the register functions, names, and settings by reading the datasheet for the Atmel 328 microcontroller.

Troubleshooting Tips:

1) It is very helpful to create small, stand alone pieces of code that test an individual function e.g. turn the motor, or light the led. If you find that the whole system isn’t working, go back to these pieces of test code and verify the individual component still works.

*Example: My RGB led that had worked fine for weeks all of a sudden would not produce green. I didn’t know if it was a code issue, a wiring issue or a component issue. I went back to my test code for the RGB led and found that just the GREEN LED component had failed. If I hadn’t kept that standalone test code I could have spend hours assuming it was some code interaction issue caused by adding the I2C code.*

2) As your hardware design gets more complex with several sensors and motors, it is very helpful to follow a consistent color coding scheme for your wiring. E.g. +5v = red, ground = black or brown, etc.

**SECTION 1 – Interfacing to a gyroscope sensor using I2C serial communication**

I2C is serial communications protocol that allows microcontrollers to interface with other peripheral devices such as external memory (e.g. EEPROM), sensors (e.g. accelerometers or gyroscopes) or other microcontrollers.

The detailed operation of I2C will be covered in lecture. In this part of the lab, you will use a library of I2C serial communication functions to interface with a gyroscope sensor (InvenSense model MPU-6050). This device’s datasheet and a document describing its registers are posted on MyCourses.

What does a gyroscope sensor do?

A gyroscope sensor measures the rate of rotation about an axis. It doesn’t directly measure the angle of rotation but instead it measures the rate of rotation in degrees per second (dps or deg/sec).

In contrast to a gyroscope, an accelerometer measures the rate of acceleration (meters/sec/sec) in a straight line. Note the different units (per second squared in the denominator for accelerometer) and measurement of distance (m) instead of rotation (degrees).

Remember, a gyroscope measures rotational speed while an accelerometer measures linear acceleration (not linear speed). In this lab, only the gyroscope function of the MPU-6050 sensor will be used.

From the MPU-6050 datasheet:

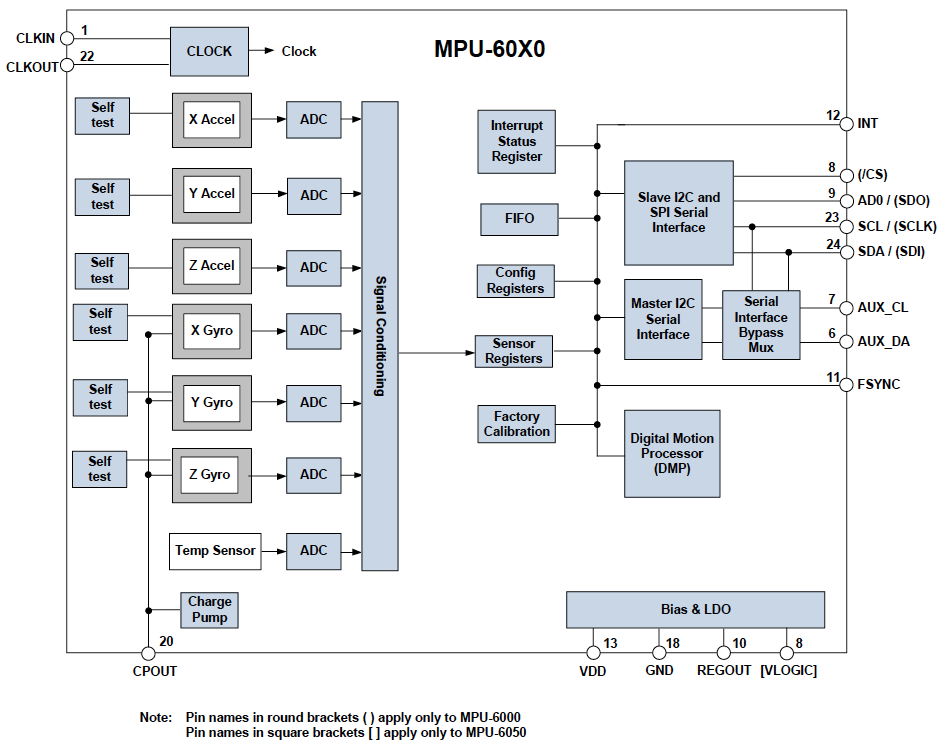
*The device combines a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor™ (DMP™) capable of processing complex 9-axis sensor fusion algorithms.* (We won’t be using the DMP functions).

*The MPU-60X0 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps) and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g.*

*Communication with all registers of the device is performed using I2C at 400kHz.*

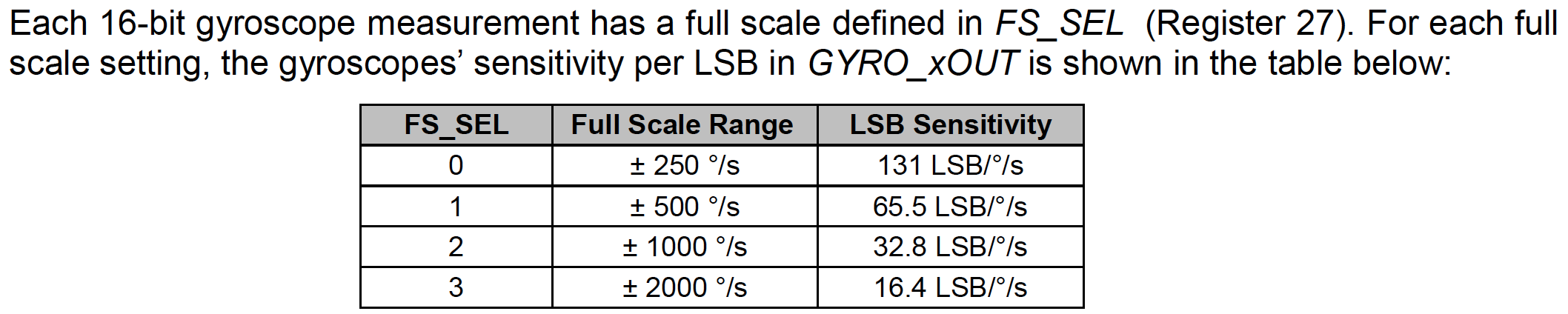
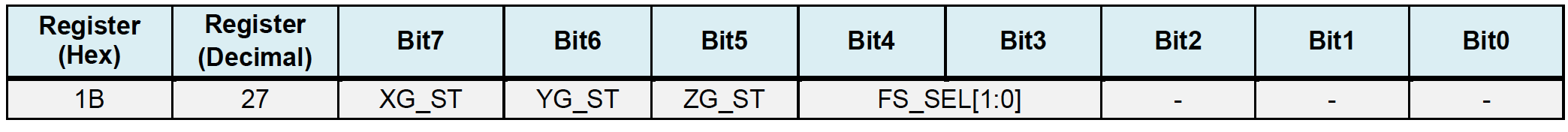
What is the block diagram of the gyroscope sensor?

The block diagram of the MPU-6050 is shown below. Notice in the lower left corner the Analog to Digital Converters for the Gyro sensors X,Y and Z axes. On the right hand side, note the serial data (SDA) and serial clock lines (SCL) that are used for serial communication. Inside the block diagram note the Config Registers and Sensor Registers that can be accessed by the internal serial bus.

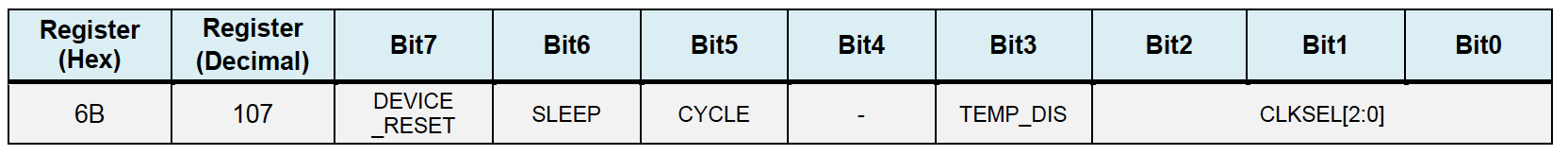
**

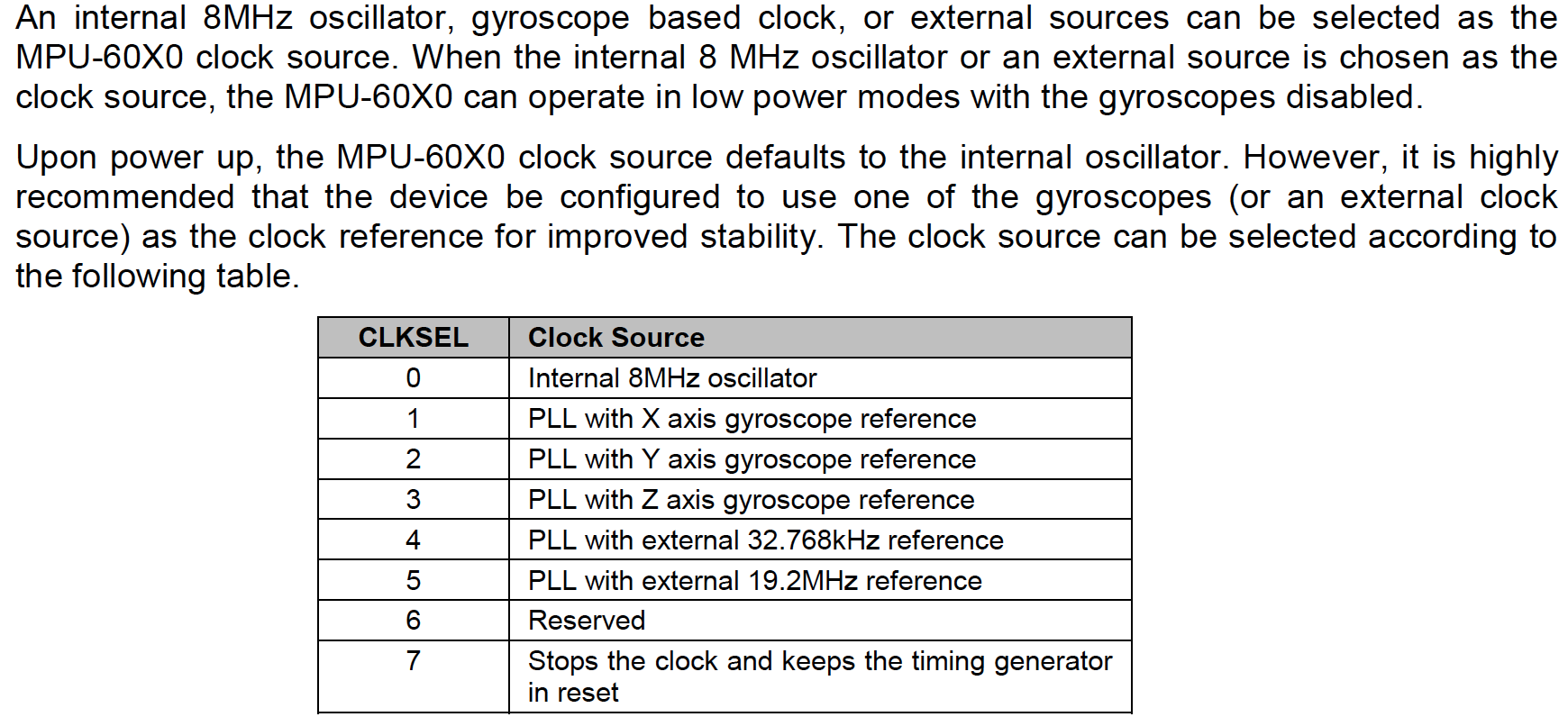
How is the gyroscope configured?

To configure the gyroscope, the full scale range is set to 2000 degrees per second using bits 3 and 4 in register 27.



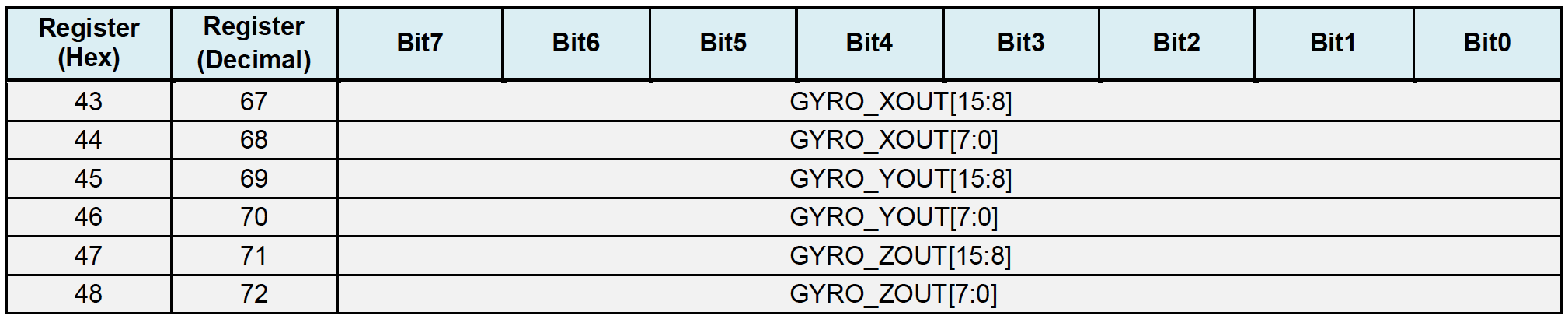
Next, the clock source for the MPU-6050 is set using the three clock select bits in register 107, as shown below. For this lab, the clock source will be set to PLL with X gyro, as that will give an accurate, stable clock.

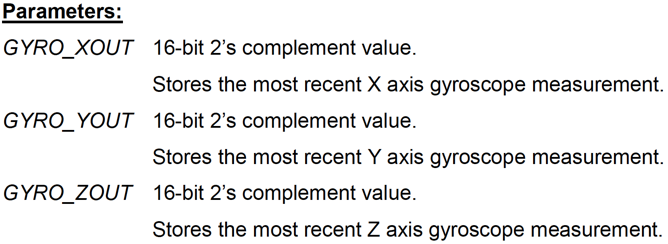




How is data read from the gyroscope?

Below is a list of the registers for the gyroscope outputs along three axes (XYZ). Note that each register below is 8 bits. The gyroscope’s output is 16 bits (for more precision). The 16 bits are stored in two different 8-bit registers. The Zaxis 16 bit value’s high byte is stored in register 71, the lower byte is stored in register 72. See the last two lines of the table below.

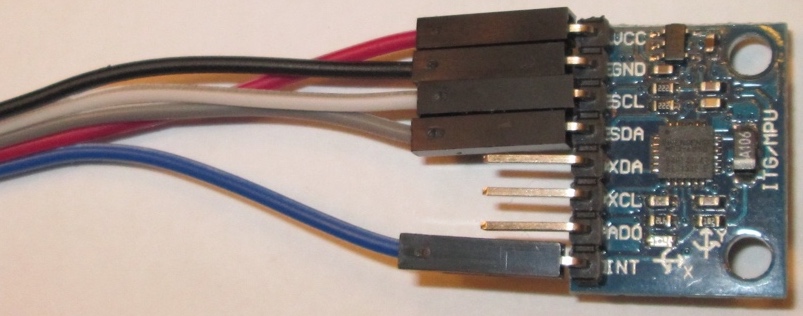
**



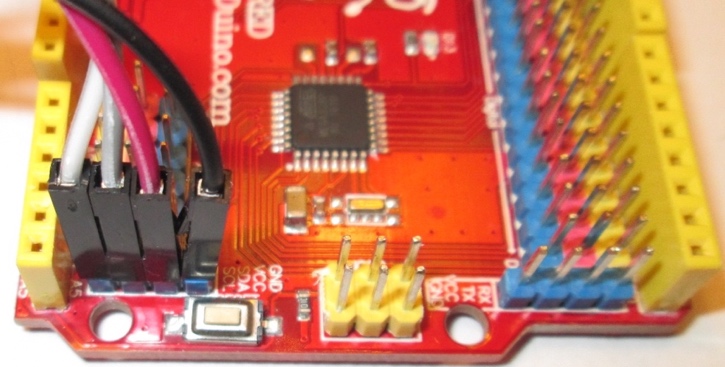
All of these registers will be set using an existing library of code for the MPU-6050.

**Procedure:**

1. Wire the sensor board as shown below. Connect red to VCC, black to ground, white to SCL, grey to SDA, and blue to INT.



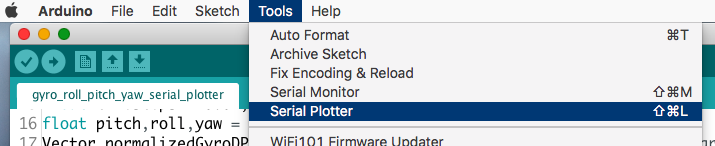
1. On the Arduino end of the sensor, there is a dedicated set of header pins for I2C communications. This header has VCC, GND, SDA, and SCL. For now, the blue wire will not be connected at the Arduino board.



1. With the sensor connected and the motor disconnected, Upload and run the test code in the box below.

**Note**: the gyro code uses a library for the MPU-6050 which is already installed on the lab computers. If you use your personal machine, you will have to install the library yourself. Download a zip file of the library from <https://github.com/jarzebski/Arduino-MPU6050> Place this file in your /Arduino/libraries folder and relaunch the Arduino IDE.

1. Open the serial monitor and in the lower right corner, set the baud rate to 115200 from its original value of 9600. The serial monitor should show three columns of zero values. As you move the sensor, around the number values should change. The values printed are the angles of rotation in degrees about the X,Y, and Z axes, where the zero value is the initial orientation of the sensor during calibration. Play around with rotating the sensor and observe the values.
2. Next, close the serial monitor window. Go the Arduino tools menu and instead of opening the serial monitor, open the serial plotter tool as shown in the figure below.



1. Observe the plotted values of rotation angles as you rotate the sensor around one axis (X, Y, Z) at a time. Show this to your lab instructor and **obtain a signoff.**

/\* Lab7\_I2C\_gyro\_serial\_plotter.ino

Written by: Clark Hochgraf, revised: Feb 18, 2019

Description: ++ PLEASE READ ALL COMMENTS ++

Demonstrates calculation of roll, pitch and yaw angles using a gyroscope sensor.

The gyro outputs the rate of rotation in degrees per second.

The gyro signal is the integrated (multipled by sample interval) to get degrees.

Hardware: uses the MPU-6050 6-axis accelerometer and gyro

Software: uses library for MPU6050, on personal machines, you will have to install

\*/

#include <Wire.h>

#include <MPU6050.h>

MPU6050 mpu; // declare a variable called mpu of datatype MPU6050

unsigned long timeStampStartOfLoopMs = 0;

float timeStepS = 0.01;

float pitch,roll,yaw = 0.0f; // pitch, roll and yaw values

Vector normalizedGyroDPS; //stores the three gyroscope readings XYZ in degrees per second (DPS)

//==============================================================================

void setup() {

Serial.begin(115200);

// Initialize MPU6050 to have full scale range of 2000 degrees per second

while(!mpu.begin(MPU6050\_SCALE\_2000DPS, MPU6050\_RANGE\_2G)) {

Serial.println("Could not find a valid MPU6050 sensor, check wiring.");

delay(1000);

}

mpu.calibrateGyro(); // Calibrate gyroscope- must be done with sensor not moving.

mpu.setThreshold(1);// sets level below which changes in gyro readings are ignored.

// helps to reduce noise. 1 = one standard deviation. Range is 0 to 3.

} // setup

//==============================================================================

void loop() {

timeStampStartOfLoopMs = millis(); // mark the time

normalizedGyroDPS = mpu.readNormalizeGyro(); // Read normalized values

// Calculate Pitch, Roll and Yaw

pitch = pitch + normalizedGyroDPS.YAxis \* timeStepS;

roll = roll + normalizedGyroDPS.XAxis \* timeStepS;

yaw = yaw + normalizedGyroDPS.ZAxis \* timeStepS;

Serial.print(pitch);

Serial.print(" ");

Serial.print(roll);

Serial.print(" ");

Serial.println(yaw);

// Wait until a full timeStepS has passed before next reading

delay((timeStepS\*1000) - (millis() - timeStampStartOfLoopMs));

} //loop

**SECTION 2 – Timer 1 Overflow Interrupt to collect data at regular intervals**

In the sample code from section 3, the angles of rotational are computed by integrating the rotational speed. Each new sample of rotational speed is multiplied by the timestep and added to a running sum (e.g. pitch):

pitch = pitch + normalizedGyroDPS.YAxis \* timeStepS;

For this calculation to be accurate, each new sample of rotational speed must be collected at precisely 10 millisecond intervals.

In this section, you will setup an interrupt on Timer 1 that fires every 10 milliseconds and collects a new gyro reading.

**Procedure:**

1. Replace the loop() code with new loop() code below.

void loop() {

while (!newDataFlag) {}; // stay stuck here until new data arrives, then run loop

// this will occur every 10 millisecond

// elapsedMicroseconds=micros()-startMicroseconds; // check time for Timer 1 OVF

startMicroseconds=micros(); // mark time at start of main loop code

normalizedGyroDPS = mpu.readNormalizeGyro();

// elapsedMicroseconds=micros()-startMicroseconds; // check time for I2C comms

// Calculate Pitch, Roll and Yaw

pitch = pitch + normalizedGyroDPS.YAxis \* timeStepS;

roll = roll + normalizedGyroDPS.XAxis \* timeStepS;

yaw = yaw + normalizedGyroDPS.ZAxis \* timeStepS;

// elapsedMicroseconds=micros()-startMicroseconds; // check time without prints

Serial.print(pitch);

Serial.print(F(" "));

Serial.print(roll);

Serial.print(F(" "));

Serial.println(yaw);

// elapsedMicroseconds=micros()-startMicroseconds; // check total time main loop

// Serial.print(F("elapsed time in microseconds = "));

// Serial.println(elapsedMicroseconds);

newDataFlag=false;

} //loop

1. Add new variable declarations as follows:

volatile bool newDataFlag=false; // boolean flag to indicate that timer1 overflow has occurred

unsigned long startMicroseconds,elapsedMicroseconds; // use for profiling time for certain tasks

1. Configure Timer 1 as follows
   1. fast PWM mode 14.
   2. Clock prescale of 256
   3. Enable the Timer1 Overflow Interrupt (TOIE1) in TIMSK1
   4. Turn off the compare output module (COM1A and COM1B)
   5. Set ICR1 = \_\_\_\_\_ ??? to get an interrupt every 10 milliseconds.
2. Add an interrupt service routine (ISR) named “TIMER1\_OVF\_vect”. Inside the interrupt add the following code

{newDataFlag=true;}

1. Upload the code and use the serial plotter to verify that the gyro sensor works as before.
2. Connect the oscilloscope to show the serial data SDA and serial clock SCL lines. Set the total time range to capture at least 9 clock cycles and preferably a start sequence.
3. Show your scope traces and code with Timer 1 configuration information to the lab instructor and **obtain a signoff.**

**SECTION 3 – Measuring Code Execution Time (Time Profiling)**

In this section, you will profile the time it takes to execute each section of code. Depending on how much data is read from the sensor, communication with the sensor over I2C can take a relatively long time. In general, interrupt routines should be as short as possible and run very quickly. By measuring execution time, it will give you some idea if a routine can be placed in an interrupt.

To measure the code execution time, uncomment, only one at a time, the statements where elapsedMicroseconds is calculated.

// elapsedMicroseconds=micros()-startMicroseconds; // check time for Timer 1 OVF

**Procedure:**

1. Uncomment out the elapsedMicroseconds line for checking Timer 1 OVF. Upload and run the code and record the number of microseconds to execute the code on the signoff sheet. You will need to uncomment the Serial.print() lines at the end of the main loop where the elapsedMicroseconds is printed.
2. Do the same, one by one, for the time for
   1. I2C communications
   2. time without print
   3. total time of main loop
3. **Obtain a signoff** once you have all the values.

**SECTION 4 – Providing visual feedback**

In this section, you will add visual feedback to your system using the RGB led to show the angle of rotation about the Z axis. You will also drive the motor in response to the gyro sensor.

Specifications:

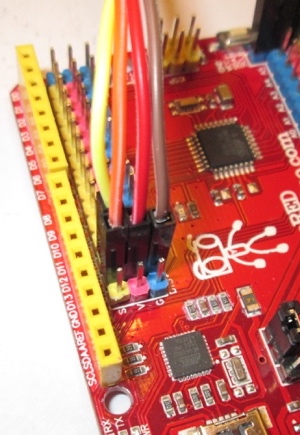
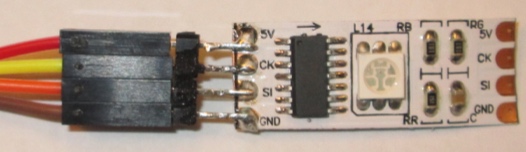
1) When sensor is not rotated from its initial position, the RGB led will show a blue color. The motor will not turn.

2) When sensor is rotated clockwise by more than 5 degrees from its initial position, the RGB led will show a red color. The motor will turn one direction.

3) When sensor is rotated counterclockwise by more than 5 degrees from its initial position, the RGB led will show a green color. The motor will turn the opposite direction.

**Procedure:**

1. Wire your RGB led using the color code shown below and connect it to the Arduino as shown in the second figure. Orange for SI on pin 12, Yellow for CK on pin 11, Red for +5, and Brown for ground.



1. Start with your code from the previous section.
2. Add # define’s for the following constants
   1. LED\_CLOCK\_PIN 11
   2. LED\_DATA\_PIN 12
   3. DIMBLUE\_COLOR 0x00001F
   4. DIMRED\_COLOR 0x1F0000
   5. DIMGREEN\_COLOR 0x001F00
3. Configure pins 11 and 12 as outputs and set them to LOW.
4. Look through your old code or lab 3 handout and paste the complete function definition for **display\_color\_on\_RGB\_led()** into your code. You will use this function to set the LED color.
5. Insert the following **bolded** code into your main loop() program towards the end, and right before newDataFlag=false;

void loop() {

...

// Serial.println(elapsedMicroseconds);

**display\_color\_on\_RGB\_led(DIMBLUE\_COLOR); // default is dim blue color**

**OCR0A=0; // set motor speed off**

**if (yaw>5) {**

**display\_color\_on\_RGB\_led(DIMRED\_COLOR);**

**OCR0A=80; // set motor speed slow**

**digitalWrite(10, LOW); //motor direction pins**

**digitalWrite(9, HIGH); //motor direction pins**

**}**

**if (yaw<-5) {**

**display\_color\_on\_RGB\_led(DIMGREEN\_COLOR);**

**OCR0A=80; // set motor speed slow**

**digitalWrite(10, HIGH); //motor direction pins**

**digitalWrite(9, LOW); //motor direction pins**

**}**

newDataFlag=false;

} //loop

1. Upload and run the code and verify that the LED changes color as expected when the sensor is rotated in the flat plane of the sensor board.
2. Now that the LED light is working, next you will add control of the motor. Connect the motor driver as before in lab 6. Remove the wheels from your robot platform so that it does not move.
3. Grab your Timer0 configuration code from section 2 and paste it into the setup() section. Make sure to include the code for setting pin OC0A as an output.
4. Upload and run your code. Connect the battery pack and verify that the motor turns when the sensor is rotated.

**Congratulations – you have built a gesture controlled motor driver system with visual feedback. Obtain a signoff** for section 4 from your lab instructor.

**SECTION 5 – Using Gyro Sensor Feedback**

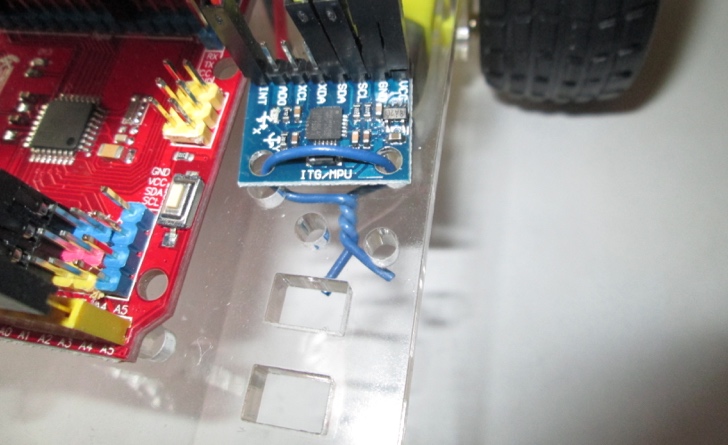
In lab 6, you probably found it difficult to get the robot to make precise 90 degree turns. It would be easier to get exact 90 degree turns if the robot knew what heading it was going before and after the turn. This can be accomplished by using the gyroscope sensor to determine the yaw angle of the robot. Recall that the yaw angle is the rotation about a vertical axis. In other words, yaw gives the heading of the robot on the 2-D plane of the floor.

In this section, you will add in the gyroscope sensor code from section 2 and use it to print out the yaw angle. Then you will manually (by hand) turn the robot 90 degrees while watching the yaw angle and observe what the printed yaw value does for clockwise and counter clockwise turns.

**Procedure:**

1. Mount the gyro sensor flat on the robot base. Use the stiff blue wire to hold the **sensor down tightly so that it doesn’t move.**

The ideal location for the sensor is centered between the wheels, but the location shown is acceptable. If you want to mount the sensor between the wheels, look for a location on the bottom side of the board.



1. Starting with your existing code from section 5 of lab 6, add in the gyroscope sensor (both hardware and software). Make any changes needed to get the code to work.
2. Print out only the value of the yaw reading.” Yaw heading is: “ e.g. 85.2
3. Manually turn the robot base to the right and to the left by exactly 90 degrees and notice if the yaw reading is accurate.
4. Demonstrate the operation to your lab instructor and **obtain a signoff**. You may need to work on this outside of class to get everything working. If so, bring the full working system to lab next week for a signoff.