**SENSOR CODE DOCUMENTATION**

**Summer 2018**

**Jessica Chapman**

|  |  |  |
| --- | --- | --- |
| **Date** | **Editor** | **Reason** |
| 2018-08-29 | Jessica Chapman | Created Document |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  | **Table of Contents** |  |
| ***How it works*** | *1.1 Sensor Setup* | *1.1.1 Full Setup* |
|  |  | *1.1.2 Improvements* |
|  |  |  |
| ***Programming*** | *2.1 Sensor Code* | *2.1.1 How it Works* |
|  |  | *2.1.2 Changing Sensors* |
|  |  | *2.1.3 Editing Code* |
|  |  |  |
| ***Circuits*** | *3.1 Sensor Circuit* | *3.1.1 List of Supplies* |

# 

# 

# **1.1 Sensor Setup**

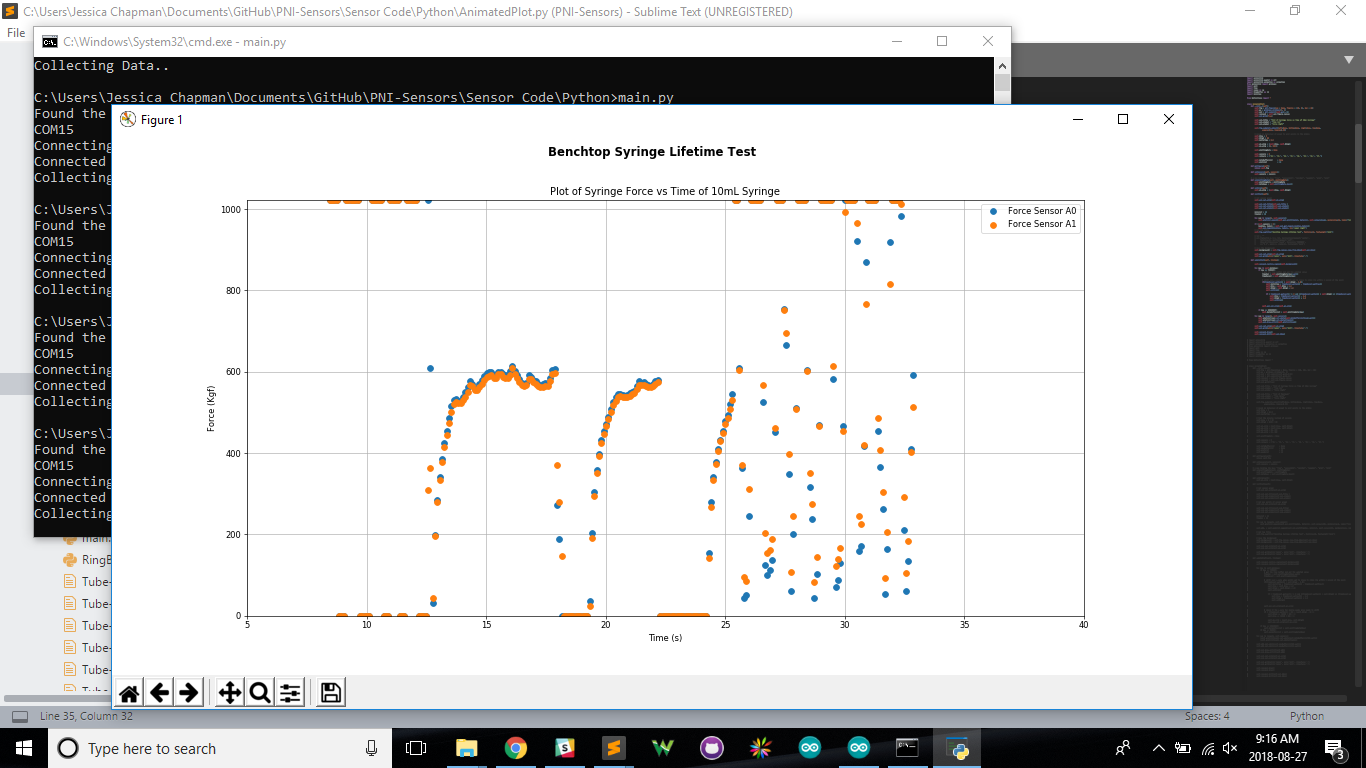
# 

# 

## **1.1.1 Full Setup**

# 

The sensor code and circuit are used for measuring different quantities with sensors and displaying it on a live plot. The titles and labels can easily be changed to match the appropriate application.



There are two programs found in [ProjektEskie](https://github.com/ProjektEskie/PNI-Sensors) under PNI-Sensors, one in C++ and one in Python 3. The C++ file should be loaded into the Arduino that the sensors are plugged into and the python script can be run from a laptop or Raspberry Pi. The two programs communicate via a serial interface using the commands found in the Definitions.h file.

Currently, the code is built for 1-6 linear voltage sensors, but has the capability to add different kinds of sensors. It has a manual or standard calibration set with the MANUAL keyword in Definitions.py. In the manual calibration the program will prompt the user for input and known values and will calculate the linear factors through these. In standard the known factors can easily be set in the Definitions.py file under ALPHA and BETA such that y = ALPHA \* x + BETA.

The code should only be used with voltage sensors, not current sensors, with an amplification going from 0-REF\_VOLTAGE, this will either be 2.56, 3.3 or 5V.

## **1.1.2 Suggestions for Improvement**

* Improving the manual calibration function, it could use some editing and testing (there may be some problems with the zeroing and then calculating the ALPHA and BETA values) also it could use some work for supporting multiple sensors that are different types (ie: a pressure sensor, a load cell, and a temperature sensor)
* In the plotting file, adding a secondary axis for a different kind of sensor (ie: left axis for pressure sensor and right for temperature) since they will have different scaling values and max/mins
* Having a way for the arduino to know if a sensor is plugged in so the user does not have to manually add the sensor to the arduino file, update the number of sensors, and then begin it.
* Adding a way for the program to automatically attempt to reconnect to the Arduino if it drops the connection (this can happen when it is on for a very long time)

# **2.1 Sensor Code**

## **2.1.1 How It Works**

The code works by using an Arduino program in combination with a python script which communicate with each other over the serial monitor.

### **Arduino Code - ArduinoCommunication.ino**

The Arduino code should be loaded into the Arduino and can be tested using the Serial Monitor. Note that any print statements added for debugging purposes which do not follow the format used with the output string will cause the python script to either be unable to connect to the Arduino or not communicate with it.

### **Python Code - Main.py**

The Python script uses multithreading to run the animated plot on the main thread (through the function fileAndUpdate), and the querying and parsing information from the Arduino on a worker thread (through the function readSensor). These threads only interact through the queues and nothing else.



The code primarily relies on the ring buffers which hold all of the data to be displayed on the plot. These ring buffers have a maximum size defined in the Definitions.py file. The ring buffers will begin to replace it’s first values with new values when it exceeds the maximum size.

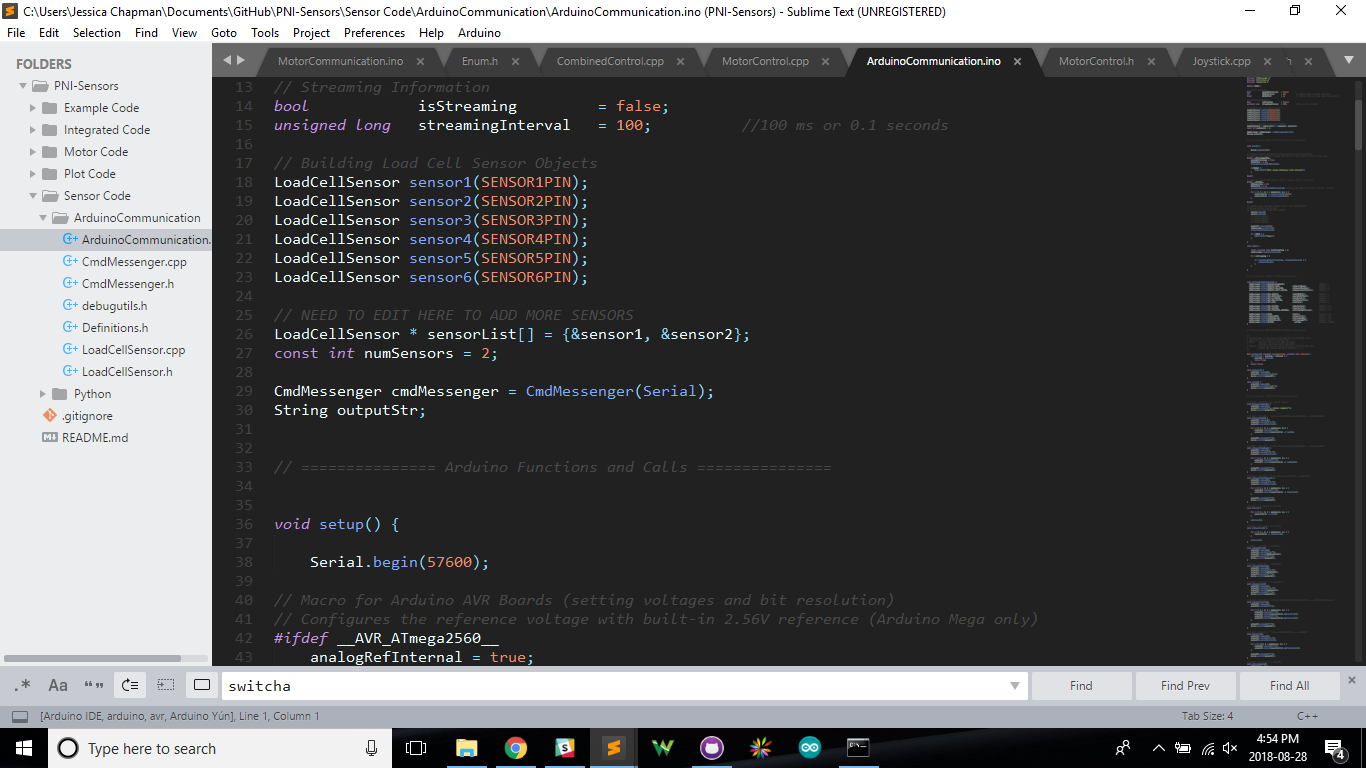
### **Python Code - AnimatedPlot.py**

The Animated Plot uses blitting to save time on updates, this means that it will only update bits that changed and leave the ones that are the same there instead of erasing and reloading every time. The plot draws the sensor values over each other and will label them automatically on a legend. It is animated so that it is very close to real time.

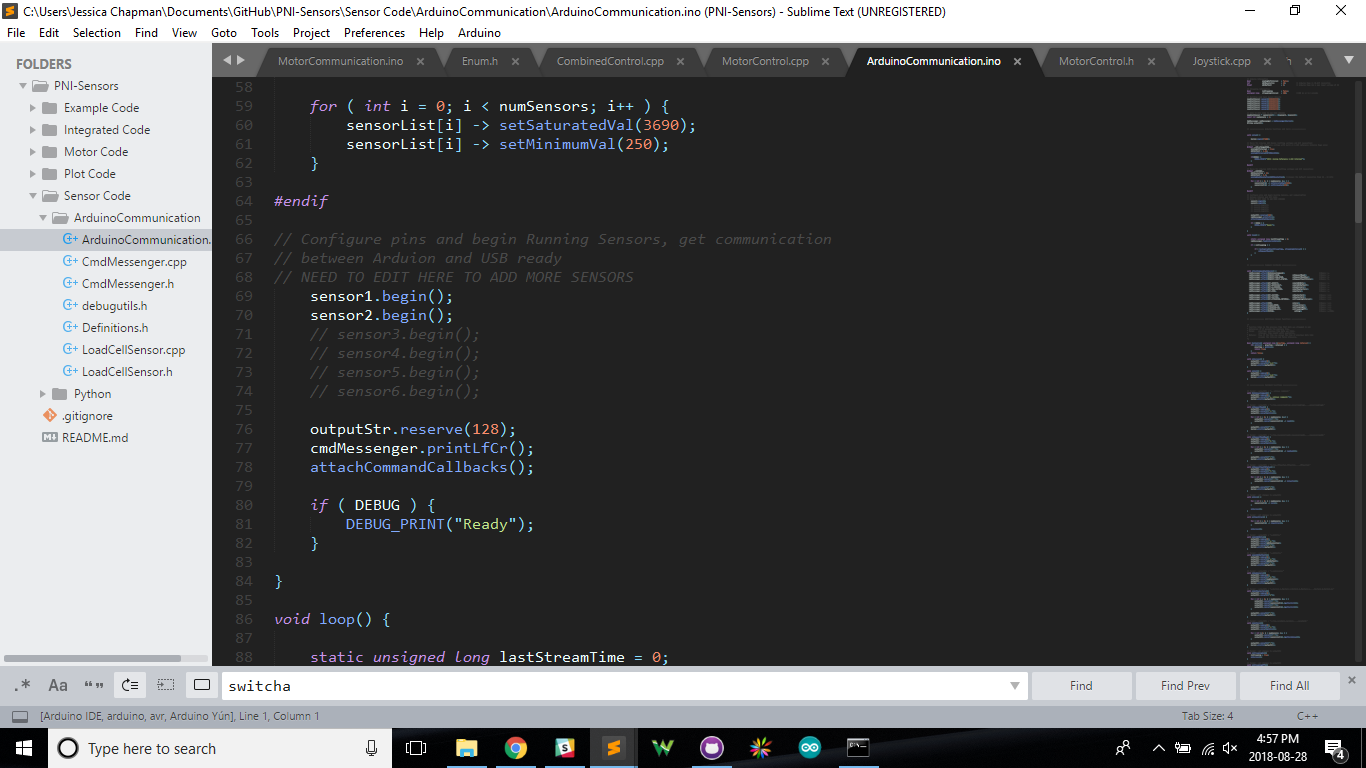
## **2.1.2 Changing the Amount of Sensors**

To edit or add additional sensors you need to edit two places in the Arduino Code.

1. Add however many sensors you want to the sensor address list as seen below such that the list should be LoadCellSensor \* sensorList[] = {&sensor1,...,&sensorN}; also update the numSensors value.



1. Begin your sensor in the setup() function.

****

No python code should be changed, it will automatically register the amount of sensors and update the plotting and filing to match.

**2.1.3 Editing the Code**

The time.sleep() at the end of the readSensor function should be updated based on how much the information is being processed. It is there so as not to clog up the Queue and lose some data when it goes to be filed and shown on the animated plot.

The animated plot can be easily edited as most of the attributes are listed at the very top of the python file.

# **3.1 Sensor Circuit**

## **3.1.1 List of Supplies**

|  |  |  |
| --- | --- | --- |
| Part | Part Number | Company |
| Socket Adapter | 309-1120-ND | Digikey |
| [IC Amplifier](https://www.digikey.ca/products/en?keywords=AD8221ARMZ-R7DKR-ND) | [AD8221ARMZ-R7DKR-ND](http://www.analog.com/media/en/technical-documentation/data-sheets/AD8221.pdf) | Digikey |
| [Load Cell](https://www.sparkfun.com/products/13331) | SEN-1331-ND | Digikey |
| Precision Resistor (250Ω) | 696-1534-ND | Digikey |

The amplifier pin configuration can be found below, the -Vs can be plugged into GND. The resistor(s) should go across Rg from one to the other. -IN and +IN should come from the load cell.

## 

The formula for calculating gain is shown below: