Aerotron

DIY mobile air quality monitoring device

Technical guide

Tomislav Mamić

MIKROTRON d.o.o.

Zagreb, Croatia

July 2016.

Table of Contents

Overview

Aerotron is a wide array of sensors hooked up to an Arduino and some power circuitry, neatly tucked away in a 3D printed box. To make things more interesting and convenient for the user, an esp8266 module provides the necessary tools to make Aerotron an IoThing.

Arduino UNO is the brain of Aerotron - easy to use and with plenty of IO available, it turned out to be a great choice for our prototype.

Power circuitry is possibly the trickiest part of Aerotron, as multiple devices with varying power demands are crammed in one box. The prototype uses a 4400mAh Li-Po battery as its power source and operates roughly 2 hours. Such a battery provides as much current as one would like, but its voltage is both too low (roughly 3.7V) and varying depending on how charged it is.

To deal with this, we feed the battery (through its charging and protection circuit) into a DC-DC step-up converter which outputs roughly 8V, which is enough to operate a standard 5V regulator. Such a regulator is found on the Arduino power shield which provides stable 5V both for the Arduino and all our 5V sensors.

Last but not least, as our esp8266 requires 3.3V, we power it from Arduino's on-board 3.3V regulator. Ideally, a logic level converter would be used between esp8266 and Arduino's serial pins, but we don't use one because we are rebels (and it works just fine).

MQ sensors are a family of analog gas sensors with a variable resistor element. Their responses are notoriously non-linear, but in certain ranges they can provide decent concentration readings. Another thing they are notorious for is severe lack of good test documentation, so one can find that what these things react to is often not just one but several gases, some of which may or may not be mentioned in the datasheets. Nonetheless, we tested and roughly calibrated all of them to somewhat satisfying results.

MQ7(CO sensor) is the most important one from the bunch, so we tested its operation extensively. So far, the average readings agree with what we've been seeing on the official air quality monitoring stations which is very encouraging.

MG811($C0_2$ sensor) is another interesting analog piece of hardware. Unlike the MQ series which uses a variable resistor, the MG acts as a voltage source whose output requires amplification. To make things more interesting, it requires 6V to operate properly. Luckily, we managed to find a lovely module from DFRobot which does both step-up and signal amplification for us. Props to guys at DFRobot for putting this one together! Its response to CO_2 changes is fast and roughly matches what we expected, but calibrating this one correctly is tricky as we have nothing to compare it to.

PPD42(PM sensor) is the poor man's version of a particle counter. It uses an IR LED to shine light on dust particles which a photodiode then picks up through a lens. Signal from the photodiode is not quite the number of particles nor is it digital, so an on-board filtering circuit turns it into something that correlates with the number of particles observed. This is further sent out as a digital signal which stays high, but outputs 10-90ms intervals of low signal as particles are observed. Measuring the total percentage of low time in a longer interval can be used to estimate PM levels.

Parts list

Electronics

- Arduino UNO
- Power shield (link)
- 4400mAh Li-Po battery (link)
- Charging circuit (link)
- Step-up converter (link)
- MQ7 (link)
- MQ131 (link)
- MQ135 (link)
- MQ136 (link)
- MG811 (link)
- Shinyei PPD42 (link)
- DHT11 (link)
- esp8266 (link)
- power connection board (this one is custom-made for prototype, sorry!)

Misc

- 3D printed box (4 parts, model in git, approx. printing time 12h)
- 18 pcs - 3x8 screw
- 4 pcs - corresponding nut
- 5 pcs - 3xanything-longer-than-8-will-do screw
- a whole lot of wires

Assembly

The easiest procedure to put Aerotron hardware together is described. Please refer to additional documents for logical layout of the device, as well as detailed wiring schematics.

Preparing the power circuitry before you begin is a good idea. Make sure the potentiometers on step-up converter and the power shield are set up correctly so that those devices output 8V and 5V respectively when hooked up to the battery.

Soldering the necessary wires and/or parts should be done before putting anything in the box. These may include power wires, a custom board for sensor power connectors as is the case with our prototype or any other connections you wish to solder. You might wish to change things up sometime later, so we recommend using standard breadboard wires for pretty much anything but the main power wires. Low quality breadboard wires can often have very little copper inside which makes them dissipate significant amounts of power at higher currents. We don't want that.

Putting the components in place comes next. We recommend starting with the esp8266 and your custom power connector board. Continue onto Arduino, then add PPD42, MG811 and step-up converter in no particular order. Only 2 3x8 screws are enough to hold the MG811 module in place. From there on, DHT11 and MQ sensors, also in no particular order. It's advisable to think about ways of neatly organizing your wires at this stage. We have found it helpful to hide some wires under the components to reduce clutter. Ultimately, even if everything is cluttered and messy, as long as the components are firmly put in place, none of it will matter when you close the lid. What happens in the box stays in the box.

Wiring all the components together. Hook the power output of step-up converter up to the power shield's input. Hook the power shield's output to your sensor power connectors, 5V line (remember, there is a 3.3V device as well, we don't want to mess it up). Hook the Arduino 3.3V output to the power connectors, 3.3V line.

Connect the esp8266 module's power to the 3.3V section. Note that one of its logic pins is supposed to be hooked up to 3.3V as well in order for the module to start properly. Hook up its serial pins to digital pins 6 and 7 on Arduino. In software, we open a SoftwareSerial class which operates on those two pins to talk to the esp.

From here, we enter a loop.

While (sensors remaining > 0)

    select random unconnected sensor

    hook its power pins to the power connector board

    hook its signal pin to the Arduino (more on that later)

    consider it connected

When the loop terminates, your device is almost entirely hooked up. Notice that we have not yet connected the power input of our step-up converter to anything. This is important as you'll want to push the wires/connector through the little square hole in the box lid *before* you close it.

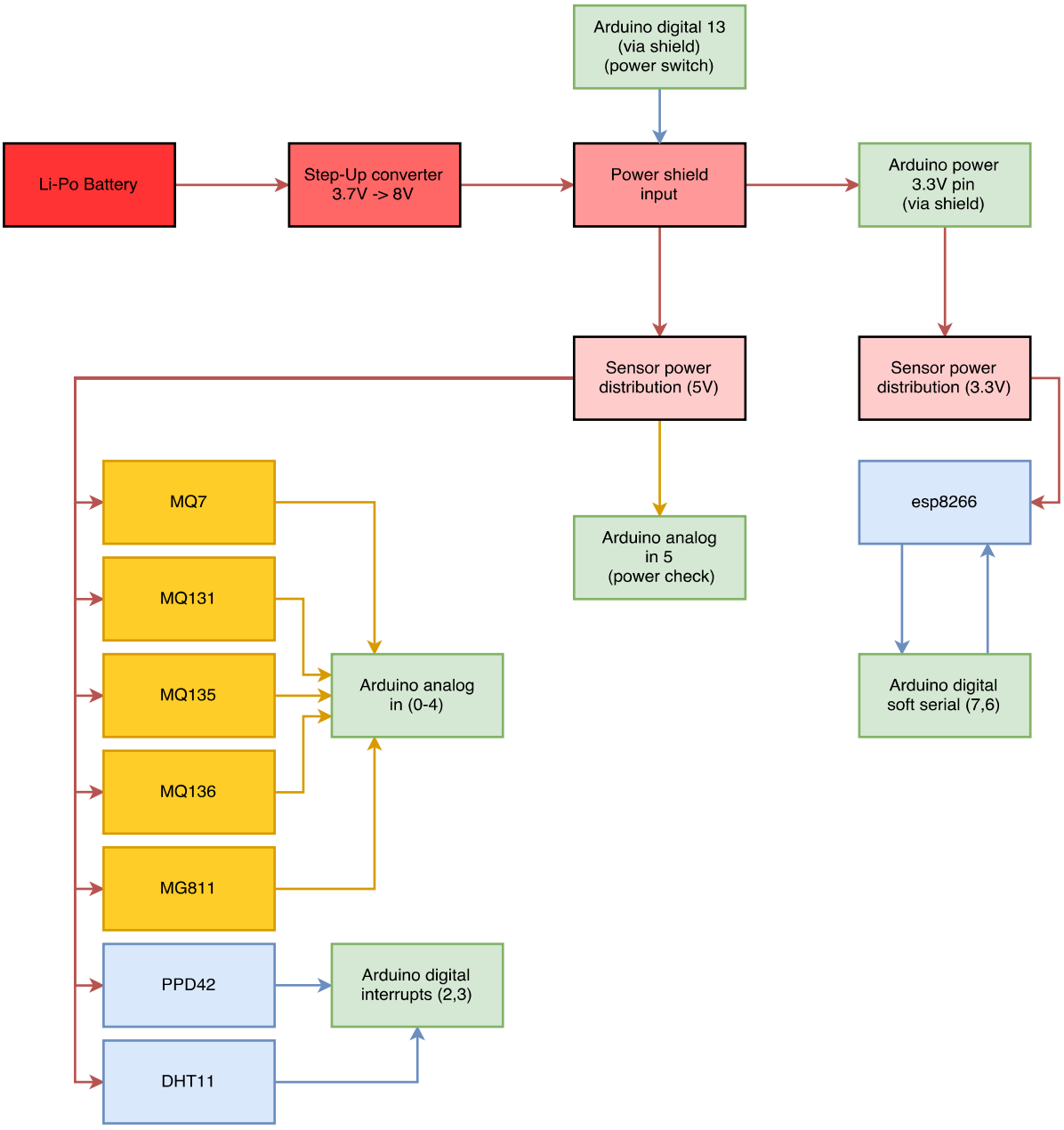
Closing the lid is to be done only after you've attached the battery compartment to it. This is done with 4 3x8 screws and bolts. Make sure the square hole in the compartment aligns with the same one on the lid so you can connect the battery circuit to your step-up converter and power up the device. Ideally, this is where you'd put an on-off switch but our prototype doesn't have one!

Quick reference

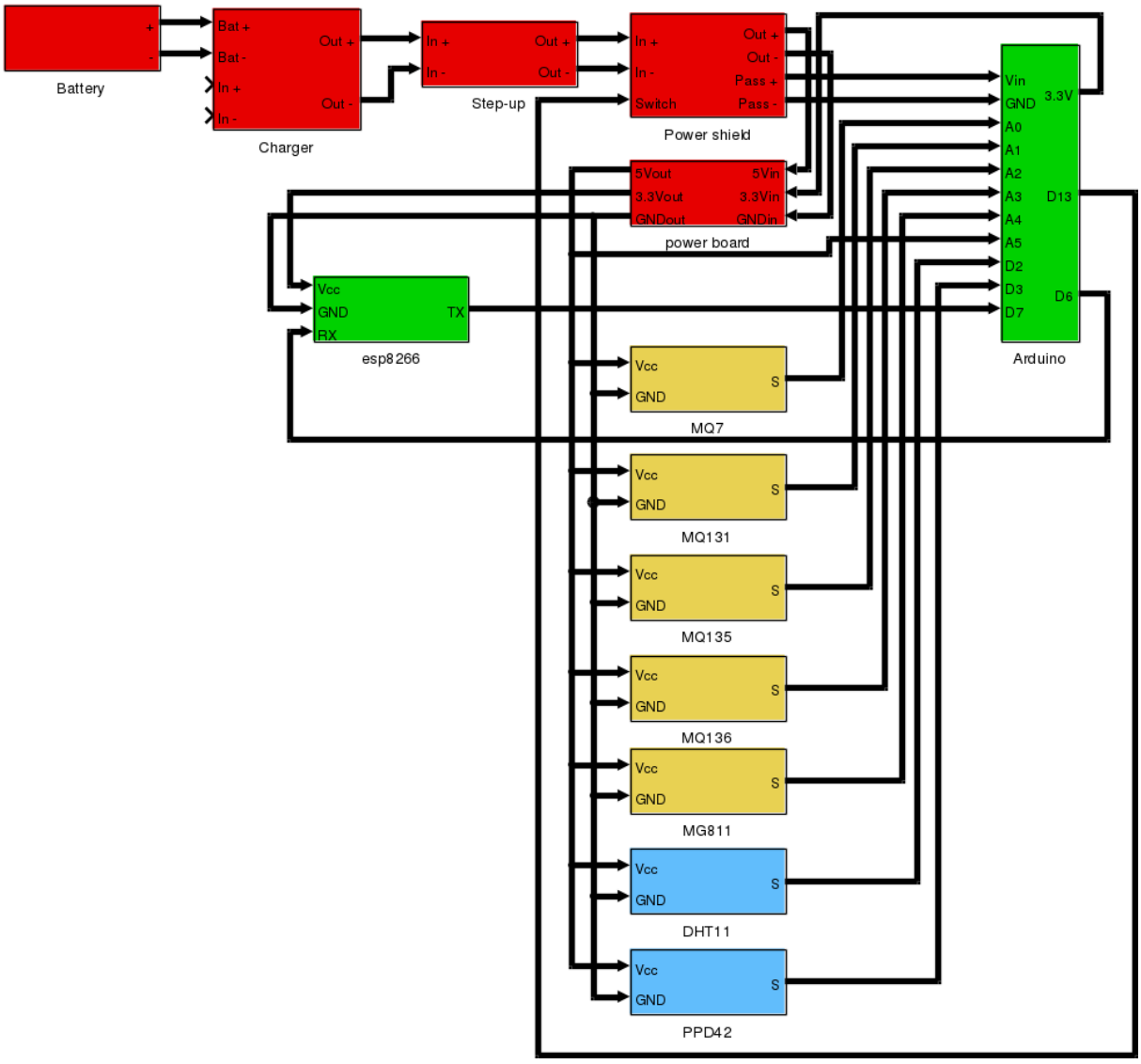
This is pin mapping for Arduino UNO:

MQ7	A0 (in)
MQ131	A1 (in)
MQ135	A2 (in)
MQ136	A3 (in)
MG811	A4 (in)
power board sens.	A5 (in)
PPD42	D2 (in)
DHT11	D3 (in)
Power shield switch	D13 (out)
RX from esp8266	D7 (in)
TX to esp8266	D6 (out)

Logical overview



Wiring


+
-
Battery
Bat +
Bat -
In +
In -
Out +
Out -
Charger
In +
In -
Out +
Out -
Step-up
In +
In -
Switch
Out +
Out -
Pass +
Pass -
Power shield
5Vout
3.3Vout
GNDout
5Vin
3.3Vin
GNDin
power board
Vcc
GND
RX
TX
esp8266
Vcc
GND
S
MQ7
MQ131
MQ135
MQ136
MG811
DHT11
PPD42
Vin
GND
3.3V
A0
A1
A2
A3
A4
A5
D2
D3
D7
D13
D6
Arduino

Measuring air quality

Carbon monoxide is measured by the MQ7 sensor (datasheet). Its variable resistance is exponentially related to CO concentration. Since it operates as a simple voltage divider, by varying the load resistance, one is able to adjust the operating range. We've let others do that for us, namely Sainsmart. So far, we are happy with their MQ modules, but not so much with their MG811 which they've managed to mistake for an MQ type it seems, as the sensor sits on an identical circuit to those of the MQs. A rather disastrous mistake which they may or may not have fixed in the meantime.

The gist of using this (and all the other MQ ones) sensor is rather mathematical - get the numbers from the datasheet into your code and work out the concentration from your analog inputs. When all the equations are set, you're left with one free parameter - the value of variable resistance in some known concentration of CO. This must be calibrated, and it can get tricky without fancy equipment.

Ozone is measured by the MQ 131 (datasheet) and is one of the sensors we've tested just enough to make sure it behaves as we'd expect it to behave. The basic operation is analogous to what has been said above, only the gas is different. It is worth noting, however, that the sensor responds similarly to Cl_2, and to a degree to some nitrogen gases.

Benzene is what we'd love to measure using MQ135 (datasheet), but we've yet to do some serious testing. Its principle of operation is the same, but the range of gases which may trigger its response is diverse which led us to be skeptical about the results.

Hydrogen sulfide is something you'd hardly miss in large concentrations as its smell makes your eyeballs drop out. Nonetheless, we are planning to measure its concentration using MQ136 (datasheet). Since H_2S isn't a gas we can safely acquire and use for testing, this one is also on hold for now.

Particulate matter or tiny dust particles is something we're highly interested in. The sensor we are using, Shinyei PPD42 (a rather poor datasheet) detects anything larger than a micron which is slightly smaller than what you'd find in official air pollution reports. However, its output still correlates highly with concentration of larger particles and after extensive testing, we learned to love it.

Software

Since the device is still in prototype phase, the code running on it has some testing features and running into pieces of dead code from time to time should not be surprising. In this section we describe the principle idea of how device operation will be organized.

Initialization - on every power-on, the device initializes code, sets up pins, checks its calibration settings, connectivity and sensor power status. If capable of normal operation, the device immediately jumps into measuring loop, else additional steps are taken.

Autocalibration - kicks in if the device is unable to read its own calibration settings. It assumes that device is in clean air and begins a heat-up process. After several minutes of doing nothing to let the sensor heaters do their job, Autotron will repeatedly measure all of its sensor outputs and assume that they correspond to the clean air state (predefined in the firmware). This form of calibration is just a means of providing rough output from an uncalibrated device to check its operation. Users are encouraged to play with it on their own, and any devices used on the field should be checked against known measurements and adjusted accordingly.

Main loop - schedules measurements and communication. The prototype revolves mostly around sending a stream of CSV formatted measurements via serial to a PC. A python script on the PC logs these measurements. This is what we use to test sensors. Most sensors are measured every 2 seconds, but PPD42 requires constant polling and one has to make time to communicate those measurements to other devices or a cloud service. Originally we wrote code for an interrupt-based PPD42 measuring function, but this gave rise to some timing problems all over the place, so eventually we decided to simply check the state of its input at most every 5ms whenever the device is not doing anything else. We can get away with this for a multitude of reasons, two of the most prominent being the fact that device is free for a good amount of time and the nature of PPD measurements is such that having continuous measurements isn't a priority as long as one can take enough of them in a given time interval, and as long as they're neither too close nor too far apart.

Wifi - Ideally, Aerotron can connect to the internet during its operation and dump all the data it logs onto the cloud. To that end, if possible, the device will hook up to your wireless network and attempt to contact the cloud every once in a while. Depending on how much time this communication process will take, we may opt to send average values over more on-the-spot measurements to reduce the time wasted trying to negotiate with the server and do more measuring work.

The esp can be a mean little weasel. Depending on where you got your esp module from, its firmware may range from utterly useless to just what you always wanted. We were unlucky to get some really old version of firmware that lacked over half the functionality exposed through AT commands, most notably the ability to change the device baud rate. Since SoftwareSerial on Arduino UNO can only go that fast, talking to esp at 115200 baud was a hit and miss thing, mostly miss. This being a problem, we spent some time playing with the module and came up with a really easy process of getting the firmware onto them:

1. upload an empty sketch to your Arduino
2. connect the esp module serial to the exposed Arduino serial (this will fool your PC into thinking that it's talking to the Arduino when in reality the esp module will be doing all the talking)

3. ground the pin on esp which forces it into programming mode
4. use the official Espressif firmware download tool (here you go) to get your code onto the esp

When you do it for the first time, it may seem a bit sketchy at best, but after playing around with it a little, the process becomes pretty easy and fast. There are many firmware versions out there with slightly different behaviours, but we recommend using one of the official Espressif versions found in their SDK distributions (link).

Future development

Gathering lots of test data for all the sensors in order to see which ones are really useful.

Refining the code from prototype testing into a device anyone can use.

Connecting to the cloud to make measurements publicly available. We are currently looking into Adafruit IO.