ME 102B Final Project

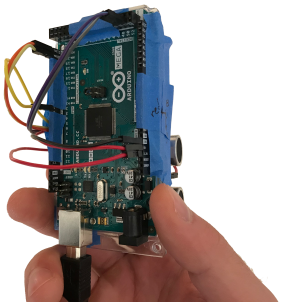
Casper Max

December 2020

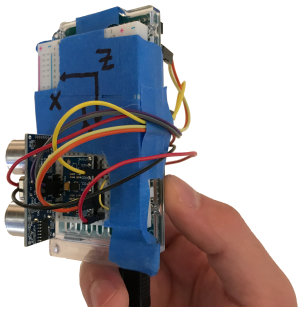
1 Project Description

My roommate recently got a new phone with a lidar sensor on it. Because I can't look forward to a new phone, I had to look towards my lab kit. We were given an IMU and ultrasonic sensor that we never got to use, so my first thought was to make a handheld scanner that I could generate point clouds with. While this proved hard to do because of issues with IMU calibration and dead reckoning, I did manage to produce the world's worst 3D scanner.

YouTube: `https://youtu.be/NlLPOdglobo`
GitHub: `https://github.com/caspermax/ME102B_Mini_Project`



(a) The Arduino Mega 2560.



(b) The ultrasonic sensor and IMU.

Figure 1: My 3D scanner. Please ignore the masking tape.

2 Circuit

There are only three components to this circuit (figure 2), including the micro. They are:

1. **Arduino Mega 2560 Rev3**: I had initially wanted to use the ESP32 for this project because it's smaller and would allow me to fit everything on a half-sized breadboard. However, the ultrasonic sensor recommends 5V supply and I did not want a second cable (5V supply) coming off my handheld scanner. I happened to have this board lying around so I used it, otherwise I probably would have used even more masking tape to fix the power jack onto the

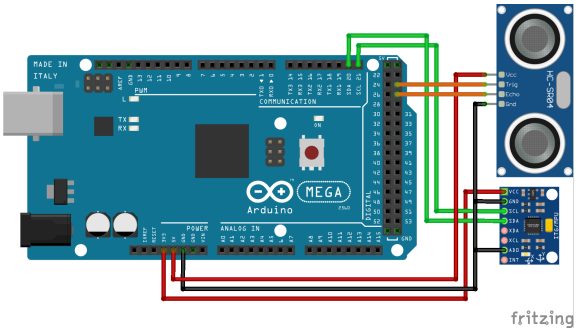


Figure 2: A circuit diagram for the scanner.

contraption. The micro provides power to both the ultrasonic sensor and IMU (both adding up to well below the micro's max current draw). The micro also publishes the sensor data to a ROS topic (more on that later).

2. **MPU 9250**: The IMU interfaces with the micro over I2C and requires 3.3V, though 5V is allowed because of an onboard regulator. The chip itself, an MPU 9250 contains a 3-Axis gyroscope, 3-Axis accelerometer, and 3-Axis magnetometer. It also houses a temperature sensor which can probably be used for more consistent calibration. The IMU does not output orientation angles but rather these 9 values, thus we must do this conversion ourselves (Madgwick filter). I also thought it would be feasible to get 3D positional data of the IMU using dead reckoning, however, because of the way position is calculated, drift builds up quadratically without some other form of reference and the IMU flies away extremely quickly.

3. **HC-SR04**: The ultrasonic sensor requires 5V supply and communicates with the micro by first waiting for a trigger pulse, sending several ultrasonic waves out, then converting the time taken for those pulses to bounce back into a pulse train. The micro converts this pulse train into a distance. While the spec sheet advertises up to 400 cm of range, I only got accurate readings up to 50 cm.

3 Program Structure

I used ROS, a framework for "robotic software development", to take data from the Arduino and visualize it. Because I'm generating point clouds, I wanted a real-time way to view the data, rather than saving it and graphing it later. I figured this would make debugging easier and more perhaps more fun. ROS allows you to do this using a series of "nodes" that subscribe and/or publish to "topics" which contain data ("messages"). Allowing programs to be visualized as a network of nodes communicating arbitrary messages makes conceptualizing projects much easier.

Figure 3 provides a reference for this paragraph. I used a ROS library to turn the main Arduino `.ino` file into a ROS node. This allowed the IMU and ultrasonic data (read by the micro) to be published to their respective topics. I wrote two more nodes (that run on my laptop), one to interpret IMU data and one to interpret ultrasonic data. This allowed offloading computation from

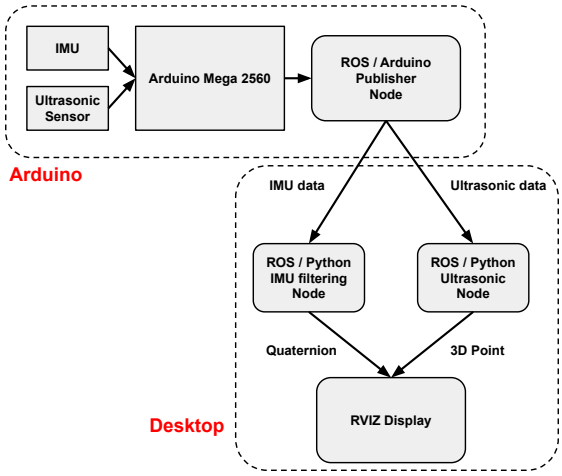


Figure 3: A diagram showing the flow of sensor data. The two main sub-blocks in the dotted boxes are the Arduino Mega and my desktop (laptop). To minimize the amount of computation done by the Arduino, it publishes the raw sensor data onto corresponding ROS topics which my laptop can read, taking over the work from there.

the Arduino to my computer. Finally, this conditioned data is read and displayed using RVIZ (ROS software for visualizing data).

While part of the reason to use ROS was to visualize data and offload computation, the biggest reason was so I could get more experience structuring ROS programs and interfacing what we've learned in this class with ideas learned in EECS C106A.

4 Code

Because I mainly wanted to focus on program structure and not programming, I used existing libraries for a majority of the heavy lifting. I will briefly describe the libraries and how I used them.

1. `https://bitbucket.org/cinqlair/mpu9250/src/master/`: Contains an Arduino sketch that reads data from the IMU. I decided to black box most of the libraries after looking over this sketch because it reads data off registers and does black magic with buffers that I don't understand yet. I felt that if I dived too deep into understanding, the mini-project might consume too much time. I added some ROS framework to this sketch that allows it to behave as a node and publish IMU data to a ROS topic.
2. `https://github.com/morgil/madgwick_py`: Contains a filtering algorithm, Madgwick filtering, that converts the 9 IMU readings into quaternions. I had never worked with an IMU before and thought they outputted orientation data by default so I ended up spending a majority of my time fiddling with getting this filtering to work.
3. `https://playground.arduino.cc/Code/NewPing/`: This library simplifies the ultrasonic sensing so you don't have to worry about triggering and reading PWM signals.

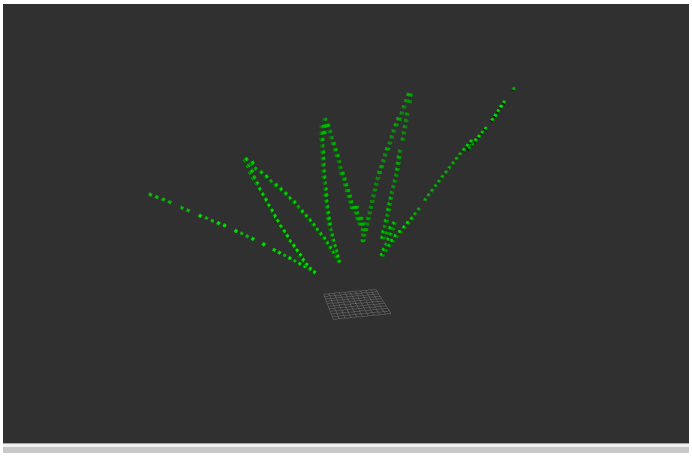
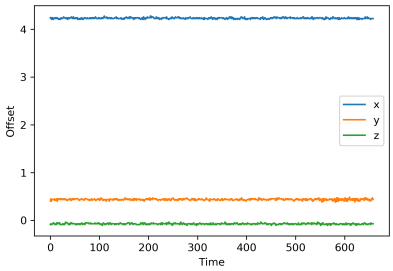
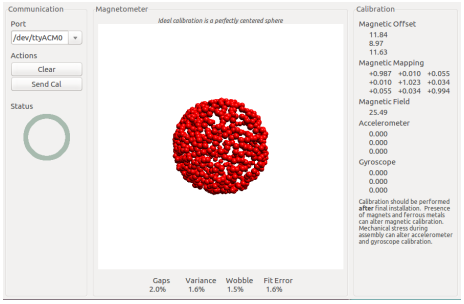


Figure 4: This set of points is generated from me repeatedly moving a notebook up to and away from the ultrasonic sensor as the IMU slowly drifts along one axis.



(a) Gyroscopic offset of IMU at rest. If I understand correctly, calibrating the gyro simply involves subtracting these offsets to center all axes around 0.



(b) I used MotionCal to calibrate the magnetometers. MotionCal reads data printed by an Arduino script and generates soft and hard iron offsets from a set of points generated by rotating your IMU.

Figure 5: Methods of IMU calibration.

5 Results

View the YouTube video for some live action footage. Overall, the ultrasonic range data came out well and it's respective transformation into the world frame works (it's not just a 2D line). However, there is significant drift in the IMU (figure 4). I calibrated the magnetometer and gyroscope (figure 5), but I didn't get to finish calibrating the accelerometer. Moving forward, I'd write a sketch to do all this calibration automatically. I still don't know if the drift comes from the accelerometer or the Madgwick filter (my implementation) and it's something I could look into in the future if I continue working on this project.