Lab 5: GPS

Engineering 100-950

Winter 2020

Materials

- 1 GlobalTop FGPMMOPA6H GPS Module and Antenna
- 1 MPX4115 Pressure Sensor
- 1 HIH-4030 Humidity Sensor
- 1 ADXL335 Accelerometer
- 1 B57164K Thermistor
- 1 LM7805 5V LDO
- 1 1kΩ Resistor
- 1 TMP36 Temperature Sensor
- 1 Battery
- 1 OpenLog
- 1 Level-Shifter

YouTube Videos

These videos are extremely helpful. We *VERY* STRONGLY recommend you watch these to succeed.

We developed three videos for you to complete this lab, along with supplemental videos for topics you may be unfamiliar with. This lab can be very, very frustrating if you don't pay careful attention to what we described in these videos.

- 'Pointers & Arrays (Part 1)' is a supplemental mini lesson on pointers. Our starter code expects you to have a basic understanding of pointers, so if you don't (or you need a refresher), please watch this video.

- ‘Arduino Lab 5a: GPS Modules, Part 1’ helps you wire your board up and verify it is working
- ‘Arduino Lab 5a: GPS Modules, Part 2’ guides you through the starter code and gives you advice on completing tasks. **This part of the lab will take you the longest, so it’s worthwhile to watch this video in its entirety.**
- For plotting your maps, you have a choice of implementing this code in either MATLAB or Python.
  - If you want to use MATLAB, you will be completing Lab 5b parts 1 and 2. Click here for a link to part 1.
  - If you want to use Python, you will be completing Lab 5c. Click here for the video.

Introduction

Up until this point, you have been dealing with **analog sensors** - their only output is a continuous DC voltage that is related to the physical phenomena they measure. However, analog sensors are only a small subset of sensors that exist! In fact, this single output format is rather limiting - we can only express information about one specific parameter of interest.

In previous labs, we used the Arduino to write lots of different types of information to the SD Card and Serial monitor. That information was encoded as digital signals. Thus, we’ve transmitted digital signals in the past, but until today we haven’t needed to receive and interpret them. In this lab, we’ll use our first **digital sensor**, the GPS module. This GPS module communicates through the simple UART communications protocol, which we’ve already covered in previous labs. We’ll need the **SoftwareSerial** library once more, but this time we need a few more of its features.

The GPS doesn’t really ’measure’ any sort of physical quantity. Instead, it receives messages from satellites which are used to generate GPS information packets. These packets are continuously output by the GPS module, even when it doesn’t have a proper **GPS fix**. This sort of continuous output is called **spewing**. Having a GPS fix means the GPS is reasonably sure about your location on earth, with only a few meters total of potential error.

The output of the GPS are formatted as **NMEA strings**. NMEA strings are simply a predictable way to format the location, altitude, and diagnostic stamps that the GPS provides into messages. These strings are identified by a 5 letter identifier. Here’s a list of all of them:

`http://aprs.gids.nl/nmea/`

We are only (mostly) interested in GPGGA strings, e.g. the lines of output preceded by “$GPGGA.” This 5 NMEA string organization will become clear as you view the raw output.

In the previous lab, we dealt with integrating all of the analog sensors. This was a basic primer in integration, where you had a system that was already functional and you added another sensor and integrated hardware and software. In the first portion of the lab, we will be doing the same with the GPS sensor, except we will be using the LDOs and a battery to achieve complete independence of your system from the computer and power supply.

IMPORTANT: This lab is very long, and if improperly planned out, can be very frustrating too. We recommend that a member of the team start working on the MATLAB (or Python) code to write out the Google Maps HTML file right away, while the rest of the team do parts 1 and 2. You can take the sample NMEA strings off of lecture slides to use as a test for your string parsing algorithms. Your MATLAB/Python code must eventually be able to take any number of GPS points and plot them onto a Google Maps HTML file. This system must integrate with how your Arduino code is written. That is, the output of your Arduino should be directly readable into MATLAB and then turned into a Google Maps file.

This does NOT mean you should designate one person to do all the code - that would be cruel and unusual, and that's not how we do things around here. Take this time to gather with your team and draw out a plan of attack that uses time efficiently. We recommend dedicating a minimum of one person to assembling and testing the breadboard, two people to writing the GPS code, and one person writing the HTML plotting code.

1 Wiring and Testing the GPS

The wiring and code needed is covered in 'Arduino Lab 5a: GPS Modules, Part 1'.

1. Connect the GPS to your Arduino. You only need to connect to the GPS's VIN, GND, RX, and TX pins. Designate a SoftwareSerial port to take in data from the RX/TX pins. The GPS takes 5V at its Vin pin.

2. Test your wiring by creating a new Arduino sketch. Write code that reads from your new SoftwareSerial port using the .read() function. If you aren't familiar with the .read() function in the SoftwareSerial library, there's a good reference for it here: `https://www.arduino.cc/en/Serial/read`

   Output each character to the Serial Monitor. If the .read() function returns a -1, that means that no new characters have been sent since the last reading, so you shouldn't output anything. The baud rate should be 9600 on all ports. Your output might look garbled, like a series of commas with nothing in between them, but that's simply because your GPS can't collect data indoors. We call this condition "not fixed." Now, take your board outside and see if you can get a GPS fix. This may take up to 10 minutes of waiting outside. Try to stand far away from any buildings to avoid multipath errors. Once you have a fix, your GPS module's red light will start flashing in a different frequency and you should see more dense output on your Serial monitor. Once you get a fix, come back inside.

3. Now, output everything to the OpenLog, and don't use the Serial Monitor at all. You don't need to go back outside to test this, as we'll do that later. **Save this code, and submit it with your PostLab.**

2 Completing the Starter Code

1. With the GPS and all the other sensors we've looked at thus far connected, and the system running on it's own independent power source, you have in front of you a completed sensor board. Take a moment to pat yourself on the back.

   How does it feel to have something free from the USB port? Pretty good, right? Yeah, we know.

2. After watching our video 'Arduino Lab: 5b: GPS Modules, part 2', download the starter files.

3. Rename 'gps.starter' to 'gps.ino' so you can open it in Arduino IDE.

4. Complete the starter code tasks where comments indicate. The comments look like this: *`// TASK #: Description of task`*.

   Read the **README.md** document that comes with the files for an in depth description of each task.

5. When completing Task 5, you can choose how to encode you information in the logger. You can print the complete GPGGA string to the OpenLog and do all of your processing in MATALB or Python, or you can do some amount of processing in Arduino and write that to the OpenLog. Any way you choose is fine, as long as you get the data from the GPS to some program on your computer that can make some type of plots. The only processing we specifically require to be done on the Arduino is the isolation of GPGGA strings.

6. Walk around the perimeter of the CSRB and the parking lot. The GPS will start receiving data (a 'fix') when the the LED marked 'FIX' on it stops blinking frequently. Record your output into a log file. It could take up to 10 minutes to get a fix. Make sure to walk far away from buildings and other sources of multipath error. **Save a working version of this program to submit with your PostLab. This is the second piece of Arduino code you'll need to submit.**

7. Translate your coordinates into HTML/Google Maps format using the examples posted during lecture and MATLAB (or Python).

A few things to remember:

- Debugging while only writing out to the SD card is a big pain! Change your code to write out to the serial port so you can see what is going on and debug it. Once you have it working, then put it back to the SD card. You could make a boolean variable to switch back and forth, if you wanted.
- Make sure the code you have uploaded is your final version before you begin testing. There should be a small button on your Arduino that will "reset" the code once you power it, so if you disconnect from your computer and connect to the LDO and press the button the code should start from void setup().

3 Combining your Lab 4 Code with your GPS Code

1. From the last lab, you should have a working sketch that can log data from all your sensors (except the GPS module) to the OpenLog.
2. In this lab, you completed a sketch to interfact with the GPS module.
3. Your next task is to combine these two files into one main sketch that can finally handle ALL the sensors on your board and write the data in a clean way to your SD card.
   (a) We recommend you copy code FROM the Lab 4 sketch INTO the GPS starter code file. This is because the GPS code will be the most fickle and error prone code in your final project.
4. This portion of the lab is very self directed and we encourage you to discuss code design with your team.
   (a) A good starting point might be to copy your global variables from Lab 4 into your code from this lab
5. **Save a working version of this program for submission with your postlab. This is the third and final piece of Arduino code you'll be submitting.**

Post Lab Questions

1. Why do we never connect to and pass data through the hardware Serial port (labeled as RX/TX or D0/D1) of an Arduino Nano?
2. What's faster, the clock speed of the Arduino or the baud rate? What is the clock speed of the Arduino? Explain in detail what this means about how often using the .read() function will return a -1.
3. What does the .available() function do?
4. What does the .millis() function do?
5. The Arduino has something known as a serial buffer. This buffer stores the bytes sent from the GPS (or whatever module you're connected with) to the Arduino. It works like a circular buffer. Look up what a circular buffer is (Wikipedia is a great source here), and give a brief definition below (1-2 sentences)
6. Why do we need to set the buffer index to 0 after resetting the buffer?
7. Describe what a pointer is and why we might use one in a program (1-2 sentences)
8. Describe how we represent strings in C. Explain why we need an array of size 6 to store "Hello", even though Hello is only 5 characters long (1 sentence)
9. Answer the following questions relating to `cstr`.
   - Describe the differences between `strchr` and `strstr`.
   - Why do we **never** change the last character of the GPSDataBuffer to something other than `'\0'`?
   - For the following `cstr`, get a pointer to the first newline character and get a pointer to the first occurrence of an S (capital, not lowercase).

     `"We miss Scott Smith dearly.\n You should join MBuRST and meet Scott.\n"`
10. How does a GPS use satellite signals to calculate its location? What data does the signal from a GPS satellite carry? How many signals does a GPS need to get a "fix"? Use any resources you have to help answer these questions. (3-5 sentences)

Post Lab Turn In

Unlike most other labs, this lab will be worth 30 points total in your grade and is graded as follows:

Lab 5 Postlab: 20 points (Engineering, Team)
Lab 5 Maps and Figures: 10 points (Technical Communication)

Keep technical communication lectures in mind while making your maps in figures.

Engineering Points

Turn in `.txt` files for all code written, and one `.pdf` for your postlab questions as follows.

1. All 3 Arduino programs that were created for this lab (AS `.txt` FILES, YOU WILL RECEIVE NO POINTS FOR `.ino` FILES).
2. All GPS KML and LOG data (AS `.txt` FILES).
3. All MATLAB (or Python) code used to process the GPS information (AS `.txt` FILES).
4. Answers to the post lab questions (AS `.pdf` FILE).

(Sorry to have to make you convert to `.txt`. Canvas doesn't allow previews of `.ino`, `.py`, and a lot of other file types so conversion to `.txt` is required for us to grade you. You can export to `.txt` through the IDE.)

Technical Communication

Keep technical communication lectures in mind while making your maps in figures. Among other things, be sure to:

1. Print your MATLAB or Python figures to png of jpg, do not use screencaps
2. Include all necessary titles and axis labels
3. Colors are your friend! Use them effectively