Project: Sensors

Zack Andalman

**Objective**

We need a network of various sensors to monitor the lab conditions. These sensors include a magnetometer, an accelerometer, and a temperature/humidity sensor. Data will go from the sensors to an Arduino to a Raspberry PI to the InfluxDB database.



**Raspberry Pi Code**

The Raspberry Pi code is designed to work for any type of sensor. Most settings are adjustable from the configuration file, which is formatted to work with the python library configparser. The general settings are

* interval - data collection interval (seconds); if zero, data will be collected as frequently as possible
* indef - collect data indefinitely (boolean)
* t\_collect - data collection time (seconds); ignored if indef == True
* upload - print a message when data is successfully uploaded; for debug purposes
* url - Influxdb database server url
* port - Influxdb database port
* username - Influxdb username
* password - Influxdb password
* database - Influxdb database name
* location - name of directory to store data which is not successfully uploaded

To add a new sensor of a defined type, create a new section in the configuration file by writing the sensor name in brackets. On the following line, write **sensor :**.This lets the parser know that the section represents a sensor. The sensor settings are

* use - use sensor (boolean)
* type - sensor type; the options should be listed in the configuration file
* tag\_names - tag names separated by commas
* tag\_values - tag values separated by commas; the number of tag names and tag values must match
* print - print measurements (boolean)

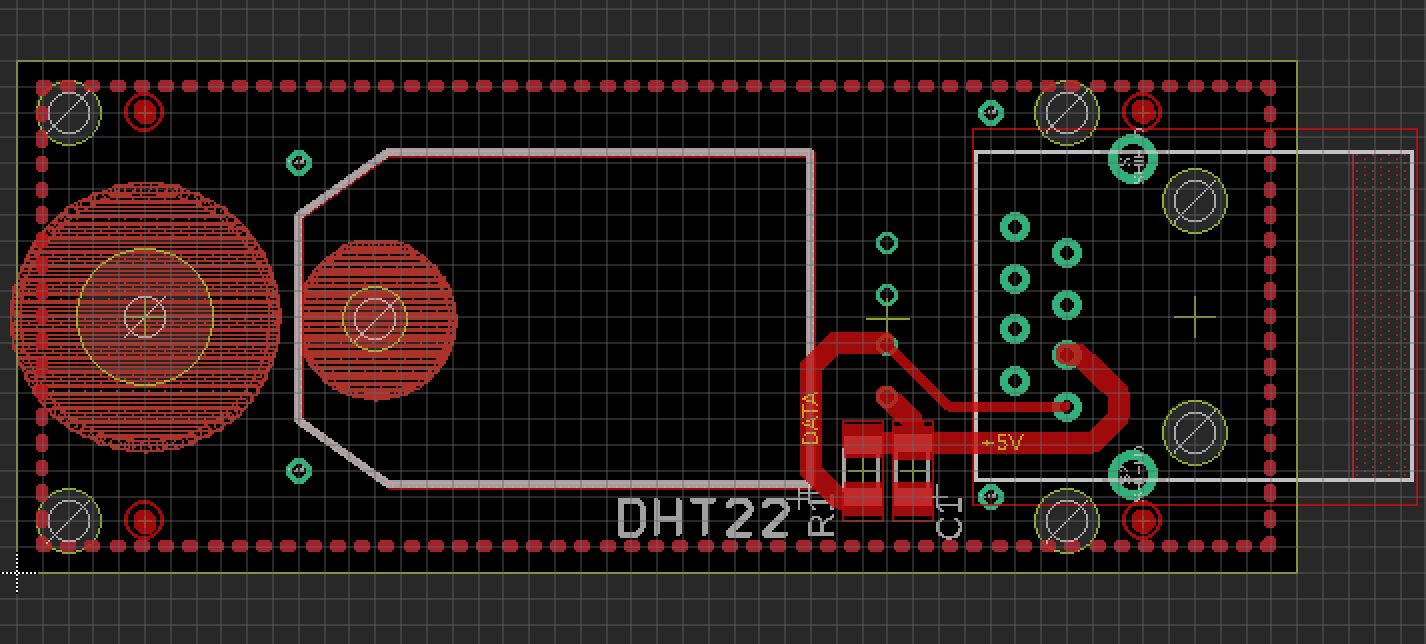
Sensors can also have settings specific to the sensor type, which should be listed in the configuration file. There are two sensor subclasses, Arduino\_Sensor and Pi\_Sensor. Arduino sensors are connected to the Raspberry Pi through an Arduino and Raspberry Pi sensors are connected directly to the Raspberry Pi. Arduino sensors require the board\_port as a setting. Raspberry Pi sensors require the GPIO pin as a setting. To define a new sensor class:

1. Create a new class in sensors.py with the appropriate inheritance class.
2. In the initialization function, define any necessary sensor-specific parameters as keyword arguments and call **super().\_\_init\_\_** to run the inheritance class initialization. Append any required libraries to **self.libraries**. Define the sensor measurements and units in **self.measure\_types** and **self.units** respectively.
3. Define a read method which outputs a list of measurements in the same order as **self.measure\_types**. Arduino sensors have a pre-defined read method but this may be overwritten if the data is encoded differently.
4. Define an optional filter method which takes the output of the read method as an input and outputs the filtered measurements.
5. Add the new sensor class to the **sensor\_types** dictionary.
6. Import the necessary libraries.
7. Document your changes in the configuration file.

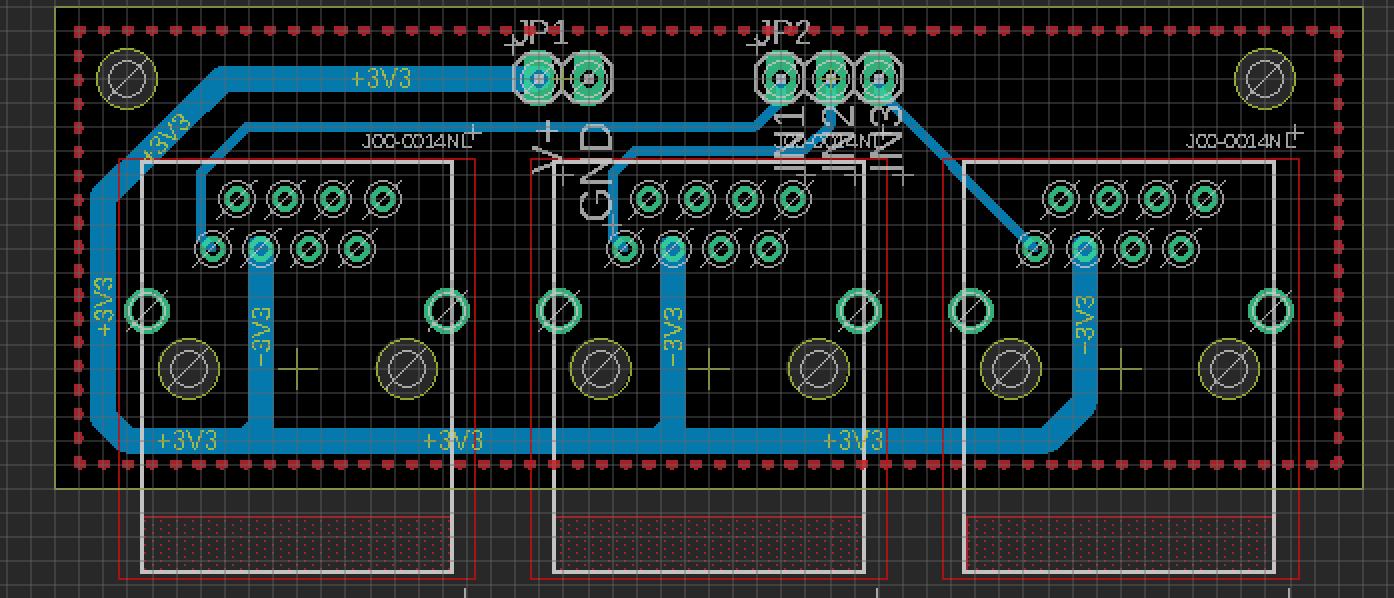
The code includes three errors classes. ConfigError occurs due to an error in the configuration file. ModuleError occurs when the required sensor libraries are missing. MeasurementError occurs when there’s something wrong with a measurement. This can be a problem with the read method definition, the Arduino code, or the sensor itself.

**Temperature/Humidity Sensor**

Our design is based on <https://github.com/JQIamo/RPi-Temp-Humidity-Monitor>. We will use three DHT22 temperature/humidity sensors from Adafruit to record temperature and humidity conditions in different parts of the lab. Each sensor will be attached to individual boards which are connected via ethernet to a central board. The central board will interface directly with the Raspberry Pi. The resistance of the resistor on the individual boards depends on the length of the ethernet cable connecting it to the central board. Use 10 kOhms for a trivially short connection and 1 kOhm for a connection longer than 15 ft. The capacitor should have a capacitance of 100 nF.



A board which connects each DHT22 sensor to an ethernet cable.



A board which interfaces with the Raspberry Pi.

*Components*

* DHT22 (x3): <https://www.adafruit.com/product/385>
* Ethernet jack (x6): <https://www.digikey.com/product-detail/en/pulse-electronics-network/J00-0014NL/553-1612-5-ND/1036859>

**Magnetometer**

We will be using the QMC5883L magnetometer to monitor ambient magnetic field conditions in the lab. The QMC5883L breakout board has five pin holes: DRDY, SCL, SDA, GND, and VCC. The RDY pin can remain open. Connect VCC to the 5 V pin on the Arduino. Connect GND to the ground pin on the Arduino. Connect SDA to the I2C data SDA pin on the Arduino. On an UNO & '328 based Arduino, this is also known as A4, on a Mega/Due it is also known as digital 20 and on a Leonardo/Micro, digital 2. Connect SCL to the I2C clock SCL pin on the Arduino. On an UNO & '328 based Arduino, this is also known as A5, on a Mega/Due it is also known as digital 21 and on a Leonardo/Micro, digital 3.

Install the QMC5883LCompass library from the Arduino IDE to run Magnetometer.ino. The code includes functions to adjust the magnetometer settings.

* BYTE - change the I2C address from the default.
* STEPS - integer from 1 to 10; the number of steps to smooth the results by. Higher steps equals more smoothing but longer process time.
* ADVANCED - boolean; turn advanced smoothing on or off. Advanced smoothing will remove the max and min values from each step and then process as normal. Turning this feature on will result in more smoothing but will take longer to process.
* MODE - operating mode; 0x00 = standby, 0x01 = continuous
* ODR - sample rate; 0x00 = 10 Hz, 0x04 = 50 Hz, 0x08 = 100 Hz, 0x0C = 200 Hz.
* RNG - range; 0x00 = 2 G, 0x10 = 8 G.
* OSR - oversample ratio; 0xC0 = 64, 0x80 = 128, 0x40 = 256, 0x00 = 512.

You are now ready to start recording magnetometer data. For more information, see <https://github.com/mprograms/QMC5883LCompass>.

ALTERNATIVE

Install the QMC5883L Arduino library from <https://github.com/dthain/QMC5883L>. The code includes functions to adjust the magnetometer settings: sampling rate, range, and oversampling. To calibrate the magnetometer, call **readHeading()** continuously while rotating the magnetometer on all axes. Once enough data has been collected, the function will return integer values between 1 and 360 degrees. The function assumes that the magnetometer is parallel to the ground.

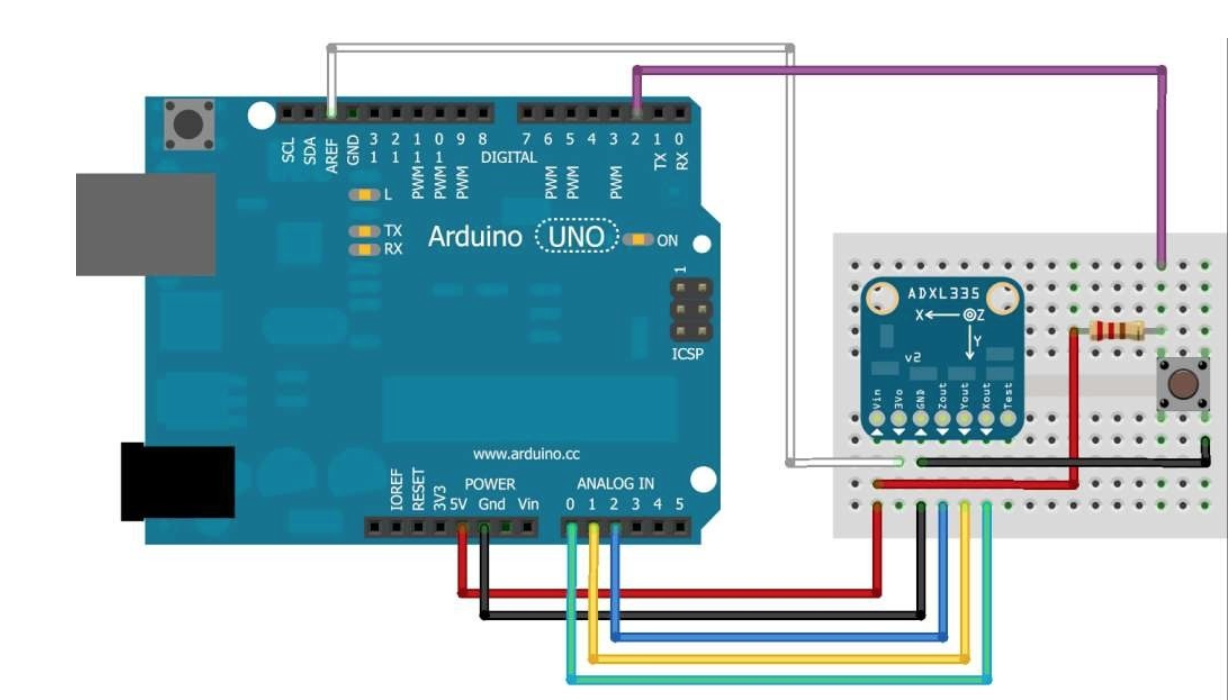
*Components*

* QMC5883L: <https://www.amazon.com/gp/product/B008V9S64E/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1>

**Accelerometer**

We will be using the ADXL335 accelerometer to monitor the stability of the optical tables. The ADXL335 breakout board has seven pin holes: Vin, 3Vo, GND, Zout, Yout, Xout, and Test. Connect Vin and GND to the 5 V and GND pins on the Arduino respectively. Connect the Xout, Yout, and Zout to pins A0, A1, and A2 on the Arduino respectively. Connect 3Vo to the AREF pin on the Arduino.

To calibrate the accelerometer, mount it on a breadboard or protoboard. Connect a switch to one of the digital pins on the Arduino as shown in the diagram below.



In the calibration code, make sure that **buttonPin** corresponds to the digital pin of the switch. Then

1. Run the calibration code.
2. Place the breadboard so that it’s normal to the +z axis.
3. Flip the switch / press and hold the button until “Calibrate” appears on the serial monitor.
4. Repeat the previous steps for the +y, +x, -y, -x, and -z axes.

Once calibrated, the output will show the calibrated raw range for each axis and the corresponding g-forces. In the accelerometer code, use the raw ranges to set **xRawMin**, **xRawMax**, **yRawMin**, **yRawMax**, **zRawMin**, and **zRawMax**. You are now ready to start recording accelerometer data. See <https://learn.adafruit.com/adafruit-analog-accelerometer-breakouts/overview> for more information.