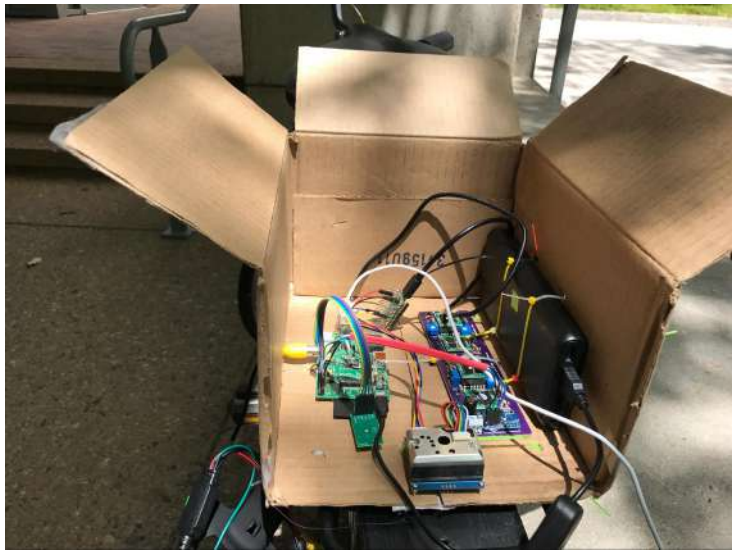
Air Quality Kit

Seeing Like a Bike (LMC 6650)
Project Documentation

Lorina Navarro
Reuben Fishback
Oriana Ott

Table of Contents

Summary



The goal of the project studio, **Seeing like a Bike** is to build a system that can be affordably replicated by citizen scientists to measure environmental stress factors experienced by cyclists.

Our goal of the **Air Quality Kit** is to measure ground air quality at a personal level for cyclists, compare measures to local sensors and national standards, and compare costs and benefits between our array and commercially available personal air quality sensor kits.

This document covers the goals of our team within the context of the smart bike project, our choice of hardware, instructions on implementation and code, and a comparison between this project and existing sensor kits to evaluate their relative cost and value.

Parts Inventory

Air Quality Sensors

Purpose	Part Number	Cost	Data Sheet
Particulate matter (PM 2.5)	Waveshare Dust Sensor Detector Module with Sharp GP2Y1010AU0F	$12.99	http://www.waveshare.com/wiki/Dust_Sensor
Nitrogen Oxide	SPEC sensor (1684-1017)	$20	http://datasheets.globalspec.com/ds/15/DigiKey/07AB7B5C-AEE5-4F36-8579-A6E02882350A
Sulfur Dioxide	SPEC sensor (1684-1021)	$20	http://www.spec-sensors.com/wp-content/uploads/2016/04/3SP_CO_1000-P-Package-110-102.pdf
Ozone	SPEC sensor (1684-1013-ND)	$20	https://www.digikey.com/product-detail/en/spec-sensors-llc/110-401/1684-1013-ND/6136376
Carbon Monoxide	SPEC sensor (1684-1000)	$20	https://www.digikey.com/product-detail/en/spec-sensors-llc/110-102/1684-1000-ND/6136363

Other Electronics

GPS sensor	GTPA010
Analog to Digital Converter (ADC)	ADS1015 (4-channels)
Microcontrollers	Raspberry Pi Model B Arduino Teensy 3.2
Resistors	15 1 MΩ resistors 3 740kΩ resistors 3 150Ω resistors 4 500kΩ potentiometers
Capacitors	8 0.1 UF capacitors 1000 UF capacitor

Amplifiers	2 MCP6044 amplifiers
Power source	PowerCore 26800 26800mAh External Battery
Miscellaneous	2 micro USB cables (data-ready) Jumper cables

Notes on Microcontrollers

Raspberry Pi Model B

- Many GPIO pins for quick connection to I2C
- (Inter-integrated Circuit) Protocol and Serial interfaces
- Python Programming Language
- Ethernet port to connect to server
- Display port used for mini computer setup.

Teensy Arduino

- C/C++ Programming Language
- Reliable timing for PM sensor
- Analog inputs
- Lightweight & portable
- Convenient serial output to communicate with
- Raspberry Pi

Notes on Particulate Matter Sensor

Rationale: Provides general indicator of exposure to tailpipe emissions
Cyclists have high exposure and health risks from particulate matter

How it works: A particulate matter sensor measures the obstruction of a light by particles in the air that are too fine to see. The density of those particles is based on the power, which determines the sensitivity of the measure, and the presence of particles between 2.5 and 10 microgram per cubic meter. This sensor cannot be hand calibrated.

Methods used to test a pm sensor include exposure to tail pipe exhaust and kitchen smoke from candles or frying oil.

Factors that may cause reduced accuracy of pm sensors

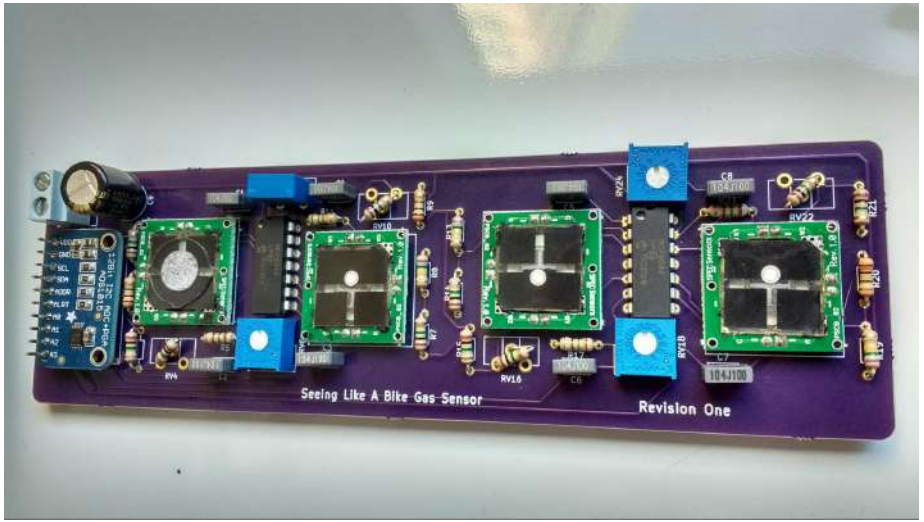
- Light
- Voltage, insufficient or inconsistent power supply
- Damage or overexposure
- Poor quality or damaged wires

Notes on Analog Gas Sensors

Nitrogen Dioxide, Sulfur Dioxide, Ozone, and Carbon Monoxide

Rationale: Compounds from fossil fuel combustion, cause serious respiratory diseases. Followed EPA standards for harmful gases.

How they work: Each electrochemical sensor produces current proportional to the concentration of the gas. Potentiostat circuit converts current to voltage and provides amplification. All four sensors are connected on a PCB (pictured right).

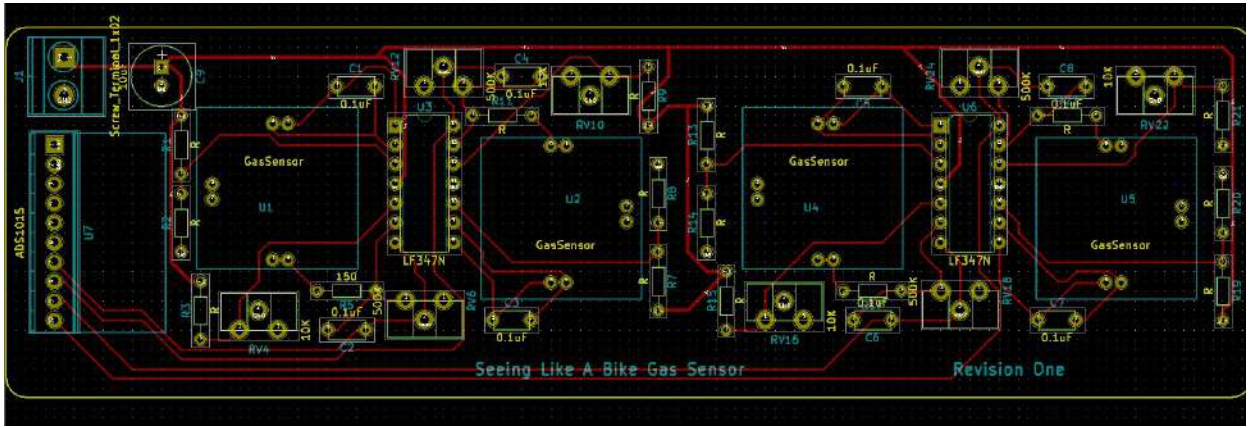


Factors that may affect accuracy of sensors

Temperature
Other gases
Exposure to water
High concentrations of contaminants

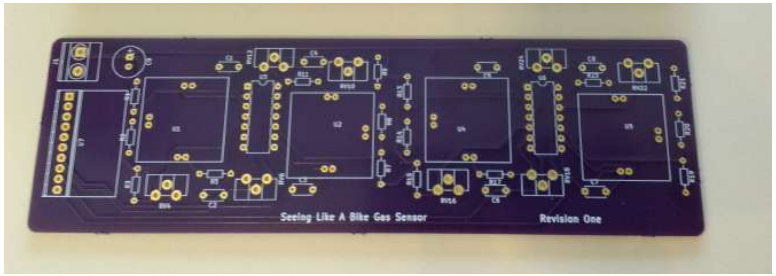
Soldering the Gas Array

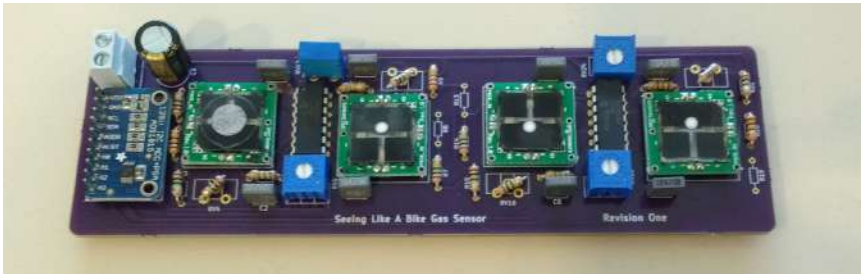
PCB Design



We created a gas array design for our four analog sensors - CO, SO2, O3 and NO2.

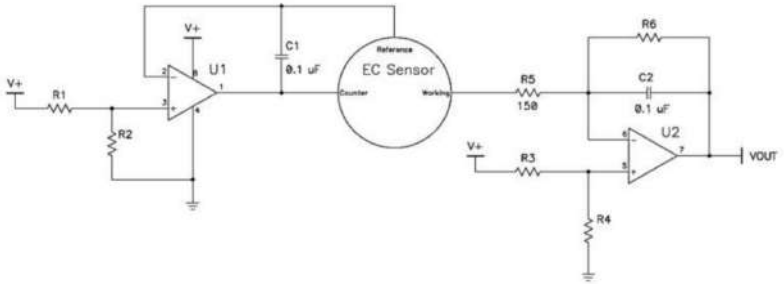
Each sensor is connected to a potentiostat circuit with output leading to an ADC (analog to digital converter). Powered by 3.3V





PCB Design

Potentiostat Circuit



Resistor Values used to get proper voltage bias.

R5 will convert current to voltage.

U2 amplifies the voltage by the value of R6

Circuit will be repeated 4 times in PCB

Typical circuit implementation based on SPEC documentation

Sensor Part Number	Target Gas	Recommended Nominal Bias, V_{WE-RE}	Typical Circuit Implementation*					
			V+	R1	R2	R3	R4	U1/U2 Amplifier
100-102	CO	0	3.0	1 MΩ	1 MΩ	1 MΩ	1 MΩ	MCP6042
100-601	SO2	200	3.0	1 MΩ	750kΩ	1 MΩ	1 MΩ	MCP6042
100-401	O3	-200	3.0	750kΩ	1 MΩ	1 MΩ	1 MΩ	MCP6042
100-701	NO2	-200	3.0	750 kΩ	1 MΩ	1 MΩ	1 MΩ	MCP6042

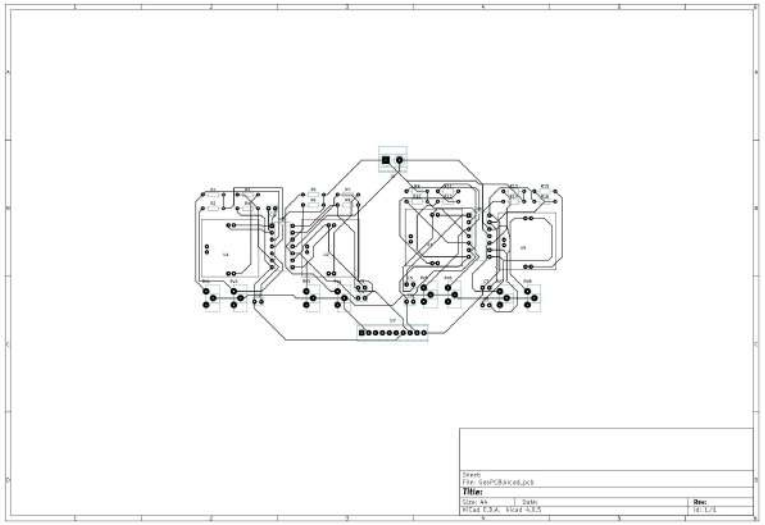
KiCad files

The following link contains all of the kiCad files that were used to create the PCB.

The file with extension ‘.kicad_pcb’ was submitted to Oshpark.com for manufacturing.

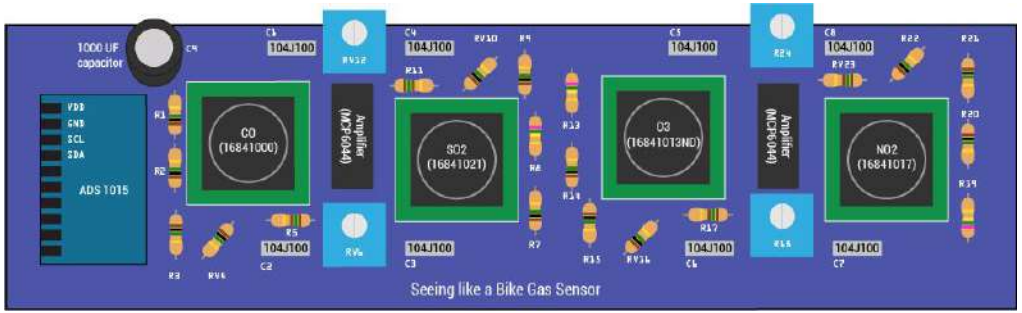
To edit the design from KiCad select open new project. Navigate to linked folder and select GasPCB.pro.

https://drive.google.com/drive/folders/0BztSVaXi7Gn-Y2NFSF9uLTR5YUk?usp=sharing



Resistors Placement

Resistor Value	1 MΩ	750 KΩ	150 Ω	500K pot
Board Position (R)	1-4, 7, 8-10, 14-16, 20-22	8, 13, 19	5, 11, 17, 23	6, 12, 18, 24



Resistor Values (in ohms)

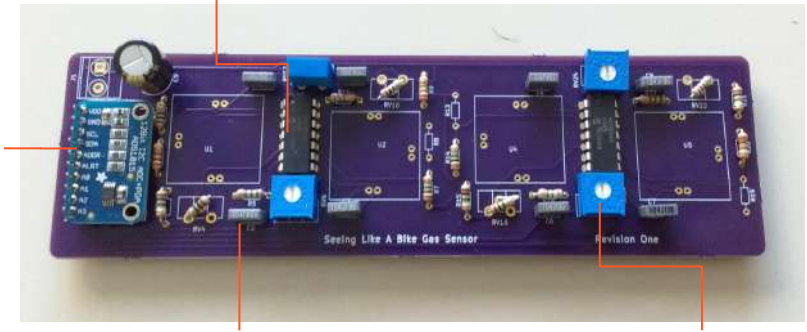
1 MΩ 750k Ω 150 Ω

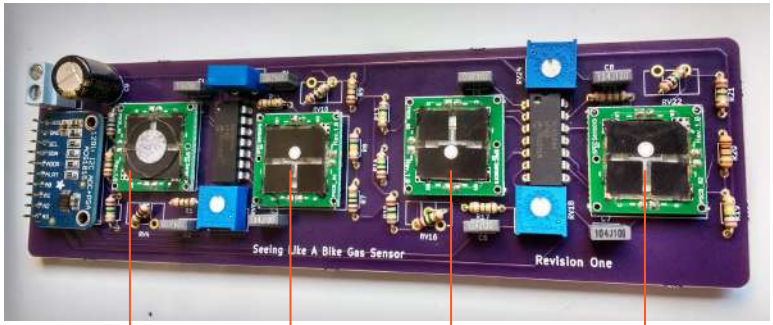
Each analog sensor has 4 resistors and an amplifier. We selected the resistor values and the amplifier based on the SPEC documentation. The fifth resistor on every circuit is 150 Ω and serves as a current to voltage converter.

Other Parts

The two amplifiers (**MCP6044**) are between the sensors.

Insert the ADC (**ADS1015**)





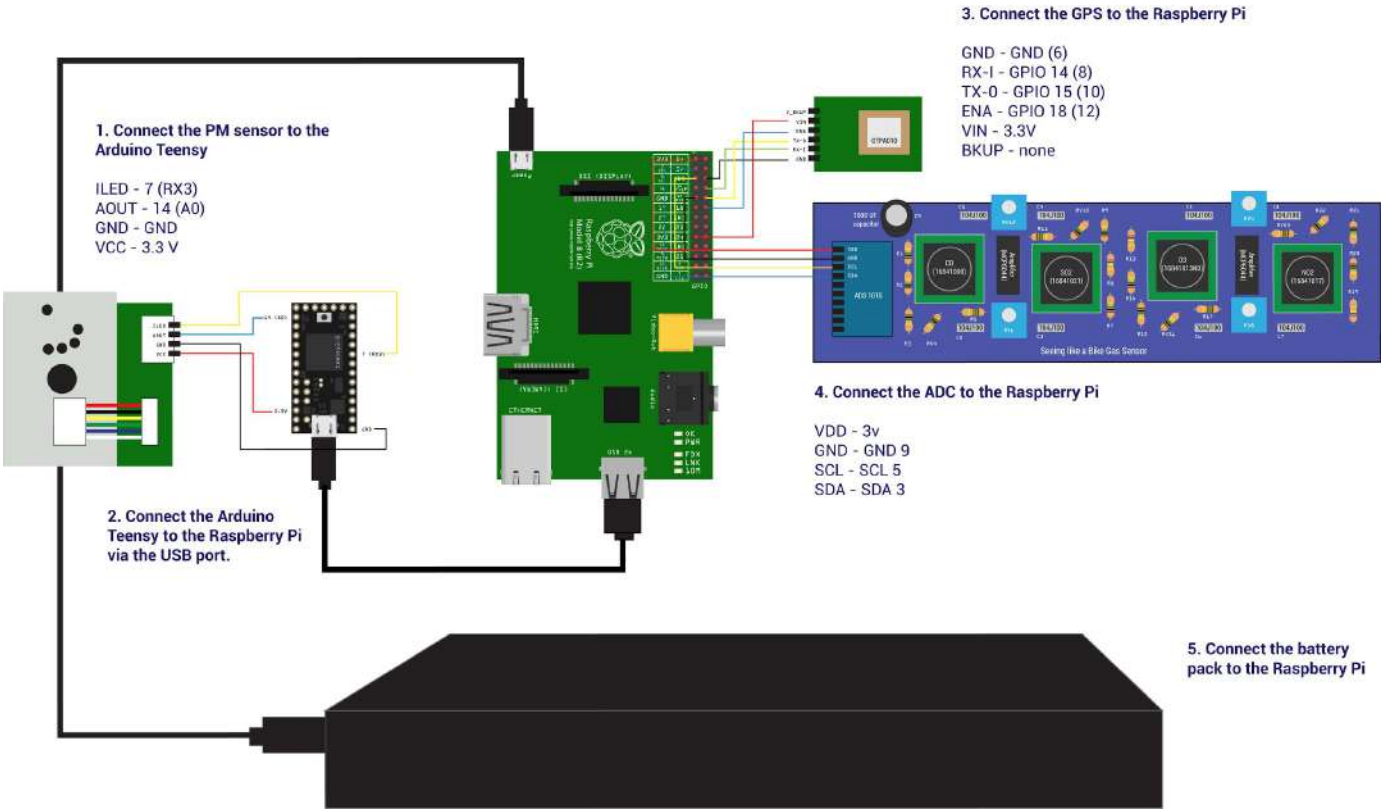
CO SO2 O3 NO2

Insert the eight **0.1 UF** capacitors into locations C1 to C8, and the **1000 UF** capacitor into C9.

Attach the four potentiometers (**500k ohm**) into RV6, RV12, RV18 and RV24.

Assembly

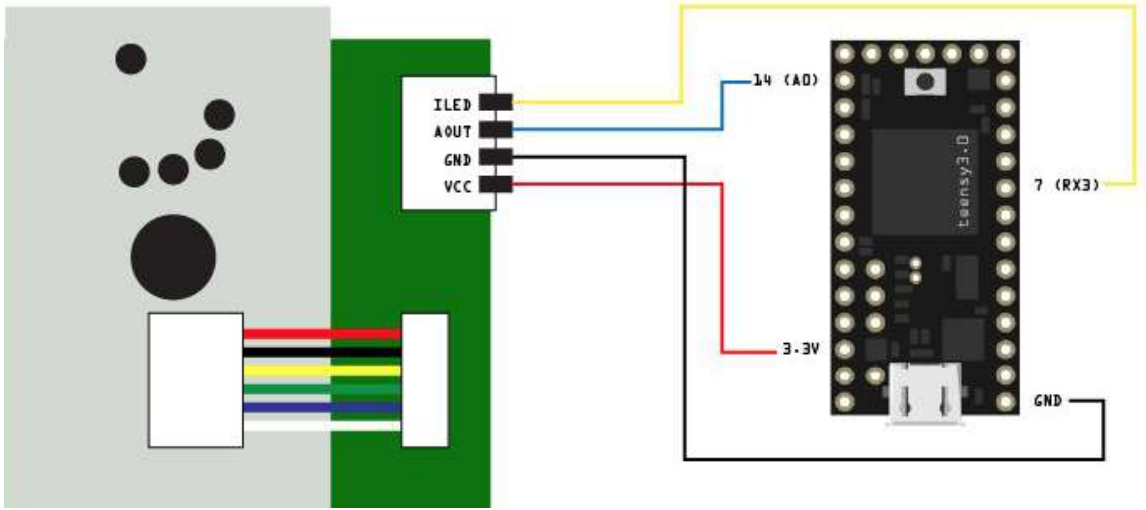
Master Schematic



Connect the PM sensor to the Teensy

Pin Connections

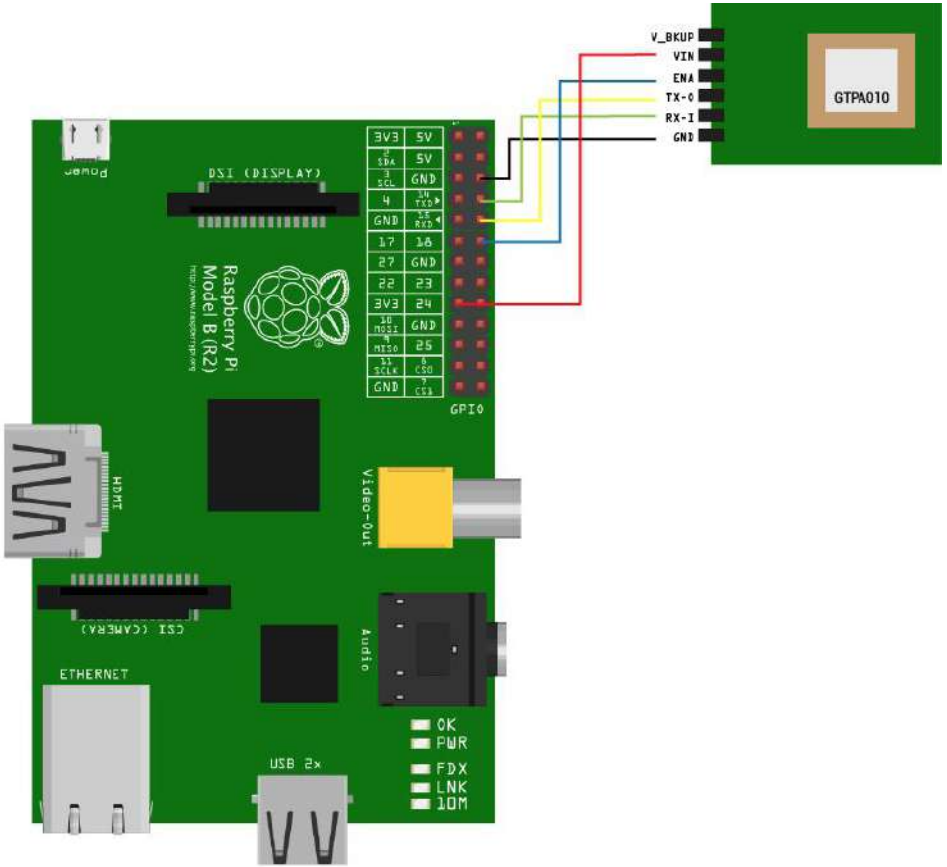
ILED - 7 (RX3)
AOUT - 14 (A0)
GND - GND
VCC - 3.3 V



Connect the GPS to the Pi

Pin Connections

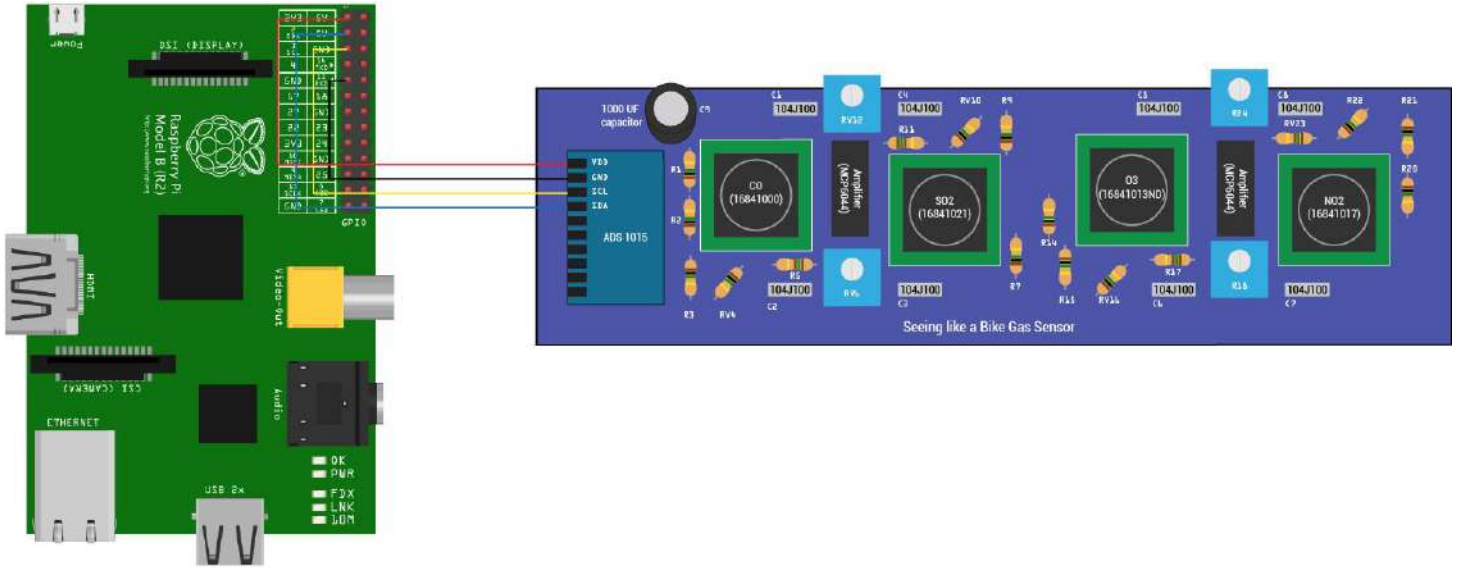
GND - GND (6)
RX-I - GPIO 14 (8)
TX-0 - GPIO 15 (10)
ENA - GPIO 18 (12)
VIN - 3.3V
BKUP - none



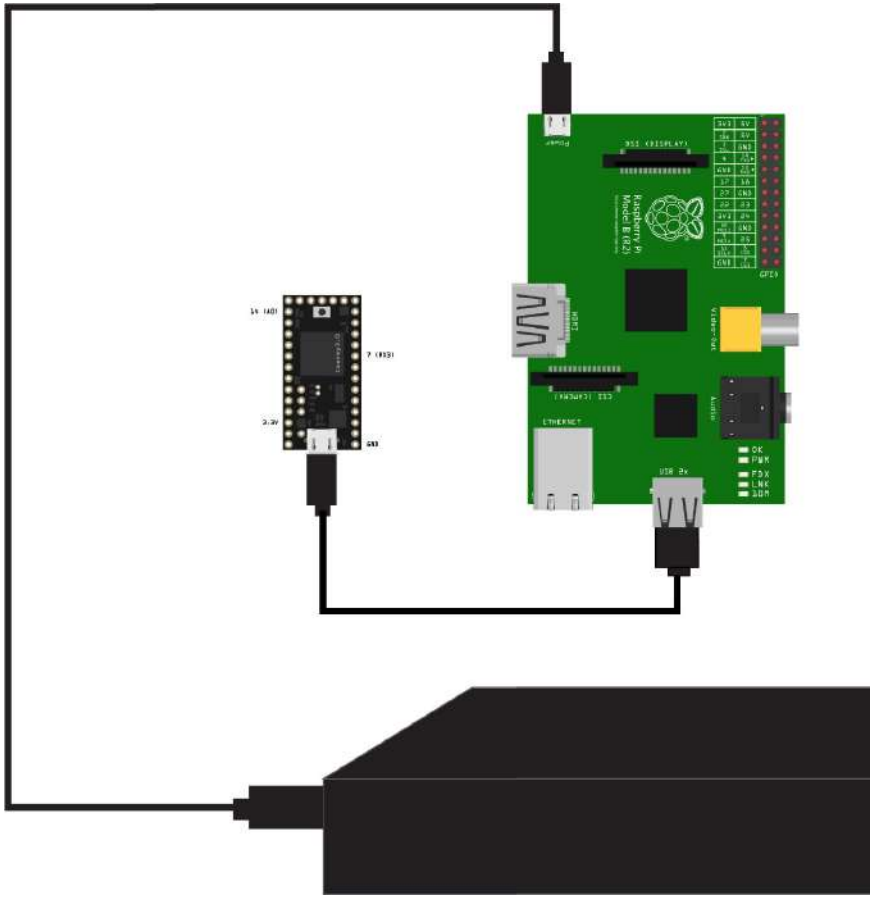
Connect the ADC to the Pi

Pin Connections

VDD - 3v
GND - GND 9
SCL - SCL 5
SDA - SDA 3



Connect the Pi to the Teensy and Battery Pack via USB cables



Running the code

How to run python script AirMaster.py

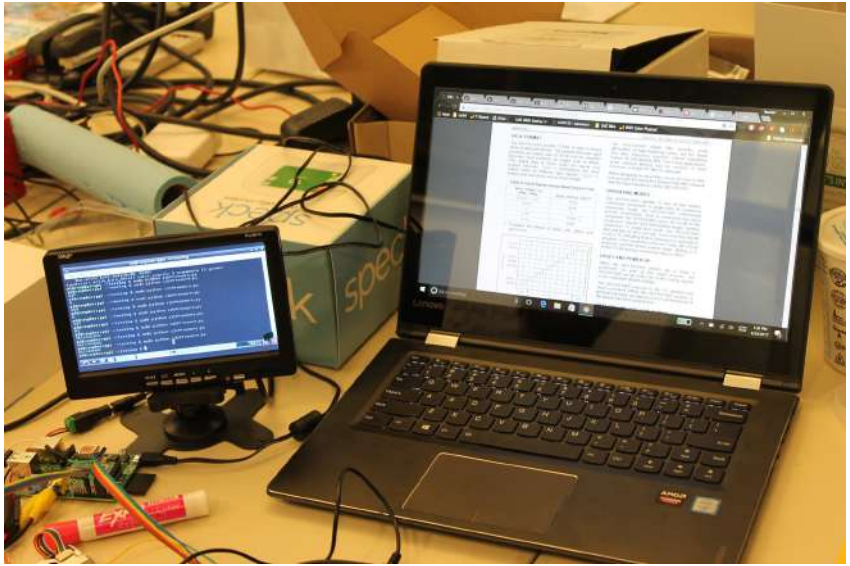
‘AirMaster.py’ is the script that will control and gather data from all of our sensors.

This script will be added to rc.local on the RPi and will run on boot without any need to manually run it.

The code will read from a GPIO pin connected to a switch. Once that switch is triggered the data capture will run, sending data to the server.

To manually run the script navigate to the folder containing AirMaster and type: ‘sudo python AirMaster.py”

The script is designed to have no output. All data gathered will be sent to the server.



Running Code (AirMaster.py)

```
############################
### Particulate Matter ####
############################
import serial
import time
import json
import requests
import ast
# Import the ADS1x15 module.
import Adafruit_ADS1x15

USBser = serial.Serial('/dev/ttyACM0', 9600)
url = 'http://192.168.1.10:5000/gas'
headers = {'Content-type': 'application/json'}
def getpm():
          USBser.flushInput()
          USBser.flush()
          read_data = USBser.readline()
          #data is already in json
          pmdata = ast.literal_eval(read_data)
          return pmdata['data']

############################
########## Gas Array ########
############################

# Create an ADS1115 ADC (12-bit) instance.
gasArray = Adafruit_ADS1x15.ADS1015()
GAIN = 1

#Low limit for each sensor @ Vx ppm = 0
VxCO = 818
VxSO = 823
VxO = 812
VxNO = 810
```

Download and import these Python libraries

Lowest values based on the Helium test (p. 29)

```
#amount of current output for each ppm
COi = 0.00000000475
SOi = 0.000000025
Oi = 0.000000032
NOi = 0.000000040

def getgas():
          # Read all the ADC channel values in a list.
          values = [0]*4
          for i in range(4):
          # Read the specified ADC channel using the previously set gain value.
          values[i] = float(gasArray.read_adc(i, gain=GAIN))
          #compute conversion
          values[0] = (values[0] - VxCO)/(500000*COi)
          values[1] = (values[1] - VxSO)/(500000*SOi)
          values[2] = (values[2] - VxO)/(500000*Oi)
          values[3] = (values[3] - VxNO)/(500000*NOi)

          gasarray = {"co":round(values[0], 2), "so2":round(values[1], 2),
"o3":round(values[2], 2), "no2":round(values[3],2), "pm":getpm()}
          return (json.dumps(gasarray))
##        print ("{'Sensor':'GasArray CO', 'data':  %.2f'}\n" %values[0])
##        print ("{'Sensor':'GasArray SO', 'data':  %.2f'}\n" %values[1])
##        print ("{'Sensor':'GasArray O', 'data':  %.2f'}\n" %values[2])
##        print ("{'Sensor':'GasArray NO', 'data':  %.2f'}\n" %values[3])

# Main loop.
while True:
          print str(getgas())
          response = requests.post(url, data=getgas(), headers=headers)
          print(response.content)
          time.sleep (5)
```

Code Explanation

Arduino

GPS

Gas Array

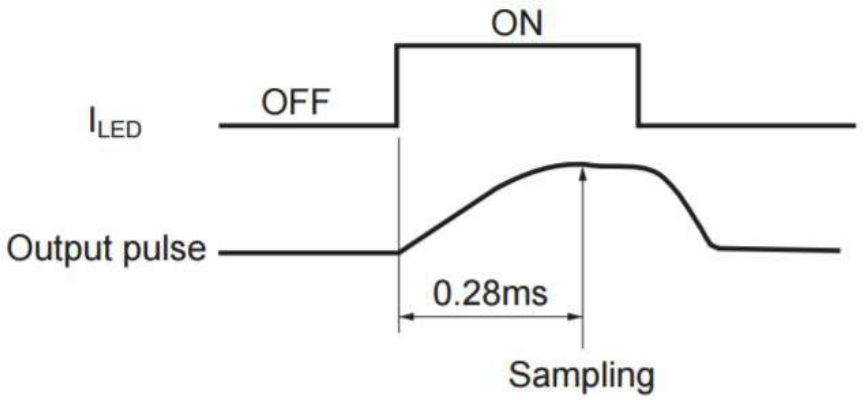
Arduino

The timing and analog sampling code runs on the arduino. After each sample the PM measure will output the value to serial.

Sends a pulse of light that is 0.32ms in duration. The analog output is sampled at 0.28ms.

The analog value is scaled to reflect ug/m^3.

Code on Pi simply reads serial value.



GPS

The GPS was connected to the serial read pins on the Pi.

The code uses the default settings of the GPS and returns useful data including:

- Speed
- Longitude
- Latitude
- Course
- UTC-time

Code available at
https://github.com/cledantec/Cycle-Atlanta-SLaB

Gas Array

Borrowed libraries from Adafruit to read from the ADC

Once values were read they were interpreted based on the data sheet.

R is the value of the pot resistor (R6)

500k was used but it easily could have been more to increase gain.

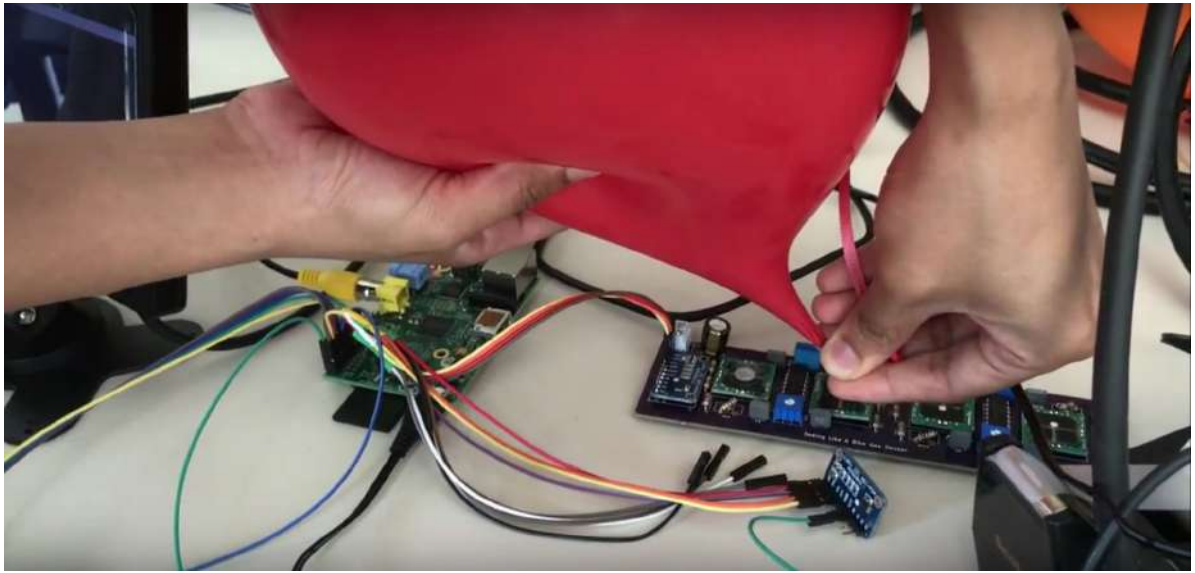
$$Vout = Vx + i*R$$

Sensor	CO	SO2	O3	NO2
Current/PPM (nA)	4.75	25	-32	-40
Detection Range	0-1000 ppm	0-20 ppm	0-20 ppm	0-20 ppm

Gas Array

In order to calibrate the gas array, we exposed them to helium. None of the sensors react with helium, providing a base state reading.

This base reading become Vx to find out the ppm measured by sensor.



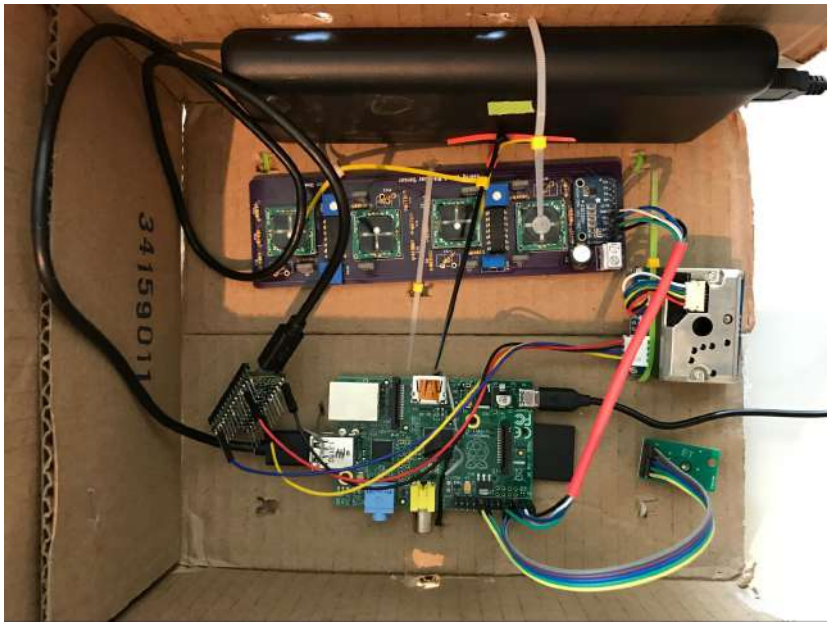
Bike Test

Bike Test

We tested our device on the bike around the Technology Square Research Building (TSRB).

The sensors are secured to a box and the bike by zip ties.

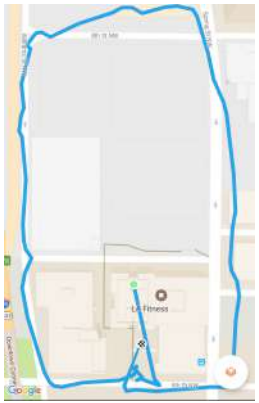
Airflow is important for the sensors to work, so we kept the box lid open and cut out the back of the box.

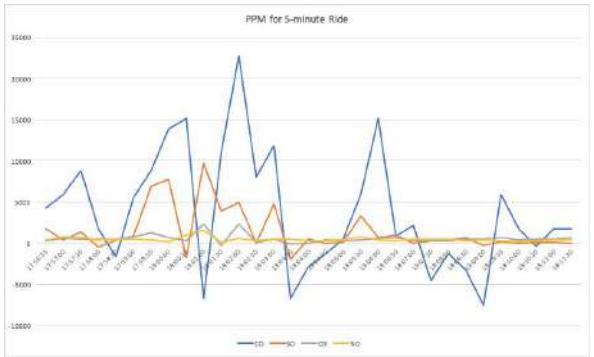


Example Data

Below is an example output data from our system in JSON format


```
{'data': {'co': 2526.32, 'o3': 875.0, 'so2': -400.0, 'no2': 550.0}, 'timestamp': '2017-04-12 21:49:11.265911', 'sensor': 'ultraviolet'}
{'data': {'co': 2526.32, 'o3': 1062.5, 'so2': -800.0, 'no2': 750.0}, 'timestamp': '2017-04-12 23:58:49.096389', 'sensor': 'gasarray'}
{'data': {'co': 2526.32, 'o3': 875.0, 'so2': -640.0, 'no2': 550.0}, 'timestamp': '2017-04-12 23:58:54.336818', 'sensor': 'gasarray'}
{'data': {'co': 2105.26, 'o3': 1187.5, 'so2': -880.0, 'no2': 850.0}, 'timestamp': '2017-04-13 00:00:18.015053', 'sensor': 'gasarray'}
{'data': {'co': 2526.32, 'pm': 23.15, 'o3': 1125.0, 'so2': -800.0, 'no2': 850.0}, 'timestamp': '2017-04-13 00:00:23.263948', 'sensor': 'gasarray'}
{'data': {'co': 2526.32, 'pm': 23.15, 'o3': 1062.5, 'so2': -880.0, 'no2': 750.0}, 'timestamp': '2017-04-13 00:01:22.085993', 'sensor': 'gasarray'}
{'data': {'co': 2526.32, 'pm': 24.23, 'o3': 1062.5, 'so2': -880.0, 'no2': 800.0}, 'timestamp': '2017-04-13 00:01:29.054935', 'sensor': 'gasarray'}
{'data': {'co': 2526.32, 'pm': 24.23, 'o3': 1062.5, 'so2': -720.0, 'no2': 850.0}, 'timestamp': '2017-04-13 00:01:36.060557', 'sensor': 'gasarray'}
{'data': {'co': 2526.32, 'pm': 24.23, 'o3': 1062.5, 'so2': -720.0, 'no2': 850.0}, 'timestamp': '2017-04-13 00:01:43.060620', 'sensor': 'gasarray'}
```



We sampled data from a 5-minute bike ride. The sensors were able to receive great amounts of data per second.

However, because of the short duration of the ride, we are unable to determine the accuracy of our data and see correlations between sensors.

Comparison with other sensors

Industrial Sensors

We visited a former EPA installation that houses industrial air quality sensors.

Recommendations:

- Low cost air quality sensors as generally inaccurate
- Need multiple hours to establish accurate readings
- To calibrate sensors properly, it is best to run a minimum 24 hour sample with industrial sensors

Measures are in parts per billion. Ours are in parts per million. Our values are also extremely high compared to the industrial sensors.

Commercial Sensors

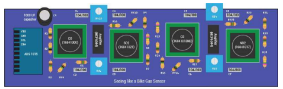
Air Beam

Speck 2

Air Quality Egg

We compared our sensors to commercial Air Quality sensors to compare features, cost and accuracy.

Features and costs analysis



	AirBeam	Speck 2	Air Quality Egg	Seeing Like a Bike
PM units	µm/mg3 (microgram per cubic meter)			
Nitrogen dioxide	No	No	Yes	Yes
Sulfur Dioxide				
Ozone				
Carbon Monoxide				
Humidity	Yes			
Temperature				
Pressure	No			
Price	$250.00	$199.00	$840.00	$200.00-$300.00

Analysis of Commercial Sensors

The EPA has compared commercial sensors with regulatory monitors. Speck & Air Quality Egg have mixed reviews in terms of accuracy.

The Air Quality Egg is less robust than the others, but can be modified as it is Arduino-based. Airbeam seems to have the best correlation, portability and ease of use in previous evaluations and our experience. However, it is still more expensive and has fewer features than our build.

For projects that want to quickly and easily measure fewer factors, the Airbeam is a superior choice. For projects that have limited funds and want to collect multiple types of data, our build is preferable.

Reflections/ Recommendations

Reflections/Recommendations

- Low-cost sensors are generally inaccurate, and thus challenging to calibrate.
- In general, the system needs to be tested for longer durations. It was recommended to turn on the sensors for at least two hours to properly calibrate (much longer than an average bike ride!)
- The GPS unit must acquire a signal before it can be used. A small battery to provide power while system is off my help GPS maintain signal acquisition.
- Overall, our Air Quality Kit was able to produce more types of data than other open source projects (AirPi, Airbeam, etc.). Our main contribution is the PCB design, which simplifies the wiring and makes the more complicated analog gas sensors easier to deploy.

Thank You!

Seeing Like a Bike (LMC 6650)
Project Documentation

Lorina Navarro
Reuben Fishback
Oriana Ott