An Affordable Vehicle-Mounted Sensing Solution for Mobile Air Quality Monitoring

Pedro Santana[1] , Alexandre Almeida[1] , Pedro Mariano[1,2] ,
Carolina Correia[2] , Vânia Martins[2] , Susana Marta Almeida[2]
[1] *Instituto Universitário de Lisboa (ISCTE-IUL), Instituto de Telecomunicações, Lisboa, Portugal*
[2] *Centro de Ciências e Tecnologias Nucleares, Instituto Superior Técnico, Lisboa, Portugal*

***Abstract*—This paper presents the first prototype of the ExpoLIS system and its preliminary laboratory and field experiments. The ExpoLIS system is composed of an affordable vehicle-mounted mobile sensor network and its supporting user-centred services whose aim is to provide citizens with real-time and dense spatiotemporal air quality data. A set of preliminary static laboratory experiments and dynamic field experiments were conducted, showing that the current prototype is already able to track changes in the air quality and provide citizens with access to these events via a mobile application.**

***Index Terms*—Mobile Sensor Networks, Air Quality Analysis**

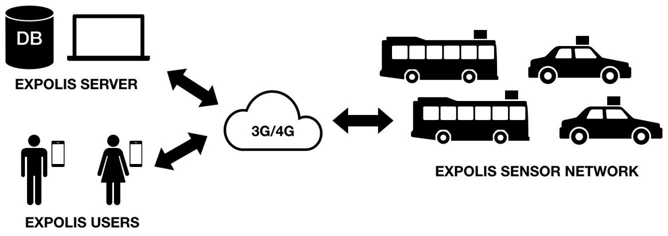


Fig. 1. The ExpoLIS system.

I. Introduction

Despite the efforts to improve Air Quality (AQ), the citizens are still exposed to levels of air pollution above the limits set by the European Legislation and World Health Organization [1]. Monitoring airborne pollutants is of utmost importance to reliably assess the impacts of air pollution on human health. While it is current practice to monitor air pollution with networks of static stations, their acquisition and maintenance costs limit the number of installations, and thus the ability to monitor AQ in the entire city. In the recent years, there have been many efforts to provide less expensive solutions for AQ monitoring. For instance, it has been shown that implementing a wireless network of low-cost sensors can increase the spatial distribution and coverage area of the monitoring systems, especially if mounted on mobile platforms [2], [3]. In fact, the use of low-cost sensors aims to complement the readings from the Monitoring Stations and provide useful information about AQ at specific locations [4], [5].

Although spatiotemporal sampling of environmental variables can be achieved by means of tele-operated and autonomous mobile robots [6], [7], costs and risks can be reduced if the sensors are moved across the city on existing public transportation vehicles. The ExpoLIS project aims exactly at exploiting the natural advantages of existing public transportation infrastructures by developing an affordable AQ exposure sensing system composed by a network of sensor nodes deployed on city buses. As depicted in Fig. 1, this network will gather in a central database real-time air pollution data to support urban planning policies, environmental scientists and transport companies, as well as to provide health-optimal routing services to the population. The implementation of the ExpoLIS system is happening in Lisbon in a partnership with the public transportation buses company (CARRIS), where the sensors are being installed.

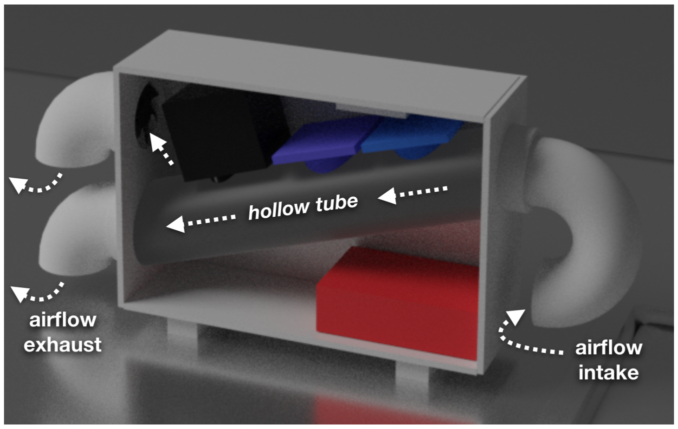
The paper is organised as follows. Section 2 describes the mechanical aspects involved on the sensor node, including its design, prototype, and dynamic behaviour analysis. Section 3 presents the hardware architecture of the sensor nodes, with particular emphasis on the rationale underlying the selection of the air quality sensors. Section 4 presents the main software components composing sensor nodes, server, and graphical user interfaces. Section 5 exposes laboratory and field preliminary static and dynamic results. Finally, some conclusions and future work directions are drawn in Section 6.

II. Mechanical Solution

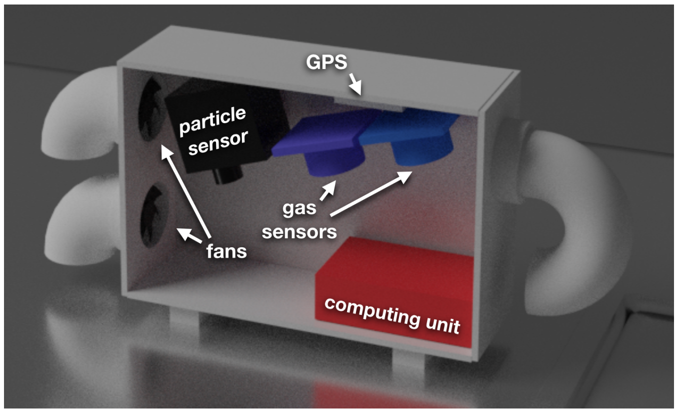
This section presents the rationale underlying the design, the prototype, and the fluid dynamics analysis of the sensor node's enclosure and air sampling device.

A. Requirements

Although the main deployment focus of the ExpoLIS project are city buses, to foster future expansion of the ExpoLIS network, the sensor nodes were designed in order to allow their installation on any vehicle, including bicycles (attached to the bicycle's frame or to the rider's backpack). Therefore, the sensor node has been designed to be as small and light as possible and to be able to operate in static and dynamic scenarios. The solution withstands the weather, while taking into account that it is not possible to have a completely hermetic solution as the sensors must be in contact with air. Finally, to allow the sensor node's replication and massive deployment as quickly and inexpensively as possible, the attained solutions relies only on low-cost consumable components.



(a) Rendering of the box's 3D triangular mesh with airflow direction represented by dashed arrows.



(b) Rendering of the box's 3D triangular mesh without the hollow tube (for the sake of the other components visibility).

Fig. 2. Renderings of the sensor node's 3D model.

B. Design

Fig. 2 depicts two renderings of a 3D model of the designed sensor node, whose bounding volume has a width, height, and depth of 38 cm, 22 cm, and 9 cm, respectively. The box has been designed assuming that it will be deployed on the vehicle's roof, away from the influence of the vehicle's exhaust pipes. To ensure that no rain droplets enter the sensor node, the air inlet has a downwards U-like shape. An inclined hollow tube, with 5 cm diameter, is connected to the airflow inlet. The inclination of this tube ensures that whatever comes in (e.g., water droplets) and condensed water are drained away. The drained elements and airflow go through this tube until they reach the air outlet, a downwards curved elbow-like hollow tube. The shape of the air outlet helps preventing water drops inside the tube when washing the vehicle. To ensure a sufficiently strong airflow through the tubes in static situations, a fan placed at the air outlet forces the air extraction. The use of fans is not necessary when the vehicle is in motion.

Inserted along the top of the inclined tube, the gas and particles sensors analyse the air flowing through them. The accumulation of dirt and water droplets on top of the sensors' sensible surfaces is mitigated by placement of the sensors facing down. There is a second airflow outlet with its own air extraction fan. This outlet extracts the air inside the box, yet outside the tube, that has been heated by the electronics and exposure to solar light. The GPS device is placed on the top face of the sensor node in order to improve sky view.

C. Analysis

To assess the ability of the sensor node to properly sample the air while in motion, a computational fluid analysis was conducted by feeding the software package OpenFOAM [8] with the 3D model presented in Fig 2. The air extraction fans were removed from the model for this analysis. The goal of removing the fans is to test the ability of the device to properly sample the air in a mechanically passive way.

Fig. 3 presents the results of the computational fluid analysis for a vehicle moving at 10 m/s (36 km/h). The figure shows a significant pressure difference between the box's frontal face (the one encompassing the air inlet) and the box's rear face (the one encompassing the air outlet), which helps building a flow of air that enters the air inlet, moves along the tube, and exits through the air outlet. The existence of this air flow can be verified by the horizontal and vertical air speed depicted in the figure. Hence, even without air extraction fans, the air flows through the sensor node as desired, provided that the vehicle is in motion with adequate speed.

The particles sensor is provided with its own embedded fan, which is used to suck air in for an internal chamber in which a laser-based analysis is performed. If the air speed in the inclined tube is too high, there is the chance that, due to inertia, the particles sensor becomes unable to pump in the heavier particles. In the same line, the gas sensors may not be able to perform their analysis if the air flows on the sensors' surface too rapidly. Fortunately, the computational fluid dynamics analysis shows that the air's horizontal speed in the inclined tube is roughly 2 m/s, despite the fact that the vehicle is moving at 10 m/s. This means that the device is acting as an airflow dumper.

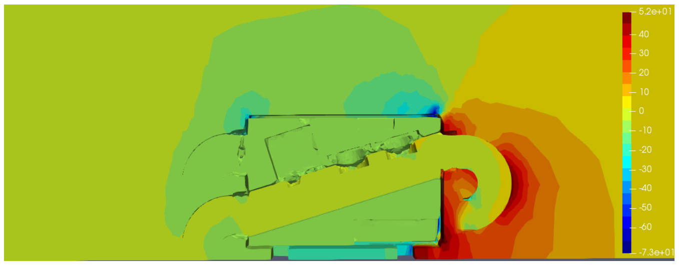
D. Prototype

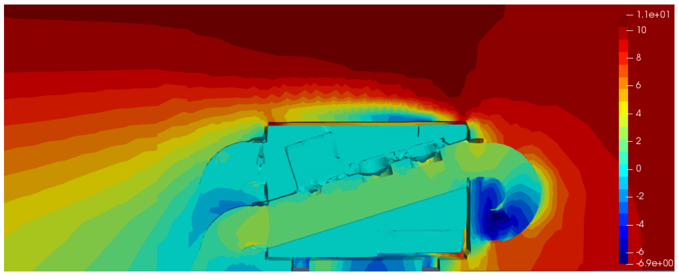
Supported by the computational fluid dynamics analysis, a prototype of the low-cost sensor node was built (see Fig. 4). The node is based on a standard IP65 electronics plastic box. The inclined tube is based on a standard PVC tube, whereas the curved tubes are consumable plastic toilet plumbing pipes. All these components are consumer-based to ensure affordability. All elements junctions are bonded and protected with silicone and aluminum tape for additional mechanical strength and better waterproofing.

III. Hardware Solution

A. Processor

The block diagram for the sensor node's hardware is depicted in Fig. 5. The specific sensors, model and manufacturer, are listed in Table I. At the core of the unit we use the RaspberryPi 3B+ board computer, which is a 1.4 GHz 64-bit quad-core processor, with dual-band wireless LAN and


(a) Pressure (Pa)


(b) Horizontal wind speed (m/s), positive to the left.

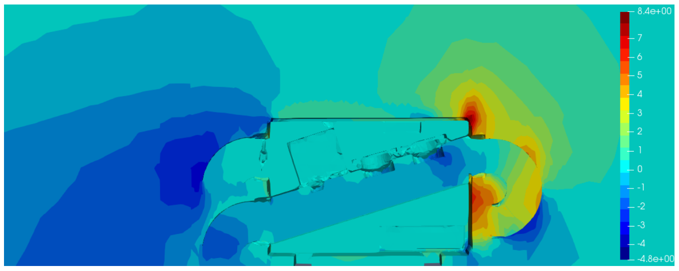

(c) Vertical wind speed (m/s), positive upwards.

Fig. 3. Box's computational fluid dynamics analysis with 10 m/s constant air flow incoming from a vertical wall-like inlet at the right, without air extraction fans (passive mode).

TABLE I
DESCRIPTION OF HARDWARE MODULES

Module	Model	Notes
DC-DC	CUI PDQE15-Q24-S5-D	5V/3A output
T/H/P	Adafruit BME280	5V/I2C (Ox77)
ADC	Adafruit ADS1115	3.3V/I2C (0x48)
CO	Alphasense CO-B4	336mv/ppm
NO_2	Alphasense NO2-B43F	165mV/ppm
Particulate	Alphasense OPC-N3	PM2.5/PM10/SPI
GPS	Adafruit Feather GPS	3.3V/UART
FAN	Sunon EF30080S2	5V/0.4W/3CFM

Bluetooth 4.2/BLE. Although the RaspberryPi model 4 is already available, we found the previous model to be a better solution due to the reduced power dissipation while providing more than enough computing power to process all the sensor data and wireless communications. Moreover, the Raspberry allows the installation of a Linux OS, which is useful to set up database and MQTT [9] servers.

B. Power Supply

In order to take advantage of the vehicle's 12V battery supply, we use a 15W DC-DC converter to obtain a 5V/3A output. In the particular case of a city bus, the input voltage $Vbus$

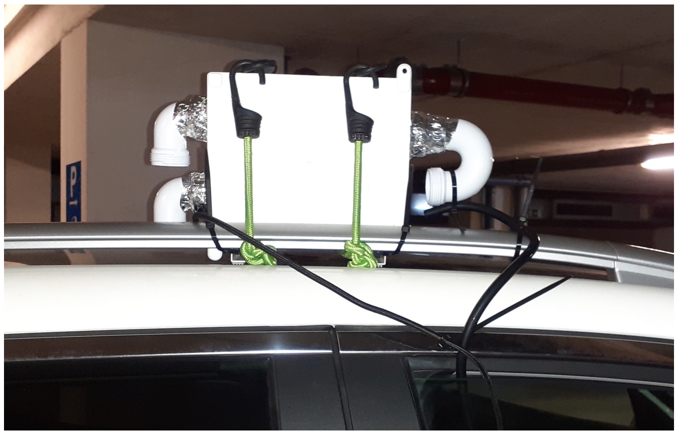


Fig. 4. Sensor node prototype strapped to a vehicle's roof. The cables entering the window are for the box's power supply and additional airflow intake for a DustTrak device (commercial device used for reference measures).

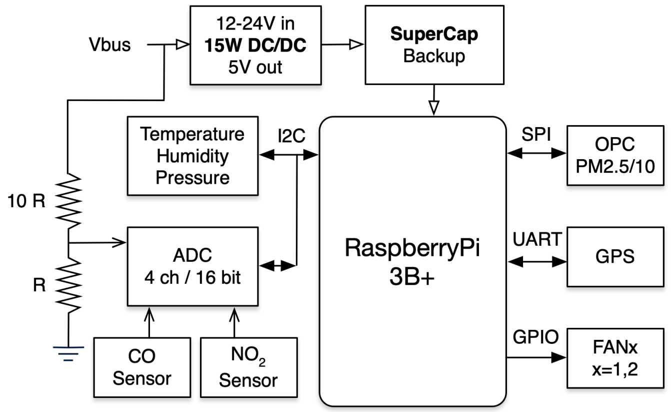


Fig. 5. Block diagram for the sensor node's hardware

can fluctuate above the standard 12V, possibly reaching 24V. As a consequence, we chose a converter with an appropriate wide range input. When the vehicle ignition is switched off, a resistive voltage divider is used by the processor to read this power loss condition and initiate a controlled shutdown. This process might take a few seconds and, therefore, a super-capacitor backup module is provided to maintain the energy required in keeping the Raspberry alive during this time interval.

C. Air Quality Sensors

1) Gas Sensors: In order to monitor the urban air we use two gas sensors from Alphasense that provide readings for carbon monoxide (CO) and nitrogen dioxide (NO_2). Gas concentrations for NO_2 are typically 20 to 200 ppb at the roadside and, so, we chose fuel cell technology sensors with fourth electrode compensation that can reliably detect very low parts per billion levels. Since these sensors provide an analog voltage readout, we use an analog-to-digital converter to provide the corresponding digital values to the processor via an I2C interface. This bus is shared by another sensor, giving information about the temperature, humidity, and atmospheric pressure.

2) Particulate Sensing: To measure mass concentrations of suspended Particulate Matter (PM), we use a laser based optical particle counter OPC-N3, from Alphasense, which is able to measure the particulate fractions of most interest, from the finest of particles up to larger particles such as pollen. The sensor (slave) provides digital outputs of PM1, PM2.5, and PM10 and communicate with processor (master) via an SPI interface.

3) GPS sensor: The geographic position of the vehicle is provided by the GPS unit, whose purpose is to allow us to correlate pollution levels with certain urban areas, like narrow streets, roundabouts or concentration of vehicles around traffic lights. The sensor uses an UART interface.

4) General purpose input/output: Some GPIO pins are used to switch on or off the inlet and outlet fans that impose an air flow even when the vehicle is at rest. Not shown in the block diagram is an RGB led to provide colour feedback about the activities in the processor.

IV. Software Solution

A. Architecture

Fig. 6 depicts the system's major software components and their connectivity. MQTT, a lightweight publish-subscribe messaging protocol adequate for unreliable networks, is used as the main inter-component communication medium. A sensor node i, associated to vehicle i, runs a persistent process, *sensor node i manager*, responsible for sensor data acquisition, pre-processing, local storage, and data transmission via MQTT messages. These messages are published on project-specific topics on a remotely located public MQTT server through recurring to the Paho MQTT Python client library. These communications are currently done over 3G/4G internet links. A project-dedicated server subscribes the topics in order to gather the produced sensor data, process it, and provide pollution maps to Graphical User Interfaces (GUI) that allow ExpoLIS users to access the project's services. Conversely, the project server is also allowed to publish MQTT messages on project-specific topics to configure the activity of each sensor node. By subscribing to these topics, all sensor nodes get the appropriate commands and react accordingly.

In some deployments it may not be possible for sensor nodes to access the internet (e.g., when placed on an underground parking lot). In this case it is not possible to send commands and receive sensor data via the public MQTT server. Therefore, the sensor node runs a private (local) MQTT server on which it mirrors the messages otherwise published on the public MQTT server. To provide external access to the private MQTT server, the sensor node is also configured as a WiFi hotspot (i.e., it provides its own local wireless network). Through this link, users in the immediacies of the sensor node are able to send commands and receive sensor data. A typical use case is the following. A user deploys the sensor node on an internet-absent site, connects to the sensor node's private WiFi hotspot with a GUI running on the user's laptop, via MQTT messages

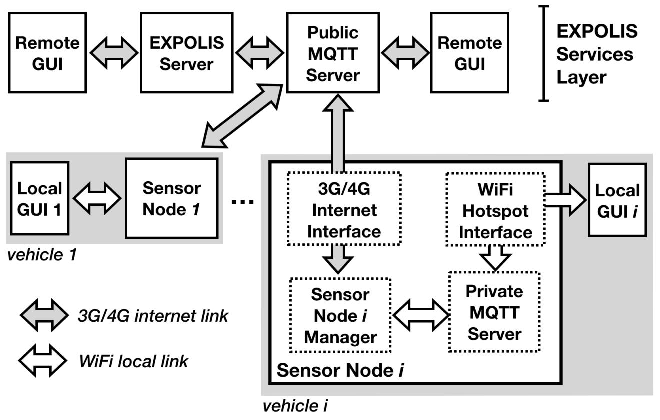


Fig. 6. Software components. Each solid box corresponds to a software component running on a separate machine, whereas the dashed boxes correspond to software sub-components (within the same machine).

the user asks the node to store the acquired data on local text files rather than sending it as MQTT messages. Then, when the acquisition period is expired, the user returns to the site, connects to the sensor node, and publishes a MQTT message on the private MQTT server requesting all logged data. As a response, the sensor node publishes the log files as MQTT messages, which are then acquired by the user. This local interaction is also useful for debugging purposes. All messages are handled with the maximum MQTT quality of service available (two). Although this imposes additional network overhead, it reduces the chances of losing key data (in particular commands) in unreliable networks.

B. Sensor Node Manager

The sensor node manager is implemented as a multi-threaded python persistent process that gathers sensor data every second using a set of open-source and custom device drivers, pre-processes these data, stores the raw and filtered data locally in buffers and files (if requested), publishes the data on the MQTT servers, and reacts to requests published on these.

Fig. 7 depicts the bundle of topics associated to a given sensor node i. By using topic bundles specific to each sensor node, the overhead associated to the publishing and subscribing MQTT messages is reduced. The payload of the messages published every second on the `measurements` topic include a message identifier (an integer number incremented at each new message), timestamp, GPS position, raw sensors data, and filtered sensors data. The sensor node also publishes operational information to the `state` topic. This information includes asynchronous acknowledgement of incoming commands and the node's current state (e.g., whether it is saving to local text files).

The sensor node manager subscribes a `management` topic to be able to receive commands from the ExpoLIS server or a GUI. Via this topic, the sensor node manager can be asked to set Kalman Filters' uncertainty model according to a given set of parameters, to associate a textual description to

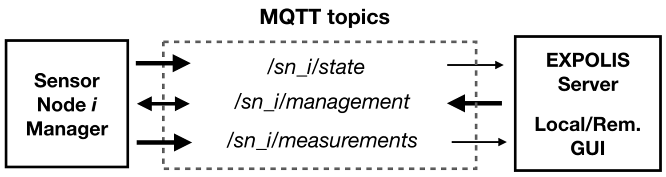


Fig. 7. Bundle of topics associated to a given sensor node i. Thick and thin arrows represent publish and subscribe requests, respectively. Bidirectional arrows represent coexisting publish and subscribe requests.

the next sensor readout (e.g., to associate a description of an observed event), to start, stop, delete, and fetch text-based log files containing all data sent to the `measurements` topic, to restart the sensor node operating system process, to reboot or shutdown the sensor node computational unit via a set of system calls, and to set the operating system clock to a given timestamp so as to cope with the limited accuracy of the real time clock (typically obtained from the system clock of the message sender).

Although the considered gas sensors do not exhibit outliers nor high-frequency noise due to their slow dynamic behaviour, that is not the case for the particles sensors. For this reason, each particle sensor output (i.e., PM1, PM2.5, and PM10) is Kalman filtered independently, that is, assuming no correlation among the outputs.

The Kalman filter is a recursive estimator that, at time-step k, takes r_k as the observation zero mean Gaussian white noise with covariance $R_k : r_k \sim \mathcal{N}(0, R_k)$. The high magnitude and frequency of outliers observed in these sensors, renders the need for an unconventional covariance computation. Therefore, the observation covariance is formulated as the result of accumulating the background covariance of the sensor and the covariance that results from extreme outlier events (endogenous or exogenous to the sensor):

$$R_k = \alpha + \gamma \log \left(\frac{\max(o_k, o_{k-1})}{\min(o_k, o_{k-1}])} \right), \tag{1}$$

where α represents the background covariance of the sensor and γ represents the base covariance of the sensor as a result of the presence of outliers, whose relevance is computed according to a ratio of change from the previous to the current observation. The stronger the change, the higher the covariance. The log function squeezes this contribution to ensure that extremely strong oulier events do not saturate the filter. Although both α and γ can be updated online by sampling the natural statistics of the sensor while in operation, currently, these are empirically defined.

The plot presented in Fig. 8 illustrates the ability of the implemented Kalman filter to smooth the sensor data, reject outliers, and yet track the signal. The peaks present in the plot that were rejected by the filter are extreme and do not last more than one time-step and, thus, are clearly outliers.

C. Error Recovery

Every ten cycles of the sensor node manager's main thread (every 10 s), a message is sent from that thread to the

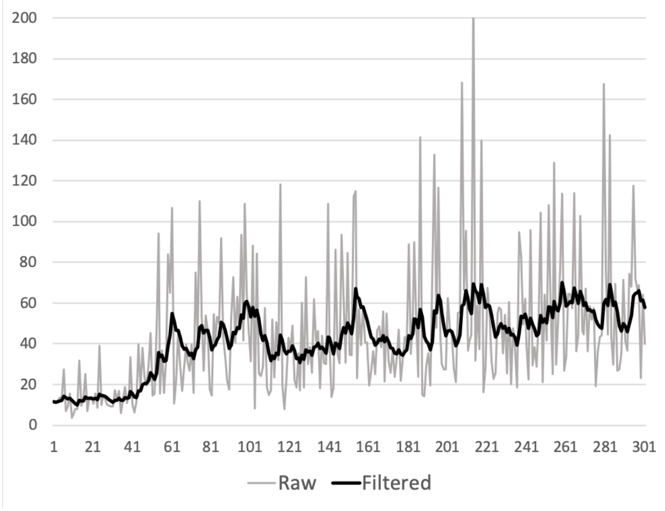


Fig. 8. Kalman filtering PM10 mass concentrations captured by the sensor node mounted on a moving car in an urban environment (see Section V-B), with $\alpha = 20, \gamma = 50$.

`management` topic at the private MQTT server. Then, if the callback function that subscribes this topic receives the sent message, it touches a specific file so as to update its modification time. Hence, if for some unexpected reason (e.g., problem with a sensor or local network interface failure) the sensor node manager's main thread gets blocked, the file is no longer touched. A *cron* service (a script running at the operating system level every minute) watches the modification time of the file to force a kill and restart of the sensor node manager whenever the file becomes older than one minute. If the file is at least five minutes old (five restart attempts have been unsuccessfully attempted), then the machine is rebooted.

Despite the fact that the MQTT quality of service is set to its available maximum (two), several messages are permanently lost in the face of network dropouts that last more than one minute. To detect these situations, the ExpoLIS server runs a periodic script every fifteen minutes that determines the existence of missing messages by finding gaps in the identifiers of the already received ones. Whenever gaps are found, the identifiers of those messages are published as a single message in the `management` topic of the corresponding sensor node. The reception of this message in the sensor node manager triggers the re-sending of the lost messages, stored locally on a circular buffer with one hour capacity, to the `measurements` topic.

D. ExpoLIS Server

Information collected by the sensor network is stored in a Postgis database. Fig. 9 shows a diagram of the tables in the database. These contain information about sensor data, hardware specifications and deployment, routes taken by the buses, existing lanes. To cope with an heterogeneous sensor network, each measured data (humidity, PM1, ...) is stored in a distinct table (represented by table measurement_*D* in the figure). Moreover, there are functions to query individual

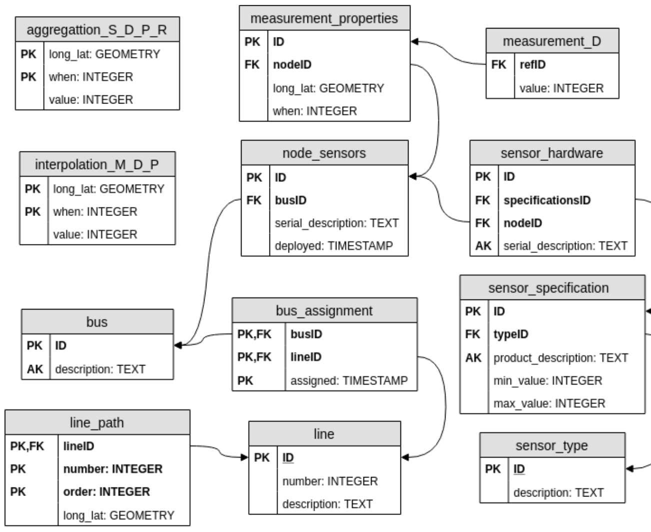


Fig. 9. An entity relation diagram of the tables in the database.

measured data, and a single function with variable number of parameters to insert sensor readings.

As the number of sensor readings can increase rapidly, there are tables that store aggregated information. Aggregation is done by applying a statistical function (maximum, minimum, or average) to data in a period (hour, or day), and in a square (with side 50 or 100 meters).

Since sensor readings are limited to bus lanes, we apply interpolation algorithms to compute pollution levels in non covered areas. This information is stored in a set of tables (interpolation_*M*_*D*_*P* in the figure).

The software to create the database was developed in Python and it includes a script to install the database as a Docker container. Both aggregation and interpolation are performed by *cron* services that run at the rate associated with the respective period.

Besides the database, the ExpoLIS server has a MQTT client that subscribes to sensor data. As already mentioned, it also has a periodic service that keeps track of incoming messages sequential identifiers. If it detects gaps in these IDs, it requests the corresponding messages by sending a MQTT management message.

E. Visualisation

In order to bring awareness to the general public regarding pollution, we have developed an Android application to display sensor data. We provide three types of graphs: 1) map with aggregated data during a time interval; 2) plot with pollution in a circular area during a time interval; 3) plot with pollution along a bus line filtered by a period. All these graphs use the data stored in the aggregated tables of the database. Fig. 10 shows an example of a map graph.

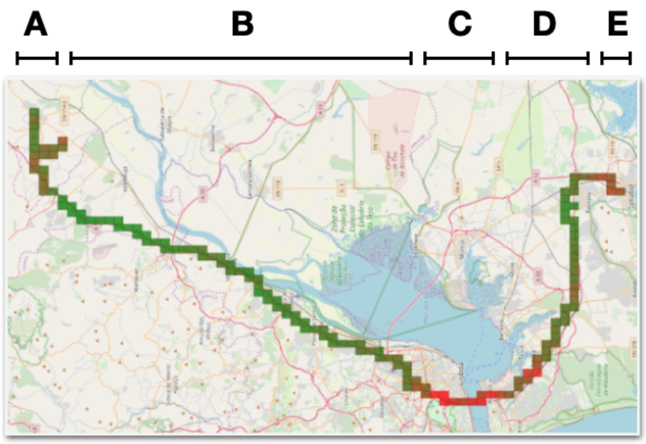


Fig. 10. Measured distribution of PM10 mass concentrations across a 100 km trip from Cartaxo to Setúbal, Portugal, January 31, 2020. Each label corresponds to a key segment of the route. A: Cartaxo, rural city with light traffic. B: main highway with medium traffic. C: Lisboa, main urban city with heavy traffic jams. D: secondary high way with light traffic. E: Setúbal, secondary urban city with medium traffic.

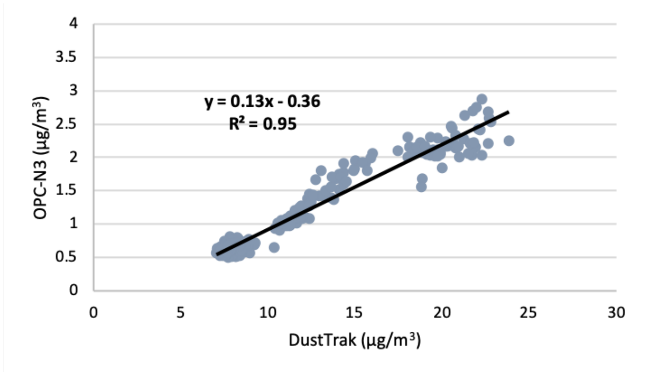
V. Results

A. Static Use Case

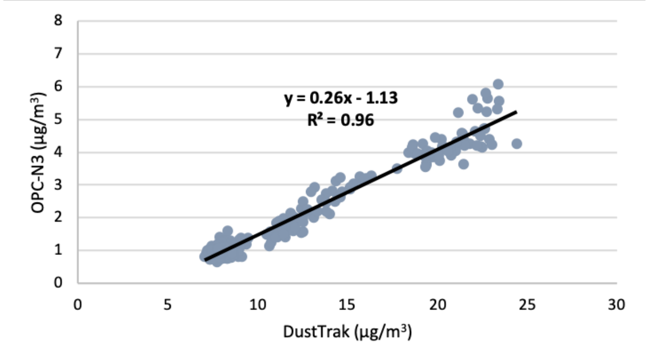
Aiming at establishing the utility of the data provided by the particulate matter low cost sensor OPC-N3 from Alphasense, the unit was set in parallel with a light scattering laser photometer DustTrak 8533 from TSI Inc.. This equipment measures mass concentrations of several size fractions of PM, namely PM1, PM2.5, and PM10. Fig. 11 presents the scatter-plots of the sensor mass concentrations versus the DustTrak measurements for PM1 (a), PM2.5 (b), and PM10 (c), based on 1 minute averaged values. Strong linear correlations were observed between the sensor and DustTrak mass concentrations. The highest coefficients of determination (R^2) were obtained for PM1 (0.95) and PM2.5 (0.96), being slightly lower for PM10 (0.85). These values suggest that the sensors have high precision. However, the same should not be concluded about accuracy, since the DustTrak concentrations are, on average, 11, 7 and 6 times higher than the sensor concentrations for PM1, PM2.5, and PM10, respectively. Although the difference between concentrations is high, the correlation that exists between them, allows the application of a correction factor to the original sensor concentrations to obtain the real PM mass concentrations.

B. Dynamic Use Case

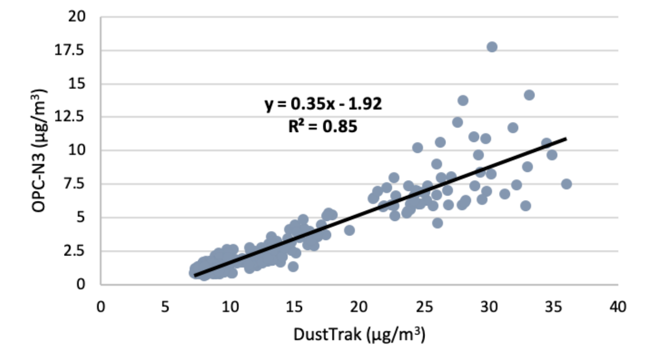
To assess the ability of the sampling solution to perform properly when mounted on a mobile vehicle a second set of tests was carried out. Concretely, the sensor box was mounted on a vehicle's roof, as depicted in Fig. 4), and a 100 km trip in the Lisbon region was performed on the 31st of January, 2020. The trip started in a rural city with light traffic, Cartaxo, included heavy traffic jams in Portugal's capital, Lisbon, and ended in a secondary urban city with medium traffic, Setúbal.



(a) PM1



(b) PM2.5



(c) PM10

Fig. 11. Scatter plot of the linear regression of the sensor OPC-N3 against the DustTrak 8533 for the mass concentrations of PM1, PM2.5, and PM10.

Fig. 10 depicts the measured distribution of PM10 mass concentrations in the mobile app, across the trip. As expected, higher particles concentrations were registered in the portions of the trip where higher levels of traffic were observed. Although the qualitative results are promising, it is necessary to perform additional tests in order to compare the obtained results against ground truth. Ground truth can be obtained from more accurate, yet more expensive and bulkier, sensors and from fixed reference stations distributed across the city.

VI. Conclusion

The first prototype of the ExpoLIS system was presented. This system aims at providing an affordable solution for air quality dense sampling of human-populated environments by means of exploiting low-cost sensors and the mobility of commuting vehicles, both public and private. The accumulated data is expected to inform citizens' decision making towards a healthier way of life. The attained design is based on affordable components, fostering massive reproduction of the system components. Preliminary static laboratory experiments and dynamic field experiments show the ability of the sensor node to track changes in the air quality while coping with challenging weather conditions. As future work, we plan to revise the mechanical design so as to ease the attachment of the sensor node to the vehicle and better handle potentially damaging vibrations. The graphical user interfaces will be improved in order to provide a more intuitive and in depth access to the generated data. We are also considering the implementation of web services in order to allow external systems (e.g., vertical applications) to efficiently interact with the data server. Finally, to thoroughly assess the robustness and accuracy of the system in comparison to high-cost (commercial) mobile alternatives and existing reference monitoring stations, various large-scale field experiments will be run with the system already deployed on a fleet of city buses.

Acknowledgment

We would like to thank our colleagues Tomás Brandão, José Moura and Joaquim Reis for their helpful comments throughout the preparation and execution of the project. This work is financed by national funds through FCT - Foundation for Science and Technology, I.P., within the framework of the project ExpoLIS (LISBOA-01-0145-FEDER-032088).

References

[1] "Eea, air quality in europe 2019 report," Luxembourg: Publications Office of the European Union, Tech. Rep., 2019.

[2] S. Devarakonda, P. Sevusu, H. Liu, R. Liu, L. Iftode, and B. Nath, "Real-time air quality monitoring through mobile sensing in metropolitan areas," in *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing*, 2013, pp. 1–8.

[3] W. Y. Yi, K. M. Lo, T. Mak, K. S. Leung, Y. Leung, and M. L. Meng, "A survey of wireless sensor network based air pollution monitoring systems," *Sensors*, vol. 15, no. 12, pp. 31 392–31 427, 2015.

[4] J. E. Thompson, "Crowd-sourced air quality studies: A review of the literature & portable sensors," *Trends in Environmental Analytical Chemistry*, vol. 11, pp. 23–34, 2016.

[5] S. Brienza, A. Galli, G. Anastasi, and P. Bruschi, "A low-cost sensing system for cooperative air quality monitoring in urban areas," *Sensors*, vol. 15, no. 6, pp. 12 242–12 259, 2015.

[6] E. Pinto, F. Marques, R. Mendonça, A. Lourenço, P. Santana, and J. Barata, "An autonomous surface-aerial marsupial robotic team for riverine environmental monitoring: Benefiting from coordinated aerial, underwater, and surface level perception," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE, 2014, pp. 443–450.

[7] P. Deusdado, M. Guedes, A. Silva, F. Marques, E. Pinto, P. Rodrigues, A. Lourenço, R. Mendonça, P. Santana, J. Corisco, S. M. Almeida, L. Portugal, R. Caldeira, J. Barata, and L. Flores, "Sediment sampling in estuarine mudflats with an aerial-ground robotic team," *Sensors*, vol. 16, no. 9, p. 1461, 2016.

[8] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in physics*, vol. 12, no. 6, pp. 620–631, 1998.

[9] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, *MQTT Version 5.0. OASIS Standard: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.*, March 07, 2019.