Sensor Fusion

Due Date: Friday, November 17

Resources

THIS LAB REQUIRES READING DOCUMENTATION AND DATASHEETS. Before you ask an officer a question, make sure your answer is not in the slides, this document, the datasheet, or the documentation for the library.

Lecture 1 Slides: **Link**
Lecture 2 Slides: **Link**
Sensor Fusion Library: **Link**
MPU6050 Datasheet: **Link**
MPU6050 Register Map: **Link**
Arduino I^2C Documentation: **Link**
Arduino Pro Mini Pinout: **Link**
Orientation Visualizer: **Windows**, **macOS**

Overview

In this lab, you will be using accelerometer and gyroscope data to obtain a more accurate representation of orientation.

This lab is designed to teach you read raw values from the MPU6050 IMU, calculate the appropriate offsets, and implement a complementary filter to combine the short-term accuracy of the gyroscope and the long-term accuracy of the accelerometer.

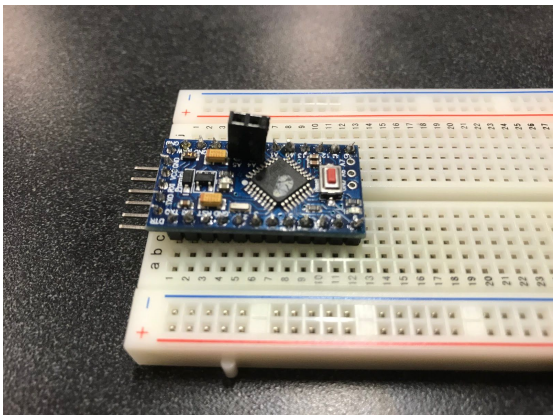
Lab Outcomes and Expectations

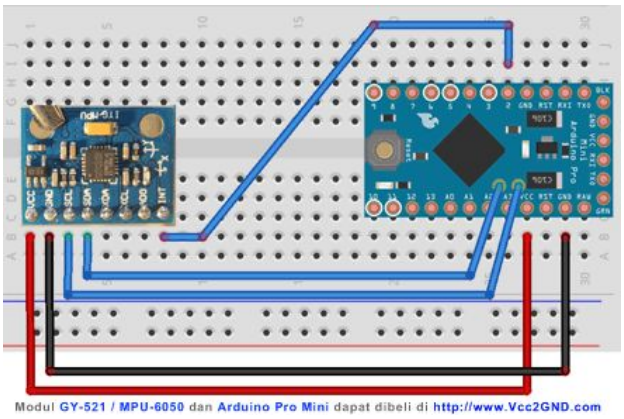
By the end of this lab, you should be familiar with how data is transmitted using I^2C, and how to perform basic sensor fusion using an accelerometer and a gyroscope. In order to be "checked off" for this lab, you must complete each checkpoint. Every team member is expected to have contributed to the lab, and may be quizzed on any of the information covered.

A. Assembling the Hardware

The Arduino has two pins - A4 and A5 - that are for the SDA and SCL lines respectively. Solder female headers to these pins so they can be easily accessed from above. SDA and SCL on the IMU should be connected to A4 and A5 on the Arduino respectively, and VCC and GND on the IMU should be connected to their respective pins on the Arduino. The other four pins should be left disconnected. You do not need the interrupt pin.

NOTE: Use the straight headers for the IMU, not the angled headers.





B. Writing Register Functions

You will need to write implementations for the readReg and writeReg functions in the **sensor_fusion** library. Refer to the **I²C documentation**, especially the beginTransmission, endTransmission, write, read, and requestFrom functions. Also refer to the "I²C Communications Protocol" section of the **datasheet** to see the format for register reads and writes on the IMU. If you need a refresher on I²C, refer to the **Lecture 1 slides**.

After writing the register functions, you must use them to configure the device. Set the PWR_MGMT_1 register to take the IMU out of sleep mode, the GYRO_CONFIG register to the largest possible full-scale range to enable the detection of high-velocity rotations, and the CONFIG register to the largest possible bandwidth. The **register map** contains all the necessary bit information.

Checkpoint 1: Set the appropriate registers and confirm that the outputs of those registers are the intended values.

C. Bias Compensation

The next step in using an IMU is to calibrate for the bias error, as mentioned in the lecture. Collect several (50 to 100) samples from the IMU while it is at rest on a flat surface, and use these to calculate the bias errors for both the accelerometer and the gyroscope. Remember that the accelerometer should read 1g upwards when at rest and the gyroscope should read nothing. In order to read data from the IMU, you must first check the INT_STATUS register to see if data is ready, as you only want to process each sample from the IMU once. If data is ready, you can read registers 59 through 64 to get the accelerometer data, and registers 67 through 72 to get the gyroscope data. Notice that for each direction (x, y, and z) there are two registers, which you must combine into a single 16-bit value.

D. Accelerometer Data

As mentioned in lecture, the simplest method for tracking orientation is using the accelerometer. Take the raw accelerometer readings from IMU and turn them into a unit vector that points in the direction of "true up" with respect to your IMU's internal reference frame (use the **sensor_fusion** library to help you perform the calculations). Remember to account for the bias you found in part C. Once you have this unit vector, print out its three components on a single line separated by spaces, and use our visualizer application (**Windows**, **macOS**) to see your basic orientation tracking. Perform these calculations in your loop function so the visualization continuously updates.

NOTE: Set the Serial baud rate to 115200 when using the visualizer. After opening the visualizer, you will see a list of serial ports in the lower left corner. Click the button for the serial port your Arduino is connected to, even if that button is already highlighted.

Checkpoint 2: Read unbiased data from the accelerometer and gyroscope, and use the visualizer to display your accelerometer-based orientation tracking.

E. Gyroscope Data

Next you will use the gyroscope data to keep track of “true up”: assume your device begins sitting on a flat surface, and use the the rotation data from the gyroscope to update the orientation vector. In order to perform the rotations, you can use our **sensor_fusion** library. When using our library, you must transform the gyroscope rotation data into a **quaternion**. A quaternion is a four-element vector representing a rotation: a rotation of angle θ around the unit vector $< x,\ y,\ z >$ becomes the the quaternion $< \cos(\frac{\theta}{2}),\ x\sin(\frac{\theta}{2}),\ y\sin(\frac{\theta}{2}),\ z\sin(\frac{\theta}{2}) >$. Quaternions allow us to perform rotations efficiently, and are used heavily when performing orientation tracking or other calculations that rely on rotations. Once you have your gyroscope-based orientation tracking working, print out the three components of your normal vector on the same line as you are printing out the accelerometer-based normal vector. This will allow you to compare your two orientation tracking methods and contrast their differences.

Checkpoint 3: Use the visualizer to compare your accelerometer-based orientation tracking and gyroscope-based orientation tracking.

F. Complementary Filter

The final objective of this lab is to implement a complete complementary filter factoring in both accelerometer and gyroscope data. Using your code from parts D and E, our library, and the equation for the complementary filter, implement a three-dimensional complementary filter. You will have to find your own weight α through trial and error. To test your filter, print out the three components of your filtered normal vector on the same line as your accelerometer-based and gyroscope-based vectors, so that all three will show up in the visualizer.

Checkpoint 4: Use the visualizer to compare all three of your orientation tracking algorithms.

G. Summary

After finishing the lab, you should have completed the following:

---

1. Finished each checkpoint of the lab and were checked off by an Advanced Project lead for each. Your entire team must be present and understand how the code works.
2. Gained a solid understanding of how to apply I^2C.

3. Created an accurate sensor fusion model for orientation data that can be applied to the quadcopter.