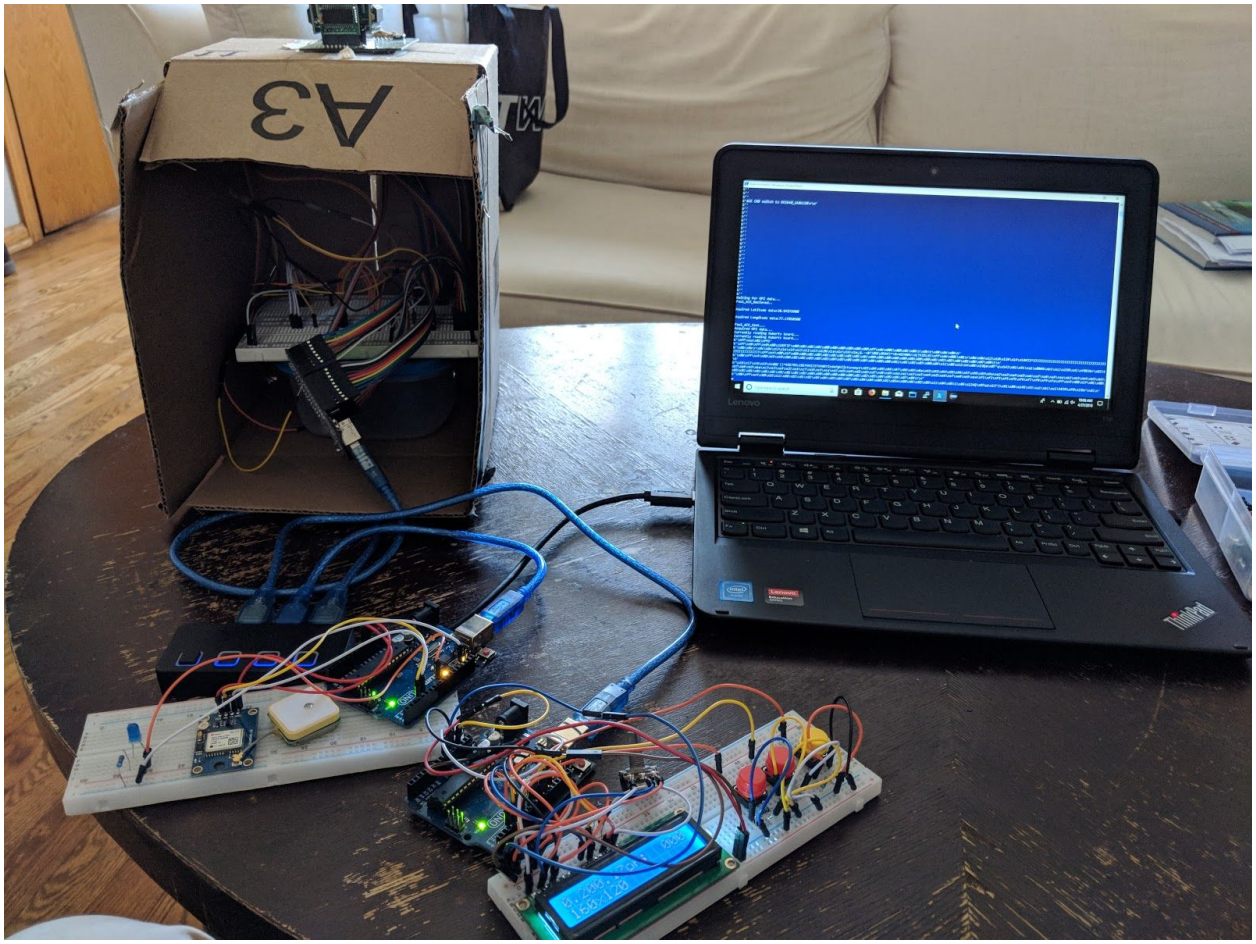
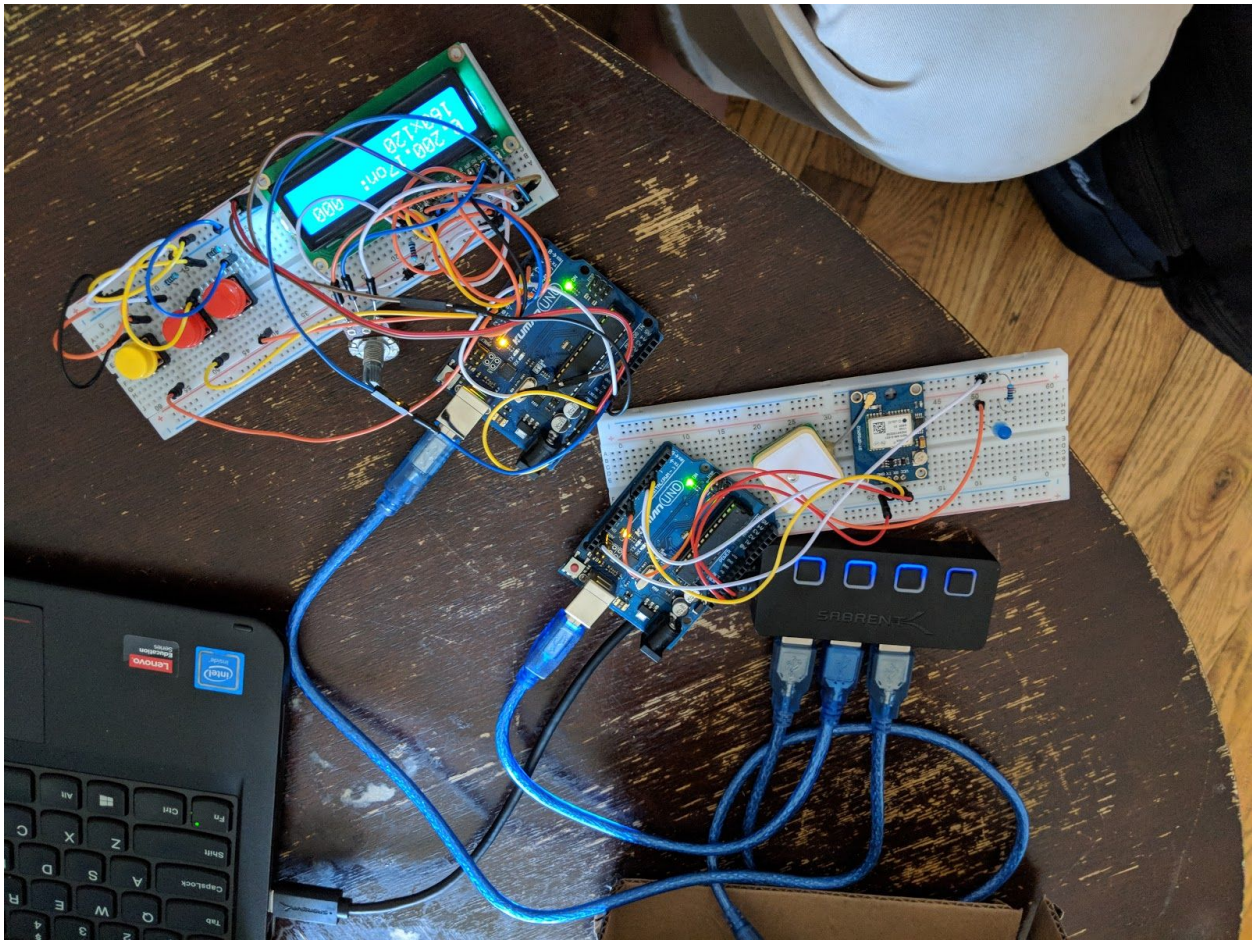
Arducam Panorama

Picture of final work



Diagram



Overview

Our vision for this project was to create a device which can take 360 degree photos much like Google maps street view. We thought about whether the pictures should be taken every interval of a couple seconds, when the GPS coordinates changed enough, or manually whenever a button was pressed. We decided to take pictures when a button was pressed for the relative simplicity compared to tracking GPS changes, and also having more control over when to take a picture compared to a set interval. With a trigger mechanism in mind, we broke the task up into three parts.

The first would be a device to take the photos. Since we wanted the photos to be 360 degree, we had to consider how to achieve the correct angle coverage. We settled on multiple cameras with special lenses designed to cover a wider angle. Luckily for us there was a manufacturer for Arduino cameras along with a shield to connect multiple of these cameras. With 4 cameras arranged in a circle connected to the Arduino, the next step was to find the right lenses to increase the viewing angles to capture a full 360 degrees around the unit. We found fish-eye lense attachments for The second would be a GPS device to track where the photo was taken.

The second part of this project comes to determining where the picture was taken. We talked about a few different methods of doing this (like using cell phones, etc…) but we settled on using GPS module, in the end. We concluded that the GPS module would offer us the most, for the cheapest amount.

With these two modules we can send data between boards through a PC and generate enough information to replicate the functionality of the Google Street View. Additionally, we have used several open source libraries to confirm that we can actually take this data and give it the aesthetic + look and feel of Google Maps and Street View.

The final piece was a controller to facilitate communication and data transfer. It starts by sending which resolution to take photos at to the Arducam, which can be changed anytime by the user pressing a button to increase or decrease resolution. When the user is ready to take a photo, they press the capture button which sends byte codes to the GPS and Arducam boards to make them do their work.

Arducam (RJ)

The ArduCAM camera module is the component(s) which take the panorama photos. All together we have 4xOV2640 camera modules and 1xArduCAM Multi Adapter Board. The ArduCAM Multi Adapter Board takes over all of the pins in the Arduino Uno. This caused a tremendous amount of headache because there is no board diagram or clear documentation for which pins the shield uses and which pins it does not.

A large majority of this project was trying to figure out how to interface the shield module with one Arduino Uno and then interfacing that Arduino Uno with the rest of the boards. At first, we tried using the SPI bus to communicate. The ArduCAM shield leaves us with a few pins (SCLK, MISO, MOSI) open (from the shield) which another board can use. However, after spending a fair amount of time figuring out how to get that to work we realized that using the shields SPI pins control the cameras functions and not the Arduinos functions. This was a large hit on our time and subsequent iterations/attempts to interface the boards moved away from this.

After trying to use the SPI bus we moved to using the Serial pins from the Arduino Uno which is connected the ArduCAM shield in parallel. This seemed promising, especially considering we were able to successfully interface controller board with the camera and transmit ~4kb of data to the controller. However, subsequent tests with the third board failed and blocked us from using any additional serial monitors via our PCs.

Additionally, if we removed the Serial cable from any of the Arduinos, we began to run into power issues — especially on the GPS module. And so the outcome of all of this was: we needed to interface the board directly to the PC and have the PC be the master and the rest of the boards be the slaves.

Despite going down the wrong path a second time, we did have a good time learning about the TX and RX I/O pins on the Arduino Uno. Unfortunately, we still do not know why the three boards couldn't be interfaced directly and also have the PCs monitoring the Serial bus. But, in the end, we were able to continue making progress.

The final design to interface the ArduCAM module with the rest of the boards has been the most promising. Instead of having board to board communication then a relay from one board back up to the PC we now have each board communicating the PC via a single serial connection. The PC now acts as sort of a task delegator in which it listens to the serial ports of all three boards and "translates" the request of one to the right board using encodings we specified in the controller. For example, if the controller has it's button pressed it sends out a signal, 0x11, which the PC picks up. Once the PC receives that signal it re routes it back out to GPS board. Once the GPS board acknowledges the request and sends the data the PC then sends another signal (0x10) via serial to the ArduCAM board. It, again, waits for acknowledgment and then scrapes the serial monitor for picture data.

To do this we needed to move outside of the scope of what the Arduino software can do; The serial monitor that comes with the Arduino software is incredibly limited. Instead, we ended up writing a custom Python program (which employs some open source libraries to establish the serial communication) which goes ahead and delegates and facilitates all of the communications and saves the images/GPS data.

As I mentioned this new approach has been the most promising. We are able to have the boards communicate and return image data. However, this method is incredibly slow and sometimes has data synchronization issues (since we are just scraping the data). We have learned

that UART is not the best method for bussing large amounts of data between devices. In future iterations of this project it might be better to explore other communication methods (maybe even go back to SPI).

One major detail I haven't exactly explained is the code on the Arduino Uno which interfaces with the shield. I am apprehensive of simply just posting code snippets and expecting you, the readers, to go through it and understand how to replicate it (since this was a painful process). Instead I want to provide small code snippets and give a comprehensive overview of how it works.

First and foremost, the ArduCAM shield uses SPI with a custom micro processor (on the shield) to send encoded data to the 4xOV2640 and picture data back to the Arduino. The ArduCAM library which comes with the board and is freely available on GitHub provides some abtractions to making this process a bit cleaner (but also a lot more confusing). We used those libraries and generated much of our code off of their boilerplate.

The high level flow of starting up the shield, cameras, and Arduino are: In your setup, send a test signal, 0x55 via MOSI (from your Arduino), let the shield handle the request, and wait for 1xOV2640 to respond. Repeat this 4 times. This initial check ensures that the cameras are in fact interfacing correctly AND can take pictures properly.

```
while (1) {
  //Check if the 4 ArduCAM Mini 2MP Cameras' SPI bus is OK
  myCAM1.write_reg(ARDUCHIP_TEST1, 0x55);
  temp = myCAM1.read_reg(ARDUCHIP_TEST1);
  if (temp != 0x55)
  {
    Serial.println(F("ACK CMD SPI1 interface Error!"));
    cam1 = false;
  }
  myCAM2.write_reg(ARDUCHIP_TEST1, 0x55);
  temp = myCAM2.read_reg(ARDUCHIP_TEST1);
  if (temp != 0x55)
  {
    Serial.println(F("ACK CMD SPI2 interface Error!"));
    cam2 = false;
  }
```

After these checks are done, there are some additional initializations. This is a good place to put code for starting, say, an LED (or button). Now you enter the main loop.

In the main loop, you have a massive switch statement that handles all of the different functionalities of the shield (and cameras). At this point in the code we read from the Serial monitor to pick the desired functionality. All of these codes come from the PC/Controller. Codes 0x0-0x8 all set the resolution of the board. Code 0x10 calls an additional function (read_fifo_burst) which takes 4x pictures and pipes them out it's Serial bus.

```
case 0x10:
  if (cam1) {
    Serial.flush();
    Serial.println(F("ACK FINAL_1"));
    flag[2] = 0x01; //flag of cam1
    for (int m = 0; m < 5; m++)
    {
      Serial.write(flag[m]);
    }
    read_fifo_burst(myCAM1);
  }
  if (cam2) {
    Serial.flush();
    Serial.println(F("ACK FINAL_2"));
    flag[2] = 0x02; //flag of cam1
    for (int m = 0; m < 5; m++)
    {
      Serial.write(flag[m]);
    }
    read_fifo_burst(myCAM2);
  }
```

This second function our team had to spend the most time fiddling with. How the ArduCAM actually takes the picture is not trivial and how it sends the code over the Serial bus isn't exactly clear. At a high level, it sends some codes over MOSI and listens to MISO for an Acknowledgement. It then goes into a while loop which buffers all of the data and then sends it into a Serial buffer. Once that is completed it the sends the buffered output.

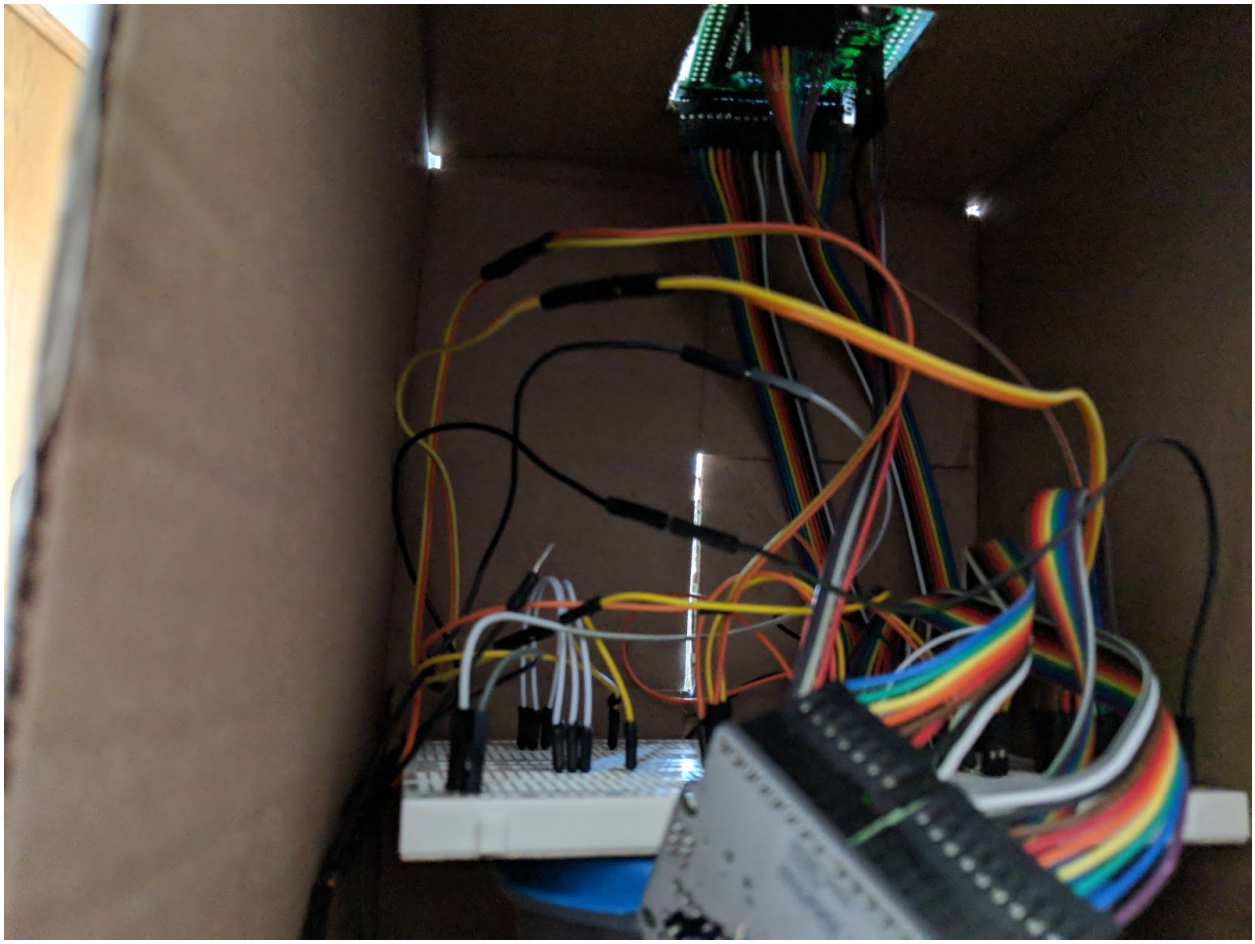
```
length--;
while ( length-- )
{
  temp_last = temp;
  temp =  SPI.transfer(0x00);//read a byte from spi
  if (is_header == true)
  {
    Serial.write(temp);
  }
  else if ((temp == 0xD8) & (temp_last == 0xFF))
  {
    is_header = true;
    Serial.write(temp_last);
    Serial.write(temp);
  }

  if ( (temp == 0xD9) && (temp_last == 0xFF) )
    break;

  delayMicroseconds(15);
}
```

At this point in the code, the PC starts (or has been) listening to Serial and will start saving the data.

Picture of internals:



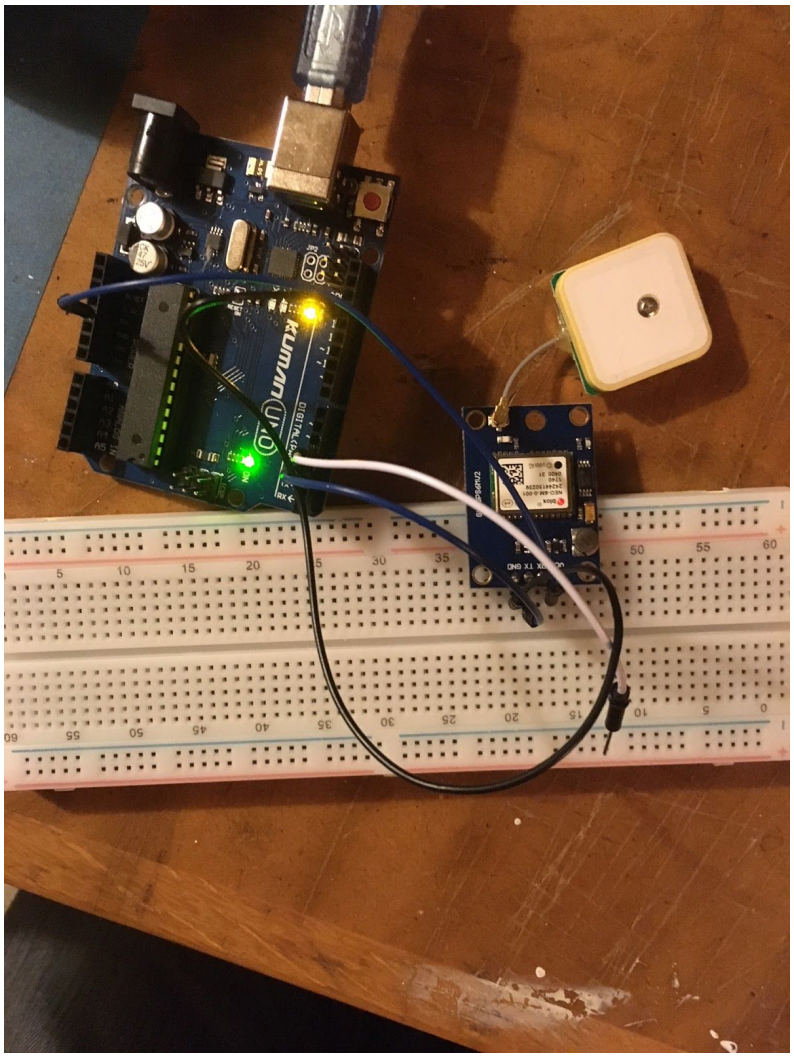
GPS (Paul)

General Aspects

One of the main motivations for the selection and coding of the GPS input component was to keep the wiring and coding of the overall serial traffic simple and clear. Since the photo processing software and hard-wired communication were likely to be more complicated to debug, the GPS device's signal transfer activity needed to be segregated, and the GPS's Arduino serial writes to the network needed to be discrete. The GPS can also then run at 9600 baud as designed, while the writes and overall system are at a higher baud rate. The schematic/architecture was selected early in the project to achieve this: the controller writes a "request" byte specific to the GPS Arduino, and the GPS Arduino returns the latitude/longitude

data on 0/1 pins. No other data is needed besides a String write of the two floats: latitude and longitude.

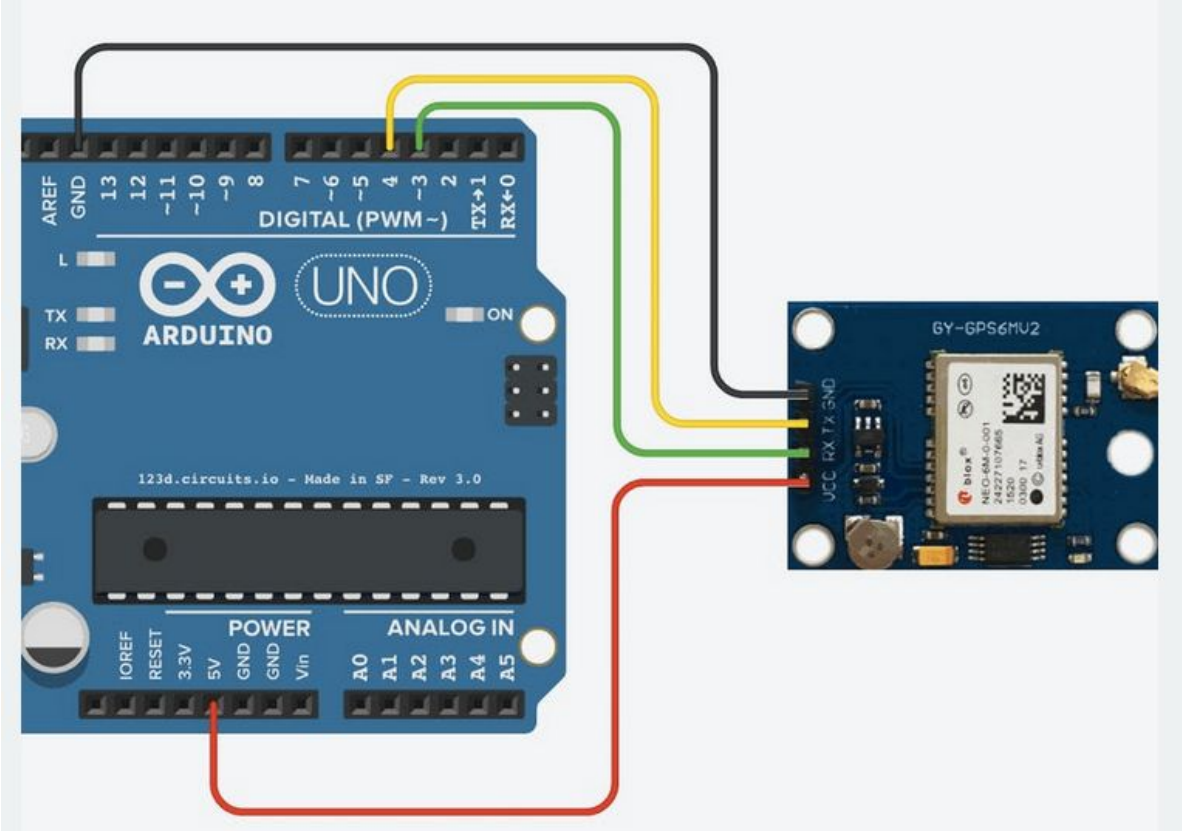
The implemented GPS unit is the NEO-6M U-Blox with Antenna, which is designed for use as a speedometer inside small drone copters, and therefore is lacking a large antenna apparatus, making it easier to position within our device/system. There is a convenient on-board LED to indicate when outdoor signal has finally been acquired.



(U-blox unit with GPS RX wire disabled; LED confirmation light not depicted here)

The wiring of the device is simply a TX/RX serial read through the 3/4 digital pins. Since no information is transferred to the GPS device, that wire is simply removed, and in fact the device doesn't update the latitude/longitude variables in our simple code, until that connection is

removed. The schematic picture and much of the code were both borrowed from Author: Ruchir Sharma @ https://create.arduino.cc/projecthub/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad , see below.



(GPS Arduino schematic, LED light for confirming signal TX is not shown here)

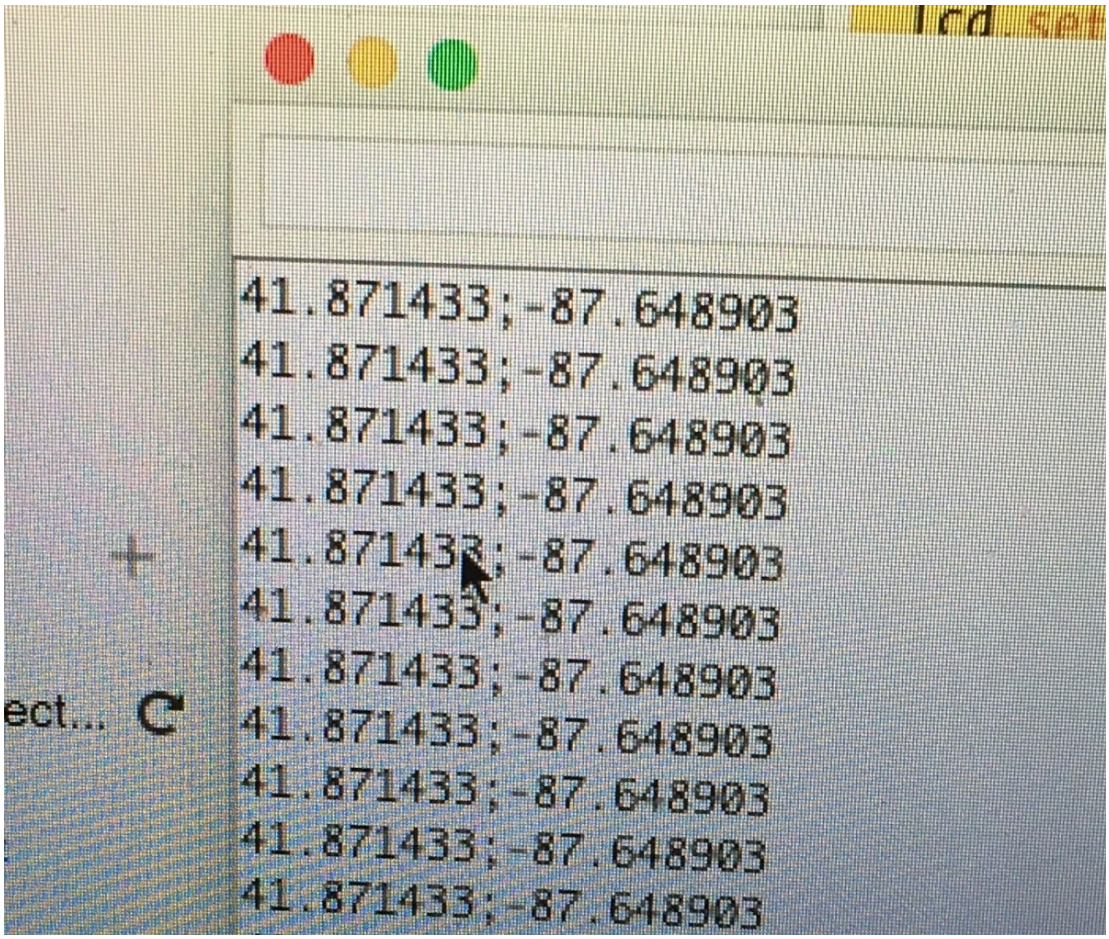
An example of the borrowed code to acquire signal would be:

```
while (gpsSerial.available()) { // check for gps data
        if (gps.encode(gpsSerial.read())) // encode gps data
        {
    gps.f_get_position(&lat, &lon); // get latitude and longitude
```

An example of code we added (to wait for the master controller's command of 0x11 byte) is:

while (Serial.read() != 0x11) {}

After this point, the newly acquired floats of longitude and latitude are simply converted to String and written with Serial.println to the network over 0/1 pins. If you comment out the *//while (Serial.read() != 0x11) {}*, you can debug the GPS signal acquisition via laptop, as follows. This entire board/program is a very simple component, intentionally so, stated again, so as not to complicated the sequence of serial writes elsewhere.



(Debugging example on Serial Monitor, not waiting for Master signal to TX)

Lessons Learned

There were a few elements of the GPS/Arduino components that could have been done differently, and which we might want to try again in the future. For example, the GPS unit is designed for an aerial device, and therefore does not always acquire immediately on the ground

(it takes a few minutes), and indoors often does not acquire at all. Since the user of our package might want to walk through structures like a parking garage while taking pictures, we might want to try the suggested antenna extension. Regarding power supply, the Arduino/GPS are being successfully supplied by USB power, but there appears to be difficulty also powering an LCD screen through the board, even when adding a 9V battery. No LCD screen was needed on this portion of the unit, however to advance the whole utility of the system by getting independent readout of GPS device to LCD screen, we might in the future want to use a separate USB port and adapter to power a screen and independently display GPS data, without dragging power off the board, or interfering with the serial connection transfer.

Overall, relating to the GPS component, we learned that it is relatively easy to request and receive latitude/longitude data in a discrete sequence, without heavy additional programming, and this is a benefit, since it allowed the other team members to focus on more complicated writes pertaining to image software. We would replicate that simplicity in any similar future project, but possibly with more advance GPS acquisition/display devices. Trivially, we also learned this device is made in India, since that is the default value of the floats!

References

We used the bulk of the code and schematic at this site, and we added a simple LED circuit to confirm transfer of the latitude/longitude to network. We also added a wait loop to the code, for the master controller request, and the LED write code.

https://create.arduino.cc/projecthub/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad

This is the actual device website:

https://www.u-blox.com/en/product/neo-6-series

These libraries were necessary, as linked from the above-referenced site:

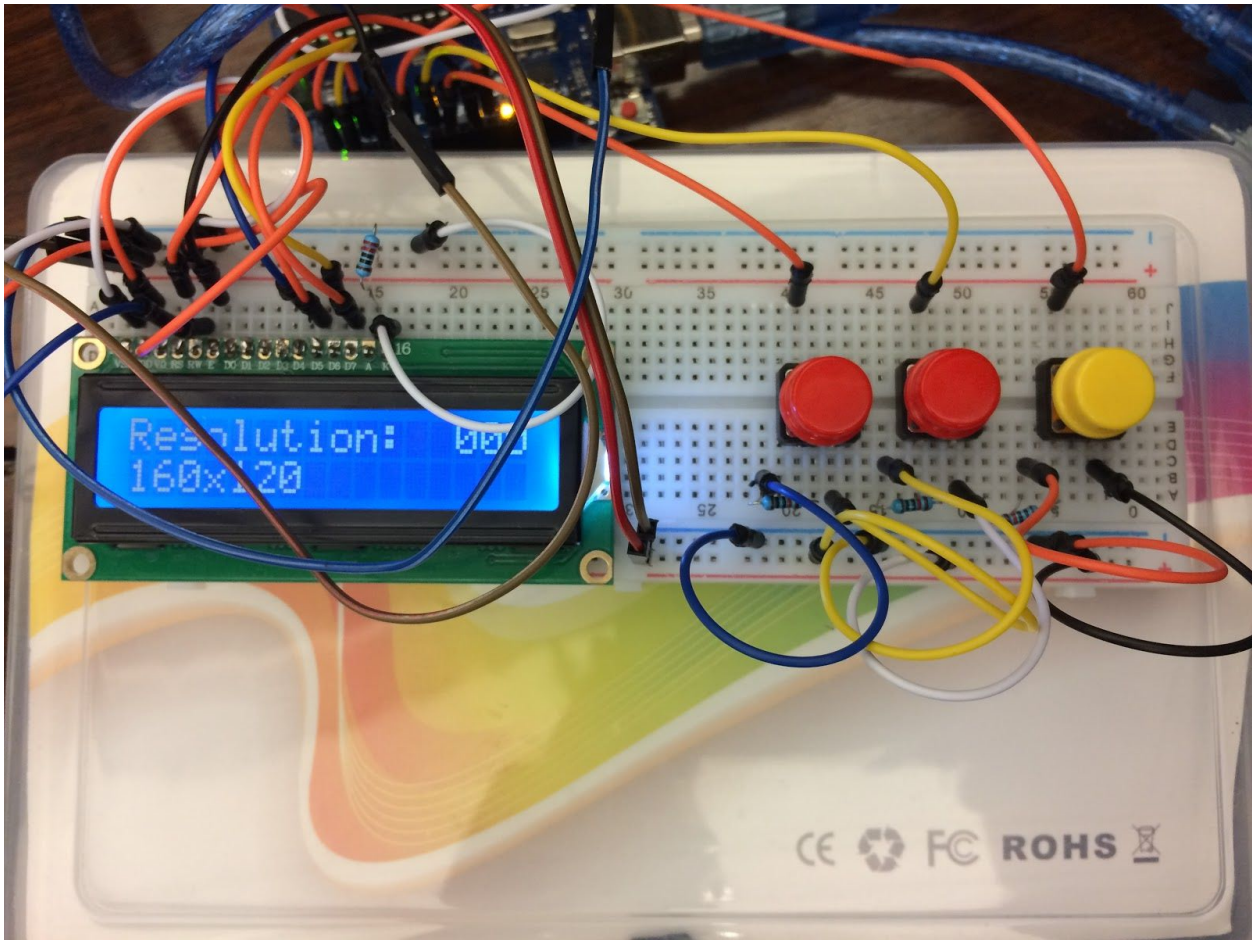
- (i) SoftwareSerial library
- (ii) TinyGPS library

Notice that some of the additional power supply/antenna product suggestions were also made at the Ruchir website above.

Controller (Hubert)

The purpose of a controller was to delegate the task of communication between the Arducam and GPS boards to a separate controller board. The controller would be the interface between the user wishing to take a picture, and the two boards who would generate the necessary data. Communication between the boards is achieved using the Serial ports. Until a button is pressed, the controller loops. The controller board writes special numbers to trigger the other boards to execute their functions. The GPS and Arducam boards will then send back the data generated to the controller on the Serial port for the controller to display.

While discussing how we would facilitate communication, we considered how to avoid having the slave GPS and Arducam boards send data which may be interpreted by the other as a start signal. Since the code would be just a byte, when the picture and coordinate data was sent on the Serial port, they may contain a byte which matches one of the byte codes the controller is using to signal the slave boards. We got around this issue by understanding how the Serial ports would be wired across the 3 boards. Port 1 for output from the controller is connected to both port 0 on the GPS and Arducam boards. This means whatever the controller writes will go across the wires to the input of the slave boards. The only input the slave boards are receiving originates from the controller board. As long as the controller writes the correct signal at the right time, interference between the GPS and Arducam boards is ruled out. Port 0 for input to the controller would be hooked up to both port 1 on the GPS and Arducam boards. This means that whatever the controller writes to Serial is all that both boards will be reading.



The controller has multiple functions. The first is to tell the Arducam board what resolution it should be taking pictures at. The resolution can be increased or decreased at the discretion of the user by pressing the corresponding button on the breadboard. The resolution is set on the Arducam board when it reads a number on the Serial port, ranging from 0 for the lowest resolution at 160x120 to 8 for the maximum at 1600x1200. When the controller is in setup, it calls Serial.write(0) which triggers the Arducam board to set its resolution setting to 160x120. Every time the user presses one of the resolution buttons, the corresponding int code is written on the Serial port and handled by the Arducam board.

Once the user has decided on a resolution, they can press a button to trigger the capture function. What this does is start the capture code which first Serial.write(0x11) which is picked up by both slave boards. The GPS board however is the one which is listening for the message 0x11, so the Arducam board keeps listening, not doing anything yet. The GPS board reads the message, executes its function to gather the latitude and longitude of its coordinates, and Serial.write() for both values. Since the only board listening to the slaves is the controller, the Arducam board doesn't read the coordinates, eliminating the possibility that the GPS board writing 0x10 accidentally triggers the Arducam board to start. After the controllers reads the coordinates from the GPS board using Serial.parseFloat(), it calls Serial.write(0x10) which sends the code the Arducam board is listening for.