ArduECO: cheap air quality control for cities

Voinea Stefan Ciprian
University of Padova, Italy
Department of Pure and Applied Mathematics
stefanciprian.voinea@studenti.unipd.it

***Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.**

***Index Terms*—Arduino, embedded, air-quality**

I. Introduction

More and more people around the world have started to understand the importance of air quality and how much having clean and pollution-free air can influence our lives, both on the small and large scale. To have clean air, all of us have to do something to avoid polluting it.
The Government of Luxembourg has set a good example becoming the first country to offer nationwide free public transport to the citizens [1].
For short commutes, vehicles like the classic bike [2], the infamous skateboard and the scooter have started to arise and conquer cities, either in their battery-powered or old-school human-powered.
Since not everyone necessarily owns one of these green methods of transportation, companies like *Mobike* [3] and *MiMoto* [4] have seen the opportunity to enter the market of shared transportation, proposing a *pay-per-use* solution for bikes, electric scooters and other similar vehicles. Each company has its own application, network infrastructure and smart devices aboard their vehicles, gathering data like GPS (Global Positioning System) position, time spent by the user, parking spot, etc. to send it in the cloud and compute information like cost of the ride and charging it to the user.
All this falls under the *IoT* (Internet of Things) paradigm, which has become a well-described market with new ideas and business opportunities being presented every day.
Among the data collected by these companies, none is about air pollution.
In this article, I describe *ArduECO*, a wireless device based on an Arduino-like board capable of gathering data from previously named vehicles and sending it to the cloud, to be processed and displayed.
This paper is organized as follows:

- Section 2: Background and description of the problem
- Section 3: State of the art on smart devices that allow tracking pollution
- Section 4: implementation of the proposed solution
- Section 5: Conclusions

II. Background problem

Background
spiegare quali sono le particelle di inquinamento nell'aria
spiegare quali sono i sensori presenti sul mercato che possono rilevarle
tabella con sensori mq e differenze
come mai la scelta di implementarlo in quel modo

III. State of the art

State of the art non solamente della letteratura ma anche di quello che viene offerto sul mercato

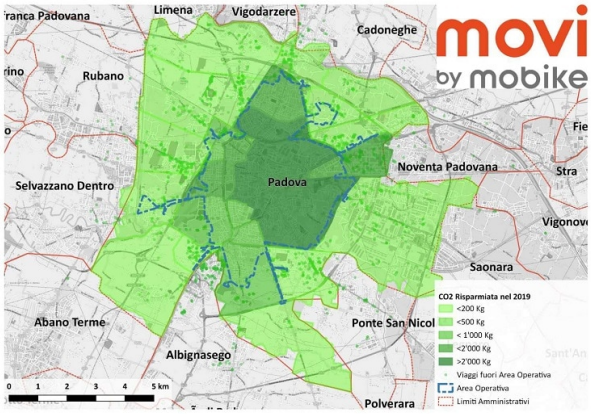

Fig. 1. Example of a figure caption. [**?**]

IV. Proposed solution

ArduECO is a wireless IoT device that represents one of the possible combinations of open source hardware and software. This wireless device is able to detect substances in the air and send the data in the cloud over Wi-Fi, and combined with data from other similar devices detect the level of pollution in the air.
The schematics for this project have been designed using Fritzing, a CAD (Computer-Aided Design) software for the

design of electronics hardware [5] and can be seen in Fig. 2. Both code and schematics for this device are available on the GitHub repository "ArduECO" [6].
Schematics for other components can be easily found online from various sources and vendors: here, for example are the schematics for the MQ7 sensor from SparkFun [7] and the NodeMCU [8].

A. The circuit

ArduECO is composed by four main components:

- *NodeMCU*: it's the hearts and brains of the device, this board is an open-source development kit based on the ESP8266 chip that allows for prototyping of IoT devices;
- *MicroSD card reader*: this allow for collecting and keeping data cached for the current ride and permanently on a removable memory card;
- *GPS sensor*: this allows for localizing the device, it has an onboard LED that blinks when it has established communication with the satellites;
- *MQ sensor*: the MQ sensor used in this project is MQ7 but it can easily be exchanged with other sensors that use the same board and connectors.

The other components, LEDs and button, are used as I/O for interacting with the user.
In Section VI I explain how the project can be improved both in hardware and software. In Fig. 2 there is no indication of

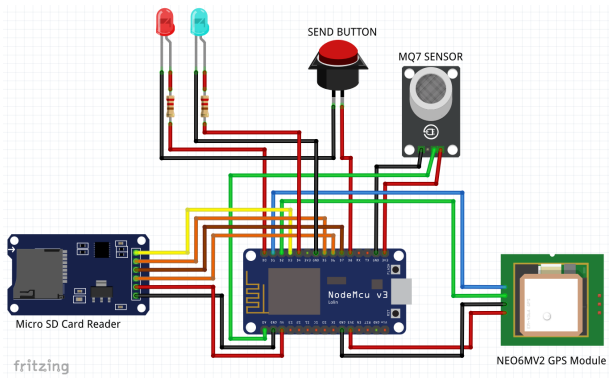

Fig. 2. ArduECO schematics

a power supply since for this prototype version it has been powered using an external battery with a standard output of 5V and 1A.

B. The Arduino software

For ArduECO I chose to use Arduino's programming language instead of MicroPython [9] because of the large community supporting it and the vast interoperability with other boards.
The main functions in the Arduino programming language are "*void setup()[]*" and "*void loop()[]*": the first one runs a single time when the device boots initializing and setting the initial values while the second one loops consecutively, allowing the program to change and respond. ArduECO.ino is the main file executed by the NodeMCU and contains the previously described functions.
In ArduECO's software, the setup function contains the instructions to correctly initialize serial communication with the GPS module, create the files in the SD card, initialize the pinouts, gather the certificates for communicating with the cloud and create a random id for identifying the ride, since each time the device boots is considered as a single ride.
An important part of the configuration of ArduECO is the file *params.json*:

```
{
        "ssid" : "",
        "password" : "",
        "in_topic" : "",
        "out_topic" : "",
        "endpoint" : ""
}
```

If this file is not present in the SD card at boot-time and correctly formatted, ArduECO will not end the boot sequence, returning an error and rebooting after 10 seconds.
This file contains configurations such network SSID and password, topics for communicating with the MQTT server and its endpoint (network address).
The loop function contains the instructions that tell the board to acquires a reading of its surroundings every 5 seconds by first detecting if a GPS fix os present and by reading the value from the analog pin connected to the MQ sensor. Each read is then appended on two separate text files (*cache_log.txt* and *perm_log.txt*) in JSON format. *cache_log.txt* is created each time ArduECO boots is related to a single session while *perm_log.txt* contains reads about each ride taken.
Here is an example of a logged loop cycle read:

```
{
        "id":"127",
        "air":"18.22",
        "lt":"45.39641",
        "lg":"11.88527",
        "dt":"50320.12173900",
        "tpc":"ardueco_proto_01_OUT"
}
```

The external LEDs are used to communicate when the setup sequence has finished correctly with errors as well as for indicating whether the SSID specified in the *params.json* file is in range and if ArduECO is transmitting data.
When the button is pressed, ArduECO searches the list of networks in range for the specified one and, if present, connects to it. Afterwards it establishes communication with the MQTT server specified by the endpoint, and sending it a message with the number of messages that will be sent.
Each row of the cache file is then transmitted to the server.
At the end of this sequence the cache file is deleted and created again awaiting new reads.
The other files that implement this project contain specific functions for either setting up the device or connecting the the server.

C. The cloud

When the button is pressed, ArduECO connects to the access point with the SSID specified in the *params.json* file and sends the contents of *cache_log.txt* to the cloud, specifically to AWS (Amazon Web Services). *IoT core* and *Lambda* are the services that are used for this project.
ArduECO sends data to the MQTT server in IoT core specified as *endpoint* in *params.js*. It connects to this server using certificates created in the IoT core console and that are loaded in ArduECO's memory, this means that they are uploaded via the Arduino IDE.
The MQTT (Message Queuing Telemetry Transport) server is designed as a lightweight messaging protocol that uses publish/subscribe operations to exchange data between clients and server, fast in data transmission [10].
This service in IoT core is connected to a serveress Lambda function written in Python, triggered each time a new messages arrives from ArduECO, which receives the JSON message as a parameter and calls a PHP page on arduECO.altervista.org, where the front end of the service is hosted. Altervista is a free website hosting platform with PHP and MySQL database. After being saved in the MySQL database in Altervista is then displayed on the homepage using Google Maps' APIs.

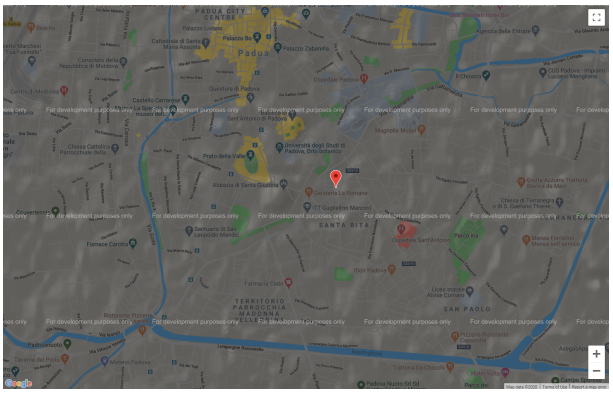

Fig. 3. Map with markers and reads from ArduECO.

Each marker on the map represents a reading and, if clicked on, shows the value read by ArduECO's sensor.
Markers are connected with different colors indicating the ride they belong to.

D. Hardware cost

Because the components used for this project are open-source, they can be found sold on various websites from many vendors. Here are some examples of prices taken from the Chinese e-commerce AliExpress [11] at the time of writing.

- NodeMCU [1]: $\sim$ 2€
- MicroSD car reader [2]: $\sim$ 0.50€
- GPS sensor [3]: $\sim$ 3.5€
- MQ gas sensor [4]: $\sim$ 1€

The total cost of these items amounts to $\sim$ 7€, considering shipping and costs of remaining components (cables, LEDs, button, SD card), it can arrive to less than 15€.
Note that the indicated prices refer to single pieces and in case of bulk purchases (at least 5 or 10 of each piece) the total cost would be much lower.

V. Results and data analysis

A. Test scenarios

1) Prototype test:
2) Real life implementation: State of the art
It make detection by method of cycle high and low temperature, and detect CO when low temperature [18]

VI. Future improvements

Since ArduECO is one of many possible implementations that this hardware and software can create, there are various improvements which could bring to a better and more stable device.

A. Hardware improvements

Many improvements can be made on the hardware: for example, the circuit described in Section IV-A has a single air sensor due to the limitation of the NodeMCU and ESP8266 chip that have only one analog input pin.
By connecting the board to an *ADC* (Analog to Digital Converter) it becomes possible to add multiple sensors expanding the analog inputs. ADCs convert an analog voltage on a pin to a digital number [12].
In order to upgrade the transmission method and use mobile data instead of WiFi (which is limited to 2.4GHz on the NodeMCU) it's possible to switch the board to an Arduino Leonardo or an Arduino Uno.
Modules capable to add mobile connectivity can be found on sites like AliExpress[5] and does not brig much additional overall cost (in this case $\sim$ 2€).
Given the small dimensions of the components, this project could be easily fitted inside of a custom made box. A solution could be to design and 3D print a custom plastic case in which fit the whole device, considering that the air sensor should remain outside of the case and the whole project could be exposed to high levels of humidity (for example in the winter season).
Power is one important issue for IoT devices. The prototype described in this essay has been powered using an external battery, but it can be coupled with other energy sources like a dynamo, so that it can be powered by movement of the wheel (in case it's installed on a bike).

B. Software improvements

The software can be improved in many ways as well. For example a control could be added to check if every message has correctly arrived at the IoT server and, in case of a timeout or an eventual error, the message could be sent again.

[1]https://it.aliexpress.com/item/32879925927.html
[2]https://it.aliexpress.com/item/32572793362.html
[3]https://it.aliexpress.com/item/32836015224.html
[4]https://it.aliexpress.com/item/4000575957425.html
[5]https://it.aliexpress.com/item/32963853961.html

C. Cloud services improvements

In the development of this project I was forced by the we server host altervista.org to connect to the database only from within

connections within its domain

i was not able to allow the lambda to insert the data directly in the database, so I was forced to create an intermediate webpage to insert that data

for this reason a possible improvement could be

VII. Conclusions and future work

This paper describes the implementation of a wireless IoT system capable to gather data from air and send it to the cloud in order to be analyzed and displayed.

References

[1] Mobilititszentral official website: https://www.mobiliteit.lu/en/tickets/free-transport/

[2] *The Vehicle of the Future Has Two Wheels, Handlebars, and Is a Bike*: https://www.wired.com/story/vehicle-future-bike/

[3] Mobike official global website: https://mobike.com/global/

[4] Mimoto official website: https://mobike.com/global/

[5] Fritzing official website: https://fritzing.org/home/

[6] ArduECO GitHub repository: https://github.com/cipz/ArduECO

[7] MQ-7 manual: https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MQ-7 Ver1.3 - Manual.pdf

[8] https://github.com/nodemcu/nodemcu-devkit

[9] MicroPython tutorial for ESP8266: https://docs.micropython.org/en/latest/esp8266/tutorial/

[10] How MQTT works: https://1sheeld.com/mqtt-protocol/

[11] AliExpress italian website: https://it.aliexpress.com/

[12] *ADC*: https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/

[13] *In bicicletta al lavoro: a Milano risparmiate 27,5 tonnellate di CO2*: https://www.ilsole24ore.com/art/in-bicicletta-lavoro-milano-risparmiate-275-tonnellate-co2-ACYBZ1FB

[14] Official Arduino website: https://www.arduino.cc/

[15] Official Raspberry Pi website: https://www.raspberrypi.org/

[16] Lua based interactive firmware for ESP8266, ESP8285 and ESP32 https://github.com/nodemcu/nodemcu-firmware
http://www.padovaoggi.it/attualita/dati-mobike-padova-10-ottobre-2019.html

[17] In arrivo anche a Padova le E-Bike a pedalata assistita: l'annuncio di Mobike In arrivo anche a Padova le E-Bike a pedalata assistita: l'annuncio di Mobike: http://www.padovaoggi.it/attualita/mobike-e-bike-dati-padova-21-febbraio-2020.html
https://arduinojson.org/

[18] *MQ-7 Semiconductor Sensor for Carbon Monoxide* specifications: https://www.pololu.com/file/0J313/MQ7.pdf