# Technical manual

## For the use of low cost air quality sensors in monitoring activities

VAQUUMS

# Index

# Introduction

In recent years an increasing number of air quality measurement methods became available, but the reliability of their results is often unknown. Within the LIFE VAQUUMS-project (Various Assessments of air QUality Measurement methods and their policy Support) we investigate which sensors can be a valuable addition to the reference measurements. The project aims to assess the reliability of air quality sensors (particle matter, nitrogen dioxide and ozone). After a literature review and an expert consultation, the most promising sensors (see sensor selection procedure, appendix 1) were tested both in the lab and in the field. Based on the results of the LIFE VAQUUMS-project, we will develop guidelines and protocols for the correct use of air quality sensors. In this way we facilitate improvements in citizen science, enabling everybody to measure the air quality in their neighborhood.

In this technical manual we describe how we build working low-cost air quality sensors, how we attained data from these sensors and how we managed to keep multiple sensors operational for over one year. Naturally there are multiple ways to do so, but the LIFE VAQUUMS-project aimed to mimic citizens' approaches when conducting air quality measurements. Therefore we assembled and calibrated the sensors in a simple way. We did not aim to build a professional housing for the sensor, neither did we pursue to develop calibration algorithms to improve the sensor quality. As a result this document provides hands-on recommendations for operating low cost air quality sensors.
LIFE VAQUUMS is a project commissioned by the European Commission under the LIFE Preparatory Project program. Two Belgian and two Dutch partners cooperate in this project.

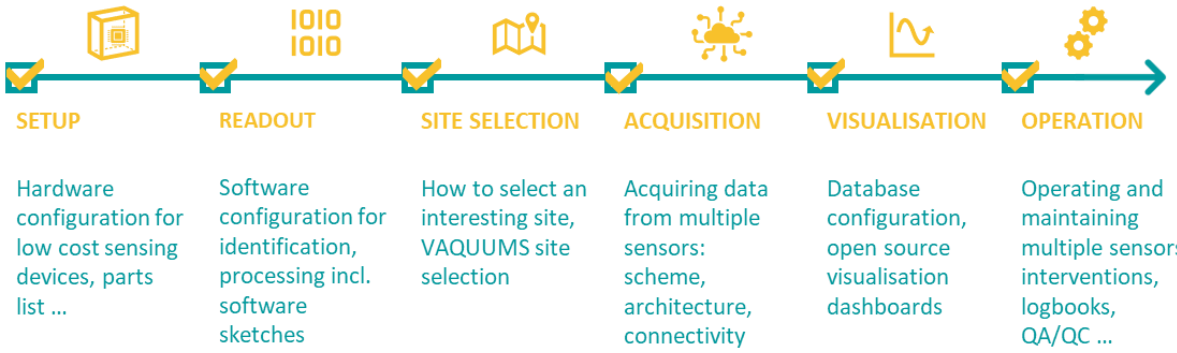



*Figure 1: Outline of the technical manual*

## Definitions

- Low-cost sensor: an affordable device to detect events or changes in the environment.
- Microcomputer: a small and inexpensive computer with all units mounted on a single printed circuit.
- PM or particulate matter: the mixture of tiny solid particles and liquid droplets that float in the air. The fine dust originates from primary of secondary sources. Since the particles are very small (smaller than 10 µm) they can not be seen with the naked eye. The smaller the particles, the deeper they can penetrate the lungs and thus cause health effects.
- $NO_2$ or nitrogen dioxide: the harmful gas that originates from the reaction of oxygen or ozone with nitrogen monoxide (NO). NO arises when burning something at high temperatures.
- $O_3$ or ozone: gas that originates from the reaction of gasses like NO and volatile organic compounds under the influence of sunlight. It has strong oxidizing characteristics which are harmful for people and animals.
- Sketch: written source code of the software of the sensor.
- Reference measurements: the measurement conducted by the official monitors from recognized networks to measure air quality according to the European Directives.
- Data validation: assessment of the sensor signal and variation in time and a comparison with nearby or co-located official instruments or other sensors. During this process invalid and suspicious data are filtered.
- Sensor vs. Module: the sensor is the raw sensor unit as it was bought vs. the module is the raw sensor + microcomputer/WiFi-unit/power supply/housing/…

## Sensor configuration

Our point of view on the technical configuration and the installation of the tested sensors was mimicking citizen (science) setups. This approach allows building plans, code etc. to create added value for citizen scientists through direct usability and increased uptake. In addition only simple calibration factors were used when needed, so our evaluation of sensor quality would be indicative of citizen-driven implementations. We did not aim to build a professional housing for the sensor, neither did we pursue to develop enhanced calibration algorithms to improve the sensor quality.

The following sections will summarize our approach, detailed configuration and programming information can be found in Table 1.

### Hardware

All 18 tested sensors were purchased as 'raw' sensors with the exception of Envea and Dylos commercial products. A general building plan was used to configure the raw sensors. A measuring sensor unit typically contains (Figure 2):

- Raw sensor: As purchased, sometimes including a sensor board
- Arduino UNO: Microcomputer used for processing the sensor unit signal
- ESP8266: Microcomputer used for the WiFi-communication with the database
- 9 V power module: Power supply for the Arduino UNO
- Power stabilizer: Ensures a stable power supply to the different modules
- Breadboard: Enables easy assembly of the above components
- Optional:
    - 24 V power module: Power supply for the sensor (some types of gas sensors only)
    - Analog-digital converter (ADC): Converts the signal of the sensor from an analog voltage (0-5V) to a digital value in 16 bits (gas sensors only)

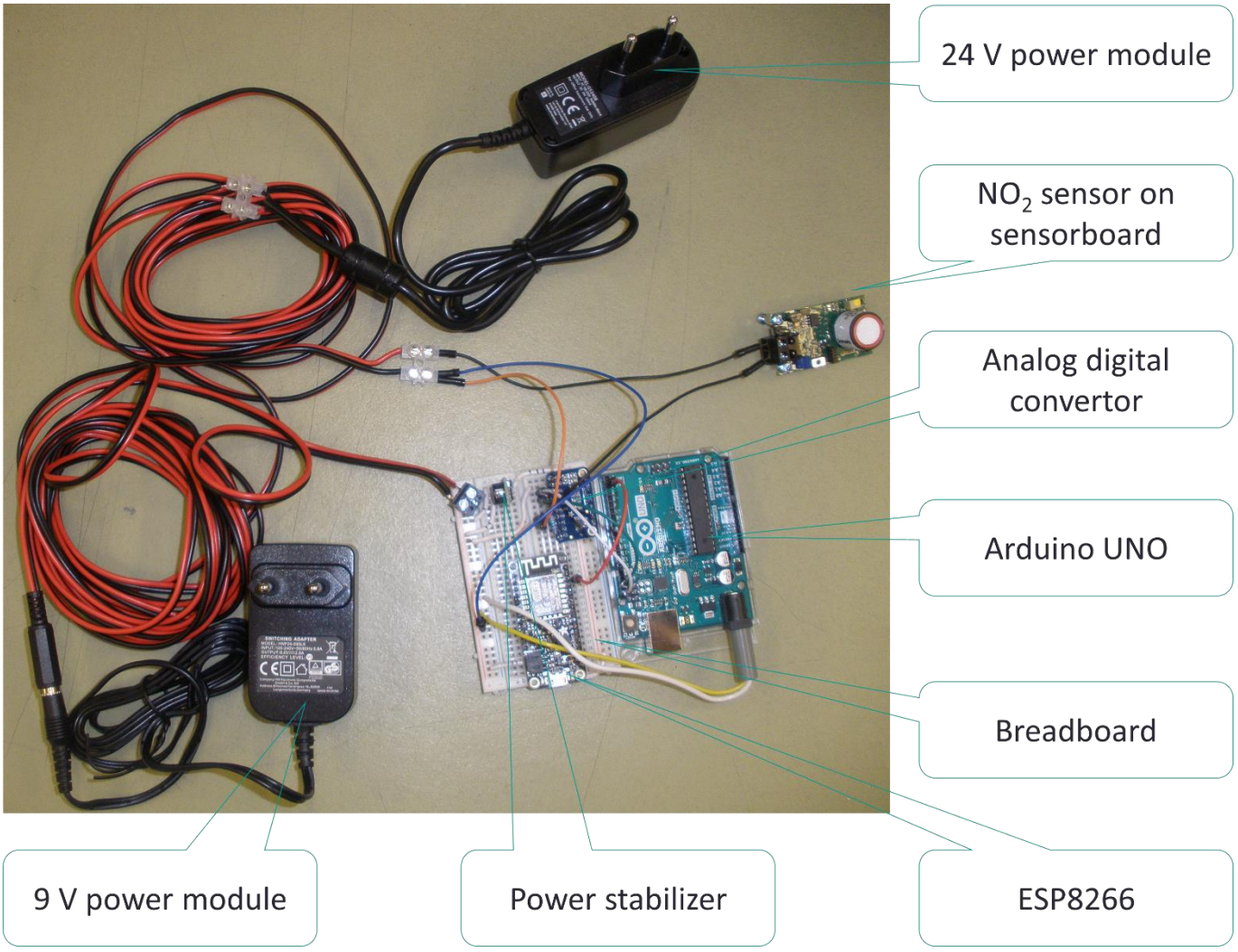



*Figure 2: modules used to build a working low cost air quality sensor*

When all modules were assembled, we placed them in an 8 cm x 8 cm x 10 cm PVC housing (Figure 2, Figure 3). These housings were placed into the exposure boxes in the laboratory tests and into three shelters in the field tests (Figure 4). This setup should translate rather well to Do-It-Yourself housing typically seen in citizen science projects, i.e. slightly larger tubing, facing all components inward and using two 90° bends to avoid precipitation entering.

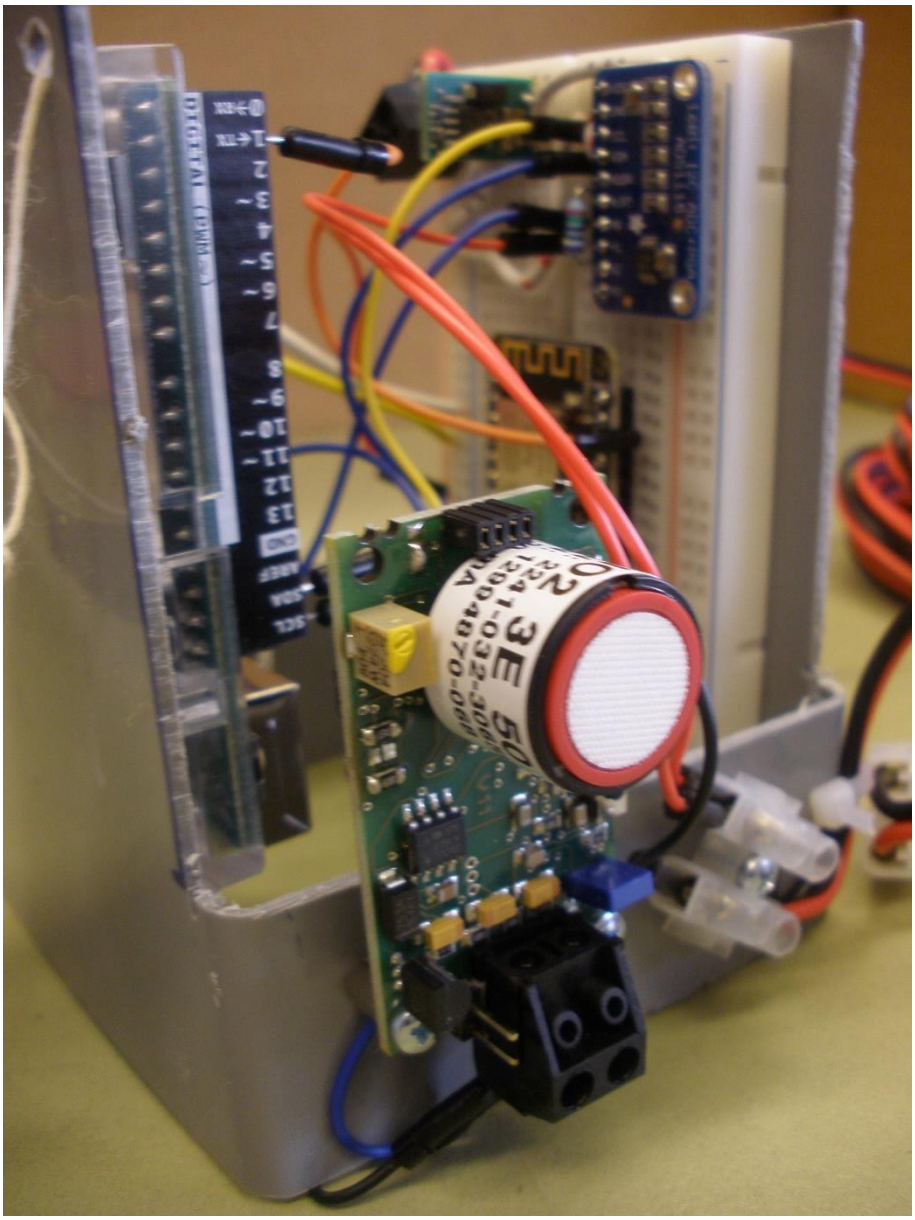


*Figure 3: Picture of working low cost air quality sensor assembled within the PVC housing*

*Figure 4: Sensors within the exposure boxes in the laboratory (left) and within the shelter at the field site (right).*

## Software

The sketches or Arduino programs were written in the standard Arduino software using a windows 10 portable. Almost all sketches have the same structure. The links to the sketches per sensor can be found in Table 1.

### *Sensor identification*

In the setup sketch one can find the sensor number (based on the code assembled using a combination of pins 2 to 4, in this way the number is embedded in the hardware and not in the software of the sensor) and the variables which are measured.

### *Sensor readout*

The serial connection with the ESP module and with the sensor or ADC converter is also initialized in the setup. In the loop every second the connected sensor is questioned and will send a serial value (with a checksum) to the ESP.

### *Signal processing*

If the sensor provides multiple measurements per second, the sketch will average these values per second. If the sensor provides measurements less frequently, every new measurement will be communicated to the ESP. In the latter case, the 1-second time frames in between the measurements will be recorded as empty (NAN) in the database. We opted to gather data every second since this would enable us to study the desired level of data aggregation (1 second, 1 minute, 5 minutes, 30 minutes, 1 hour, 24 hours) later on.

## Sensor assembly & programming documentation

More information about the assembly and programming of every sensor type can be found in the following documents (Table 1, Appendix 2).

*Table 1: Links to appendices containing the information on assemblage and programming of each sensor type tested within the VAQUUMS-project.*

| Gas sensors | | PM sensors |
|---|---|---|
| $NO_2$ | $O_3$ | PM |
| Alphasense NO2-B43F | Alphasense OX-B431 | Alphasense OPC-N2 |
| Citytech NO2 3E50 | Citytech O3 3E1F | Dylos DC1700 |
| Envea Cairclip NO2 | Envea Cairclip O3/NO2 | Honeywell HPMA115S0 |
| Membrapor NO2/C-1 | Aeroqual SM50 | Nova fitness SDS011 |
| Membrapor NO2/C-20 | Membrapor O3/C-5 | Plantower PMS 7003 |
| | | Shinyei PPD42NJ |
| | | Shinyei PPD60PV-T2 |
| | | Winsen ZH03B |

## How to select interesting sites

The site selection mainly depends on the research question, which pollutants you want to measure and why you want to measure these pollutants. It is important to consider whether you want to measure the air quality in general or if you want to measure a specific source of pollution.

Prior to selecting a site, it is best to check the information already available for this location. For example, measurements with low cost sensors performed by citizen scientists, nearby reference monitoring and the modeled air quality map based on the reference measurements.

It is also good to know which factors influence the air quality such that you can take this into account when you select a site. Think for example about the environment, the different types of pollution sources, the weather conditions and the time.

More elaborate information on drafting your own research question and using it to select an interesting measurement site can be found here (in Dutch).

## Site selection for VAQUUMS experiment

In the LIFE VAQUUMS-project we focus on particles (PM), nitrogen dioxide ($NO_2$) and ozone ($O_3$). The project aims to test the low cost air quality sensors next to a reference air quality measurement station. In Flanders the Flanders Environment Agency has 77 reference stations with semi-automatic monitors (end 2019):

- 58 measuring $NO_x$
- 19 measuring$O_3$
- 44 measuring $PM_{2.5}$

In order to keep the project manageable, we selected one site for all the sensor types. In 13 reference stations all 3 pollutants are measured. We selected the urban background station in Borgerhout, Antwerpen (Belgium) because it is a location nearby our headquarters with plenty of room to position the required setup. Interesting about this test site is that depending on the direction of the wind this site can be classified as either an urban traffic or an urban background location. This dual character also guarantees more variation in pollutant concentrations and characteristics. Meteorological information for this site indicates it should represent sufficient variation across all parameters.

## Data acquisition system

### General approach

Each sensor is able to connect to a WiFi network thanks to the onboard WiFi-module of the ESP8266 NodeMcu. A custom firmware for the ESP8266 was written so each sensor automatically connects to the pre-defined VAQUUMS WiFi access point, which was located inside the measurement station (R801). Once a successful WiFi connection was established each sensor started to send its own measurement data over the WiFi network to two laptops, which were also on the same network. Both laptops run on Linux (Ubuntu 18.04 TLS) and serve as the data acquisition system using an Influx database to store the individual time series (measurements) of each sensor. Influx was chosen because it is specifically designed for high throughput timeseries, like ours which report at 1s intervals.

The set-up of both laptops is completely identical, to assure fast write speeds they both have solid state drives. The whole WiFi network is completely detached from the internet thus being a LAN. To allow outside access for monitoring, but restricted to VMM's corporate network only, the master laptop is can route network traffic through a machine-to-machine (M2M) modem allowing for the exchange of data in both directions over a secure tunnel.

This set-up is schematically show in Figure 5.

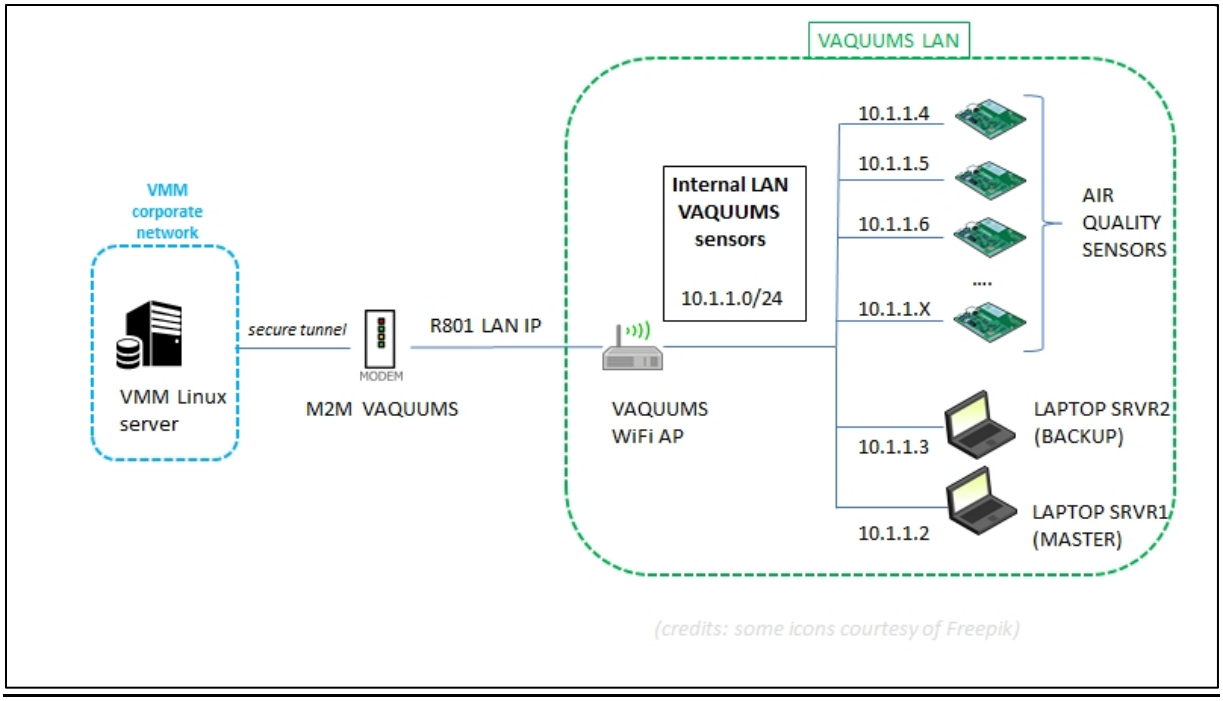



*Figure 5: Schematic outline of the data acquisition from the sensors in VAQUUMS.*

## Database and visualization

Aggregations of the individual time series (1s intervals) to 1 minute mean values for each sensor were configured in the Influx database using continuous queries. The choice to aggregate to 1 minute mean values was based on a trade-off to still allow a high enough temporal resolution while significantly reducing the amount of data (i.e. 60-fold). Specific Linux bash scripts were written to automatically (cron) and every hour transfer these 1 minute mean values for all sensors over the M2M secure tunnel to another Linux server located inside VMM's corporate network. This VMM server was also running an Influx database. Additionally Grafana was used on top of Influx to visualize these minute means for each sensor stored inside the database. Different dashboards were configured in Grafana to allow for quickly screening the near-realtime sensor measurements for a given pollutant and sensor family as shown in Figure 6. This facilitated initial validation and sensor monitoring, allowing for better planning of necessary interventions (fixes) of the sensor hardware, see Figure 7.

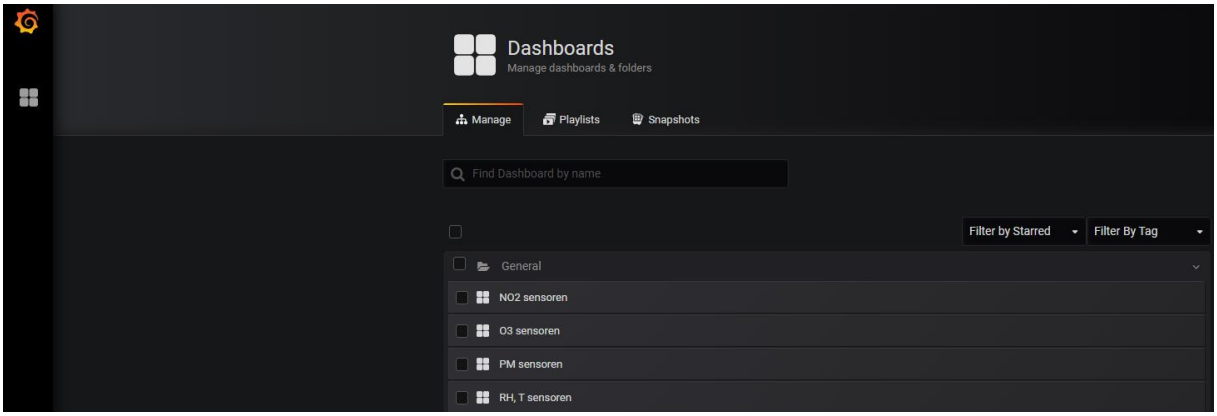



*Figure 6: Welcome screen in Grafana showing the available dashboards for each pollutant.*


*Figure 7: Detailed view of the $NO_2$ dashboard in Grafana showing the results for each sensor family and type for the whole time range of the measurement campaign.*

## Operating multiple low-cost sensors

The QA/QC process of the VAQUUMS field tests consists of two parts: (1) the incoming data were checked regularly in order to detect and solve sensor problems (operational perspective) and (2) before the analyses were done we validated the sensor data (scientific perspective). Both processes are described in the following part of this technical manual. We recommend to perform some checks prior to the actual start of the measurements. This can be done by **co-locating similar sensors for a certain amount of time** to make sure that all sensors are operating as expected and that no faulty units are present. If possible it's even better to co-locate sensors for a certain time with official monitors to get a feeling for the agreement and/or to calibrate the sensor signal.

### Quality assurance

#### Maintenance

The sensors were checked at least on a weekly base, preferable two or three times a week (ideally Monday, Wednesday and Friday), and at least covering the period to the previous check. Through the web interface (Figure 6, Figure 7), the presence/absence of data is checked as a first approach. Also, the general flow of the data is checked. A comparison is made between sensors of the same type, among sensors of the same parameter (PM, $NO_2$, $O_3$) and with the reference measurements of station R801. Deviations involving slope/offset, which will probably be corrected in later phases, are not considered as problematic. From this maintenance evaluation, problems are logged and translated into actions for the field technician.

### Types of interventions

Typically, the technician would go on site once a week to cope with the diagnosed failures and deviations, on occasions this was more frequent. On site, power supply and the general working of sensor modules could be evaluated by checking LED-signals and using a multimeter. When on-site intervention proves insufficient, complete modules are dismounted and brought to the lab for further inspection. There, signals (voltage/communication) can be evaluated in detail, and parts can be replaced.

### Diagnostics

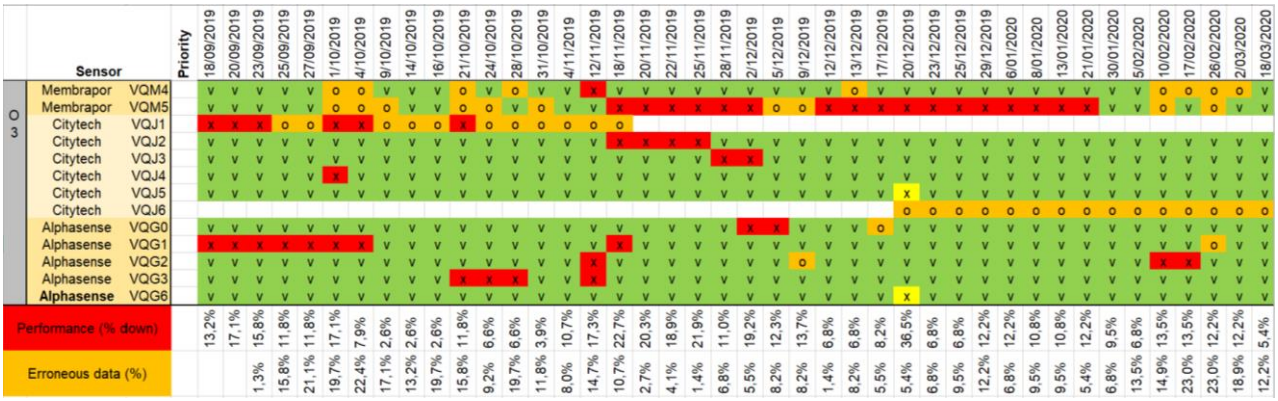
Only a limited number of failures were encountered regularly

- Sensor failure
    - PM-sensors: pollution of the measurement chamber (insects, coarse dirt) was found to be the reason for aberrant signals on multiple occasions.
    - Solid state sensors: poor operation with no further diagnostics.
- Module failure
    - Rebooting the module on site often proves successful, without further diagnostics.
    - Bad electrical contacts in the breadboard mounts, often at the 5V voltage regulator and contacts with the Arduino/ESP/sensor modules; due to mechanical disturbance (vibrations, wind) or due to oxidation of contacts.
    - ESP module failure; a protocol analyzer allows to check communication.
- Electrical breakdown
    - Complete breakdown: regional breakdown (can be tracked), human error or fuse blown; in the latter case, the complete setup is inspected for short circuits and/or water seepage.
    - Partial breakdown: for one (of three) shelters specifically; these are fused separately in power distribution boxes; the setup is inspected for short circuits and/or water seepage.

### Logbook and forms

In order to track the long-term performance of the sensors, the status of each sensor was logged and actions on the field documented.

First, a spreadsheet (Figure 8) is kept in which each row represents an individual sensor and each column represent an instant of check-up. Color coding is used for a raw performance status evaluation: red = signal absent, orange = signal available but doubtful, green = signal available and acceptable. This way, a clear overview is presented in the general performance, and simple statistics can be calculated to evaluate performance over time (e.g., % of sensors down, % of sensors with aberrant data).



| | Sensor | | Priority | 18/09/2019 | 20/09/2019 | 23/09/2019 | 25/09/2019 | 27/09/2019 | 1/10/2019 | 4/10/2019 | 9/10/2019 | 14/10/2019 | 16/10/2019 | 21/10/2019 | 24/10/2019 | 28/10/2019 | 31/10/2019 | 4/11/2019 | 12/11/2019 | 18/11/2019 | 20/11/2019 | 22/11/2019 | 25/11/2019 | 28/11/2019 | 2/12/2019 | 5/12/2019 | 9/12/2019 | 12/12/2019 | 13/12/2019 | 17/12/2019 | 20/12/2019 | 23/12/2019 | 25/12/2019 | 29/12/2019 | 6/01/2020 | 8/01/2020 | 13/01/2020 | 21/01/2020 | 30/01/2020 | 5/02/2020 | 10/02/2020 | 17/02/2020 | 26/02/2020 | 2/03/2020 | 18/03/2020 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O3 | Membrapor | VQM4 | | v | v | v | v | v | o | o | v | v | v | o | v | o | o | v | v | x | v | v | v | v | v | x | x | v | v | o | v | v | v | v | v | v | v | v | o | o | o | o | v |
| | Membrapor | VQM5 | | v | v | v | v | v | o | o | o | v | v | o | o | v | o | v | v | x | x | x | x | x | x | o | o | x | x | x | x | x | x | x | x | x | x | v | v | o | v | o | v | v |
| | Citytech | VQJ1 | | x | x | x | o | o | x | x | o | o | o | x | o | o | o | o | o | o | | | | | | | | | | | | | | | | | | | | | | | | |
| | Citytech | VQJ2 | | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | x | x | x | x | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v |
| | Citytech | VQJ3 | | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | x | x | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v |
| | Citytech | VQJ4 | | v | v | v | v | v | x | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v |
| | Citytech | VQJ5 | | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | x | v | v | v | v | v | v | v | v | v | v | v | v | v |
| | Citytech | VQJ6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | o | o | o | o | o | o | o | o | o | o | o | o | o |
| | Alphasense | VQG0 | | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | x | x | v | v | v | o | v | v | v | v | v | v | v | v | v | v | v | v | v | v |
| | Alphasense | VQG1 | | x | x | x | x | x | x | x | v | v | v | v | v | v | v | v | v | x | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | o | v | v | v |
| | Alphasense | VQG2 | | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | x | v | v | v | v | v | v | v | o | v | v | v | v | v | v | v | v | v | v | v | x | x | v | v | v |
| | Alphasense | VQG3 | | v | v | v | v | v | v | v | v | v | v | x | x | x | v | v | x | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v |
| | Alphasense | VQG6 | | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | x | v | v | v | v | v | v | v | v | v | v | v | v | v |
| Performance (% down) | | | | 13.2% | 17.1% | 15.8% | 11.8% | 11.8% | 17.1% | 7.9% | 2.6% | 2.6% | 2.6% | 11.8% | 6.6% | 6.6% | 3.9% | 10.7% | 17.3% | 22.7% | 20.3% | 18.9% | 21.9% | 11.0% | 19.2% | 12.3% | 13.7% | 6.8% | 6.8% | 8.2% | 36.5% | 6.8% | 6.8% | 12.2% | 12.2% | 10.8% | 10.8% | 12.2% | 9.5% | 6.8% | 13.5% | 13.5% | 12.2% | 5.4% |
| Erroneous data (%) | | | | | | 1.3% | 15.8% | 21.1% | 19.7% | 22.4% | 17.1% | 13.2% | 19.7% | 15.8% | 9.2% | 19.7% | 11.8% | 8.0% | 14.7% | 10.7% | 2.7% | 4.1% | 1.4% | 6.8% | 5.5% | 8.2% | 8.2% | 1.4% | 8.2% | 5.5% | 5.4% | 6.8% | 9.5% | 12.2% | 6.8% | 9.5% | 9.5% | 5.4% | 6.8% | 13.5% | 14.9% | 23.0% | 23.0% | 18.9% | 12.2% |

*Figure 8: Extract from the spreadsheet with the performance status of individual sensors and simple statistics on general performance of the sensors.*

Second, a log is kept describing changes in performance status documented in the spreadsheet, including precise timing of changes, suspicious behavior. Suggestions for action are made towards the technician when sensors black out or show aberrant signals. When sensor signals are restored, it is documented whether this is spontaneous or due to interventions.

Third, the technician who carried out maintenance on the sensors in the field and in the lab kept a detailed log of the actions done on specific sensors.

## Quality control

### Data validation tool

A new tool was developed in R Shiny to allow thorough data validation of the 1 minute mean values for each individual sensor of a given sensor family and pollutant combination, see Figure 9. The user interface allowed to mark a given timeseries window and attach a validation flag to the measurement series as shown in Figure 10 (see further also in *Data Validation*).

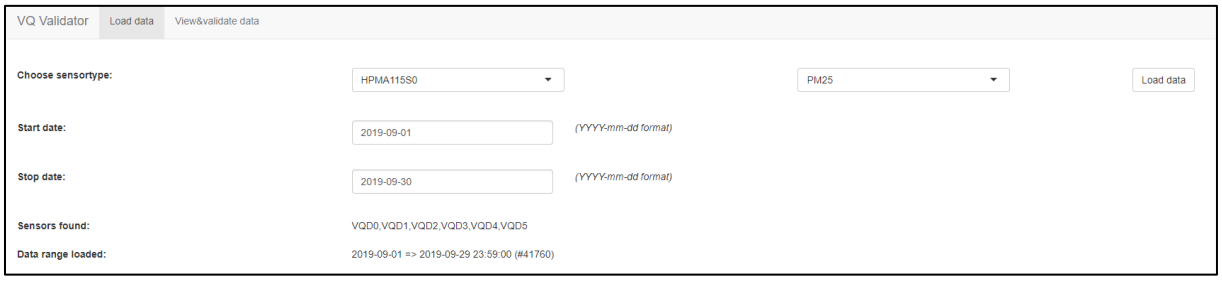



*Figure 9: Initial landing page for the validation tool allow the user to select the sensor type, pollutant and time range.*

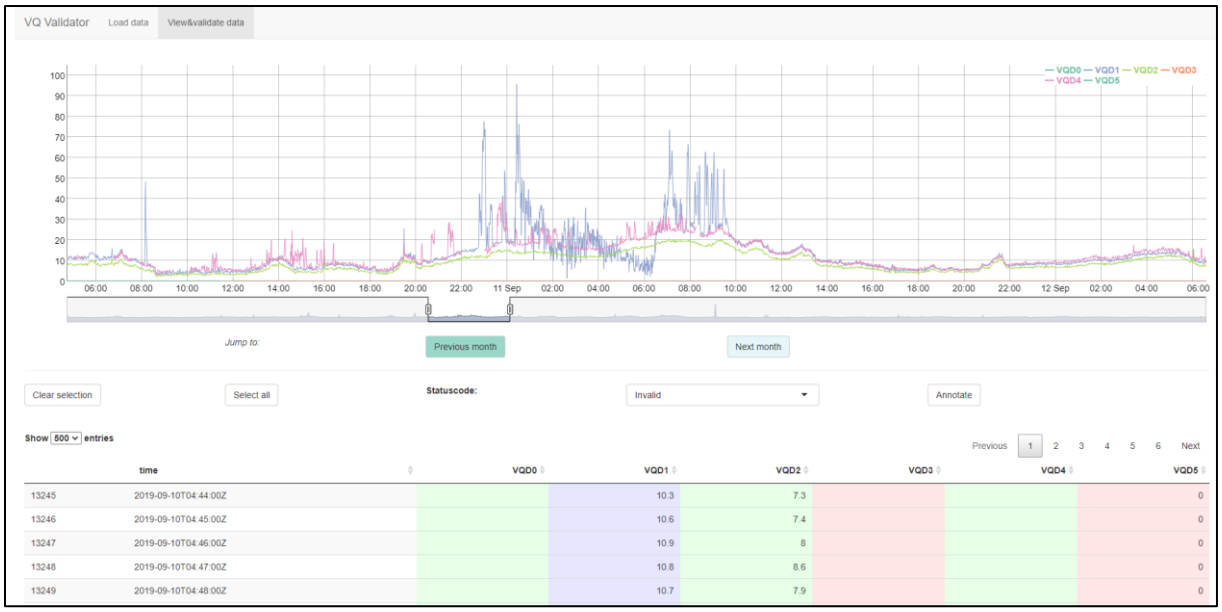



*Figure 10: Illustration of the validation window, here for $PM_{2,5}$ of the HPMA sensor for the whole month of September 2019. A zoom of the graph is shown for the results between the 10th and 12th of September. In the table below the graph one can flag each sensor's 1-minute measurements as suspicious (purple) or invalid (red), by default all measurements are valid (green).*

## Data validation

### *Introduction*

The aim of the validation process is to identify **invalid** and **suspicious** data. With invalid data we mean data that has no relation with the actual concentrations e.g. because of mechanical or electrical problems with the device. Suspicious data is harder to define and often requires expertise and further investigation to identify it.

### *General approach*

The general approach always consists of an assessment of the sensor signal and variation in time and a comparison with nearby or co-located official instruments or other sensors. When doing this one should be aware of the time resolution of the different datasets. Official monitors often show the average of the previous hour. We suggest the user tries to identify both invalid and suspicious data.

**Invalid data:** Typical indications of invalid data would be unrealistically high or low concentrations or atypical behavior such as very rapid change in concentrations that create block patterns or spikes.

**Suspicious data:** Suspicious data are data that is usually identified by signal behavior that deviates from other co-located or nearby instruments or that has an abnormal pattern. Identification of this type of data does require experience in air quality measurements.

Whether suspicious data should be discarded or not is a difficult question. If a sensor shows

a lot of suspicious data it's always best to try to identify the reason. One could for example add a similar sensor at the same location to verify suspicious signals. Or if that is not possible one could swap sensors and see whether the suspicious data remain at the same location or whether they stay with the sensor. In the former case the suspicious data is most likely real and should be considered for analysis, in the latter it is most likely a sensor issue and should be discarded.

*Sensor specific information*

**PM-sensors**: since background concentrations and weather have a huge impact on PM-concentrations we often expect very similar signals and trends when comparing nearby PM-measurements (either official reference measurements or other low-cost sensors). This feature can be used to identify invalid and suspicious data. A typical validation process would consist of comparing the PM-sensor data with nearby official monitors and or comparing a sensor with nearby or co-located sensors of the same type.

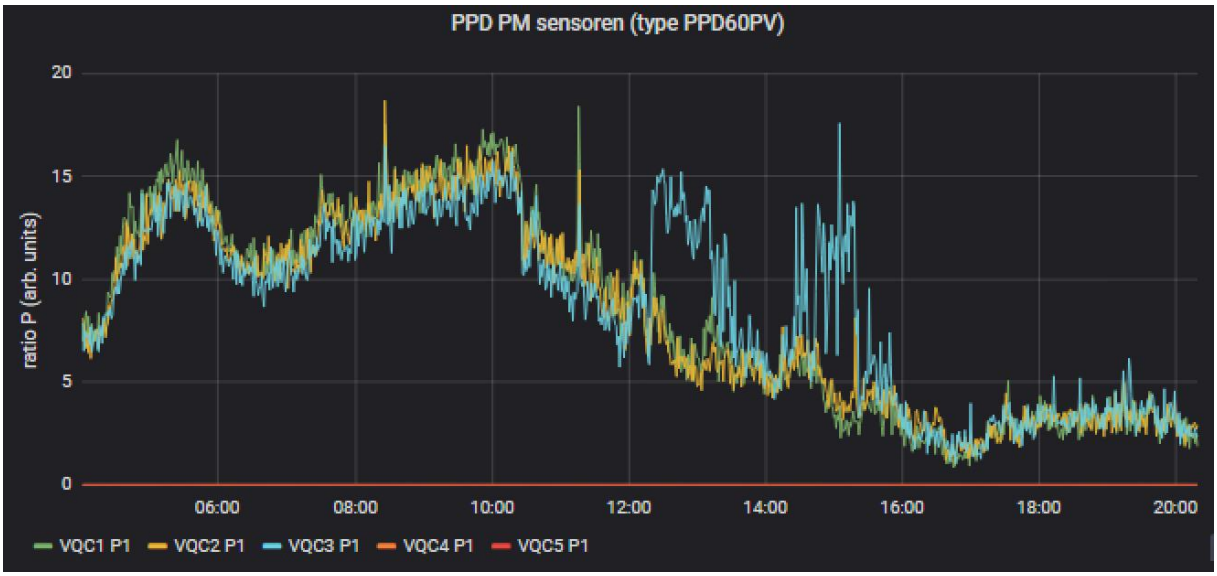



*Figure 11: Typical case of one sensor (VQC3) showing a different and abnormal behavior when compared to co-located sensors. Because sensors are co-located this signal would be considered invalid. If sensor had not been co-located it could have been marked as suspicious depending on the pre-existing knowledge about the site.*

One has to be aware that local sources (e.g. wood burning, local traffic, dust resuspensions, building works) can cause local additions to the background signal. Usually these types of pollution have specific types of signals, on top of the background signals, that allow experts to identify them.

Another issue with PM-sensors is that they are often influenced by (high) relative humidity. One should be aware of this and of the fact that there can be hyperlocal variations in relative humidity (e.g. morning fog over a grassy area or close to trees). Also the local positioning of sensor (shade vs in the sun) can have an impact on the sensor signal.

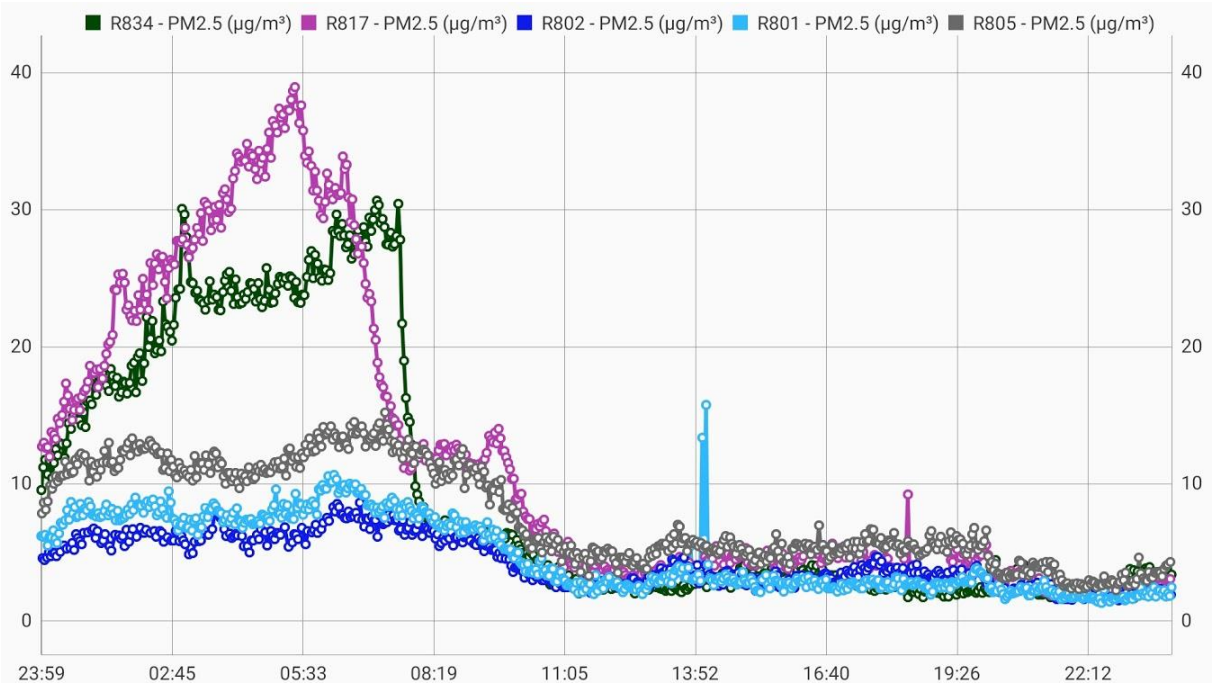



*Figure 12: Example of 5 SDS-sensor in the province of Antwerp. Sensors at R817 and R834 probably show elevated signals due to local increase in relative humidity caused by trees and grassy area near the sensor locations.*

Since PM-sensors measure light scattering certain light related issues can arise. E.g. insects or spiders can enter the sensor and can partially obstruct the inlet and/or create additional scattering. Spider webs or dust and plush particles have also been found inside sensors. Another issue that has been observed is light entering the sensor inlet and causing fake sensor signals. Depending on the time of day certain reflections could create this issue at specific time. To avoid this it is best to shield the inlet from direct light or to use sensors that have pathways that block direct light.

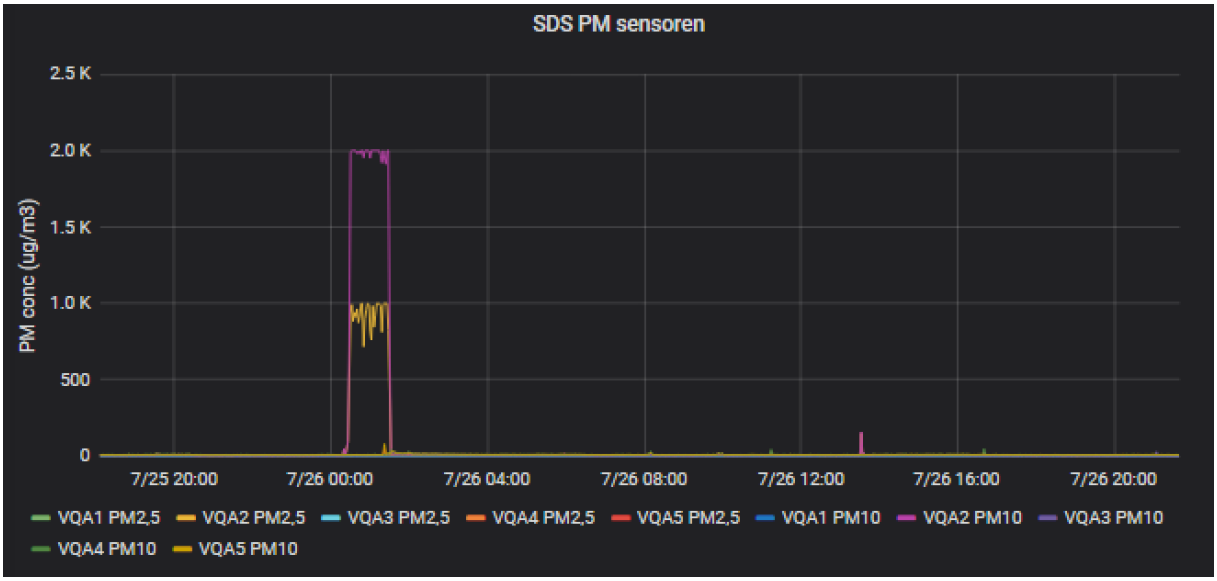



*Figure 13: A typical case of an SDS-sensor hitting maximum values, which could be caused by insects or spiders or by direct light entering the inlet.*

Also electronic interferences from nearby devices could have an effect on sensor behavior.

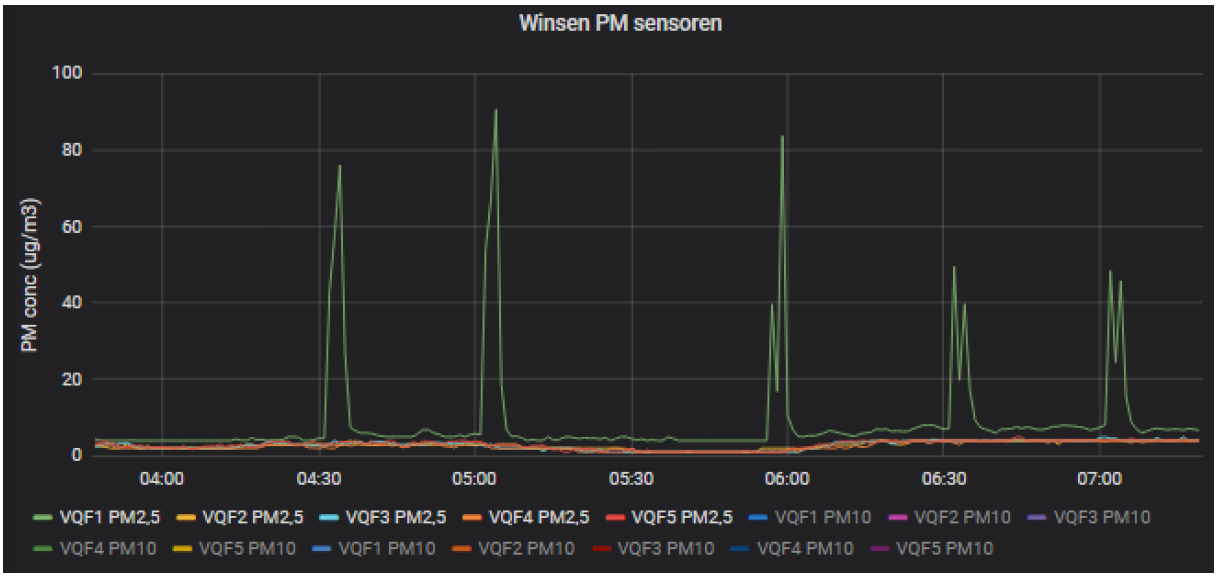



*Figure 14: Example of spikes on a Winsen PM sensor. The pattern (peaks 30 min apart) appear to indicate an electronic interference.*

**Gas sensors**: The data of the gas sensors are influenced by temperature, relative humidity and other interferents e.g. $O_3$ for $NO_2$ sensors and $NO_2$ for $O_3$ sensors. The $O_3$ sensors of Envea and Alphasense are designed to measure $O_3$ and $NO_2$.
Therefore comparison of the non-validated data with the reference method does not always provide an answer to the status of the data. This is illustrated in the figures below.

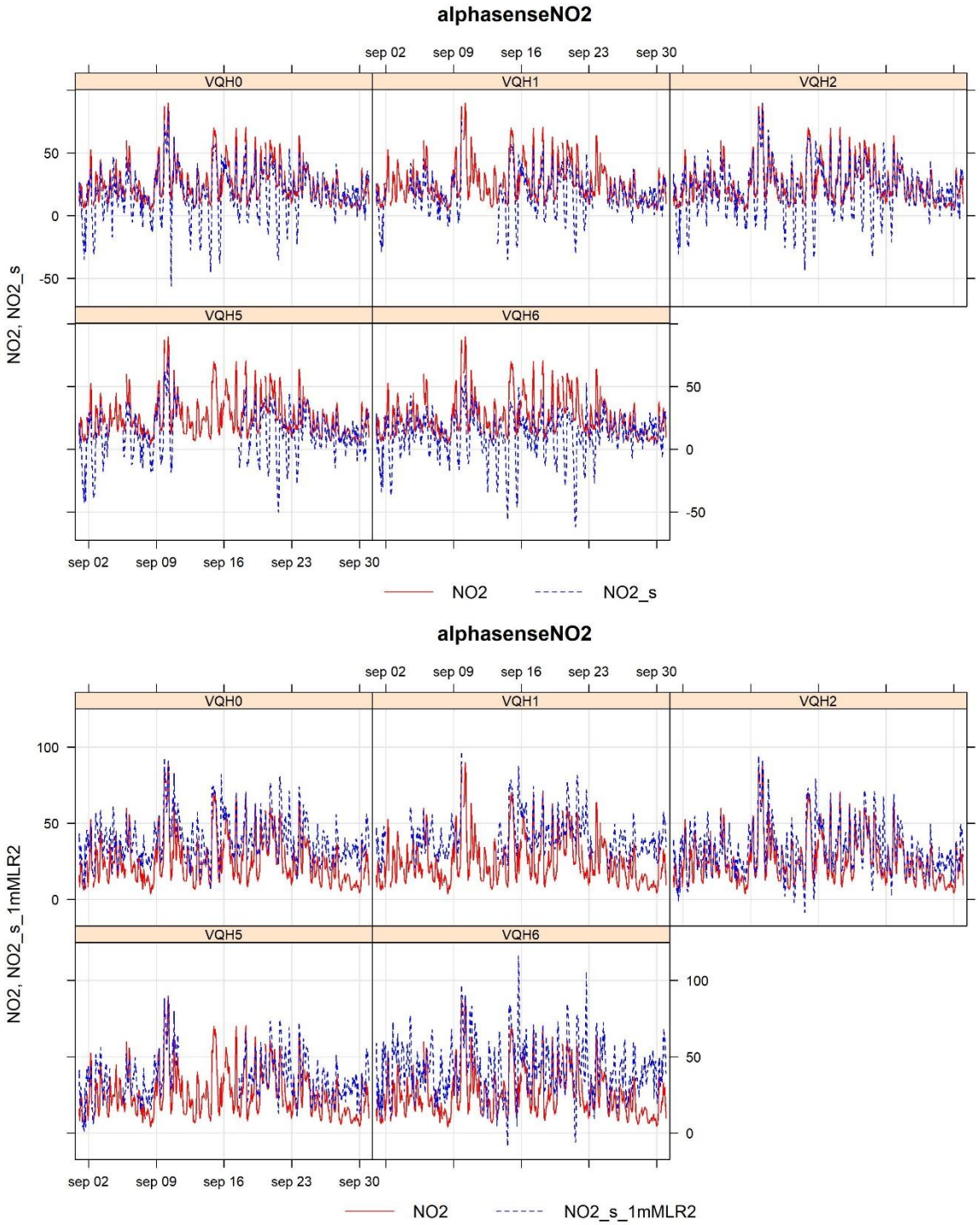



Figure 15: *Example of Alphasense $NO_2$ sensor(hourly values) where a lot of negative values are seen (figure above), which are no longer present (with a few exceptions) after calibration with parameters based on an Multiple Linear Calibraton model including variables $NO_2$, temperature and relative humidity (figure below).*

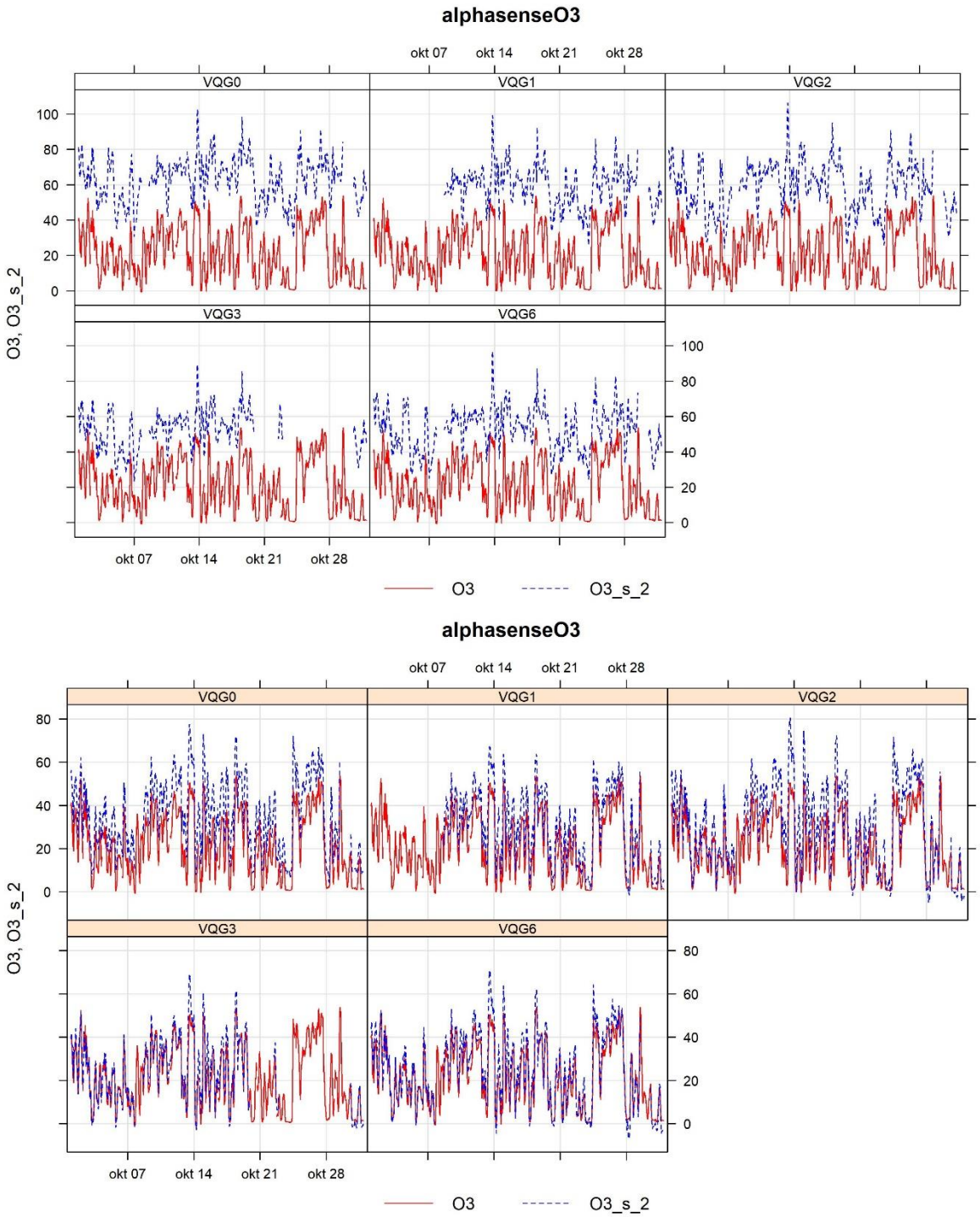



Figure 16: *Example of Alphasense $O_3$ sensor(hourly values) where the sensor data are much higher than the reference method (figure above). After correction (figure below) with the $NO_2$ sensor data, the data are more at the same level.*

Spikes after (re)start of the measurements were marked as invalid. The rest of validation mostly relied on comparing the data with the data of sensors of the same type. Anomalies were marked as suspicious. The Membrapor C20 $NO_2$ and the Membrapor $O_3$ sensors gave so much spikes and anomalies that manual validation was not feasible. The most

pronounced peaks were marked suspicious automatically when spikes were higher or lower than the established levels.

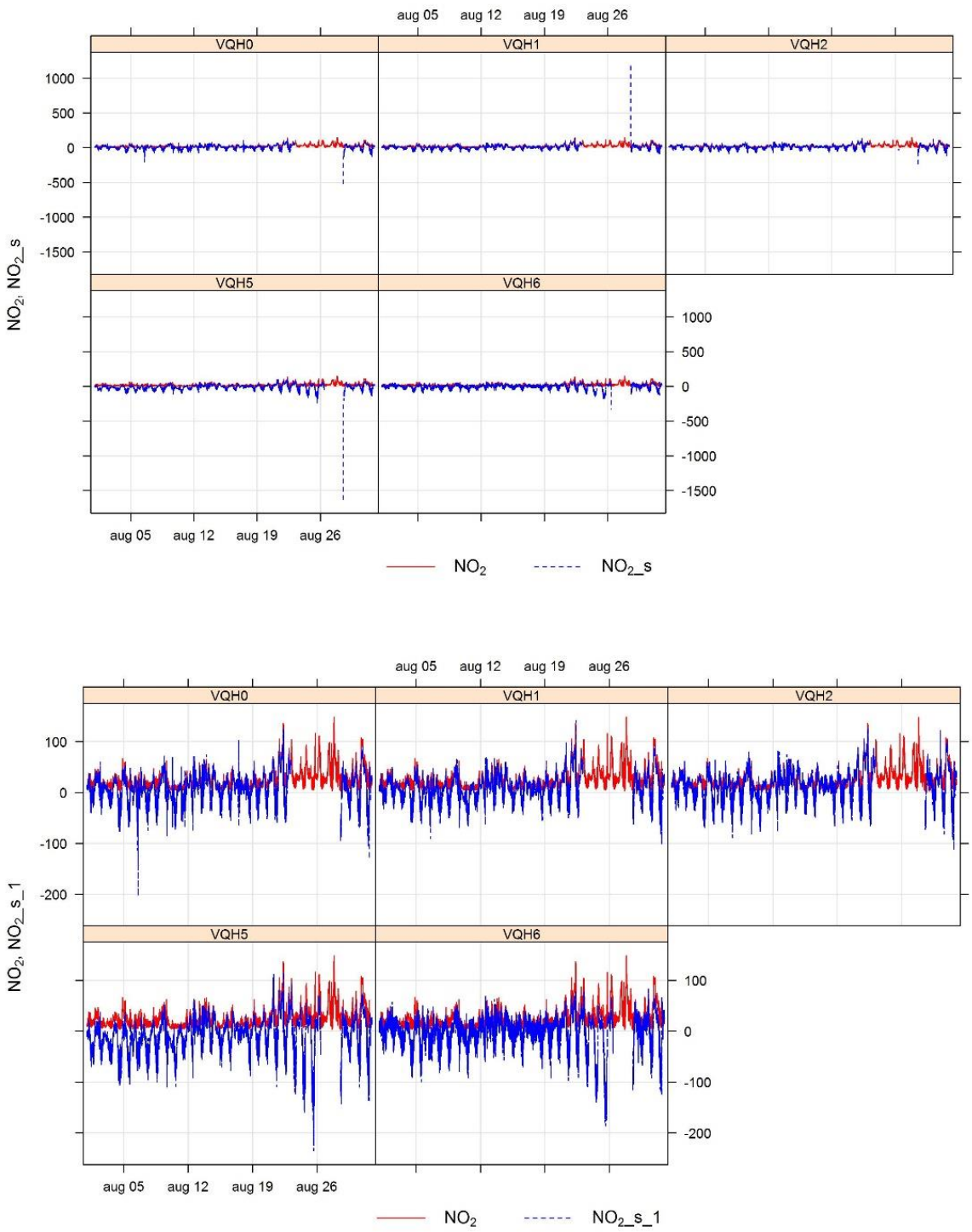



Figure 17: *Example of Alphasense $NO_2$ sensor (minute values) where some spikes after restart of the measurements (figure above) were marked as invalid (figure below).*

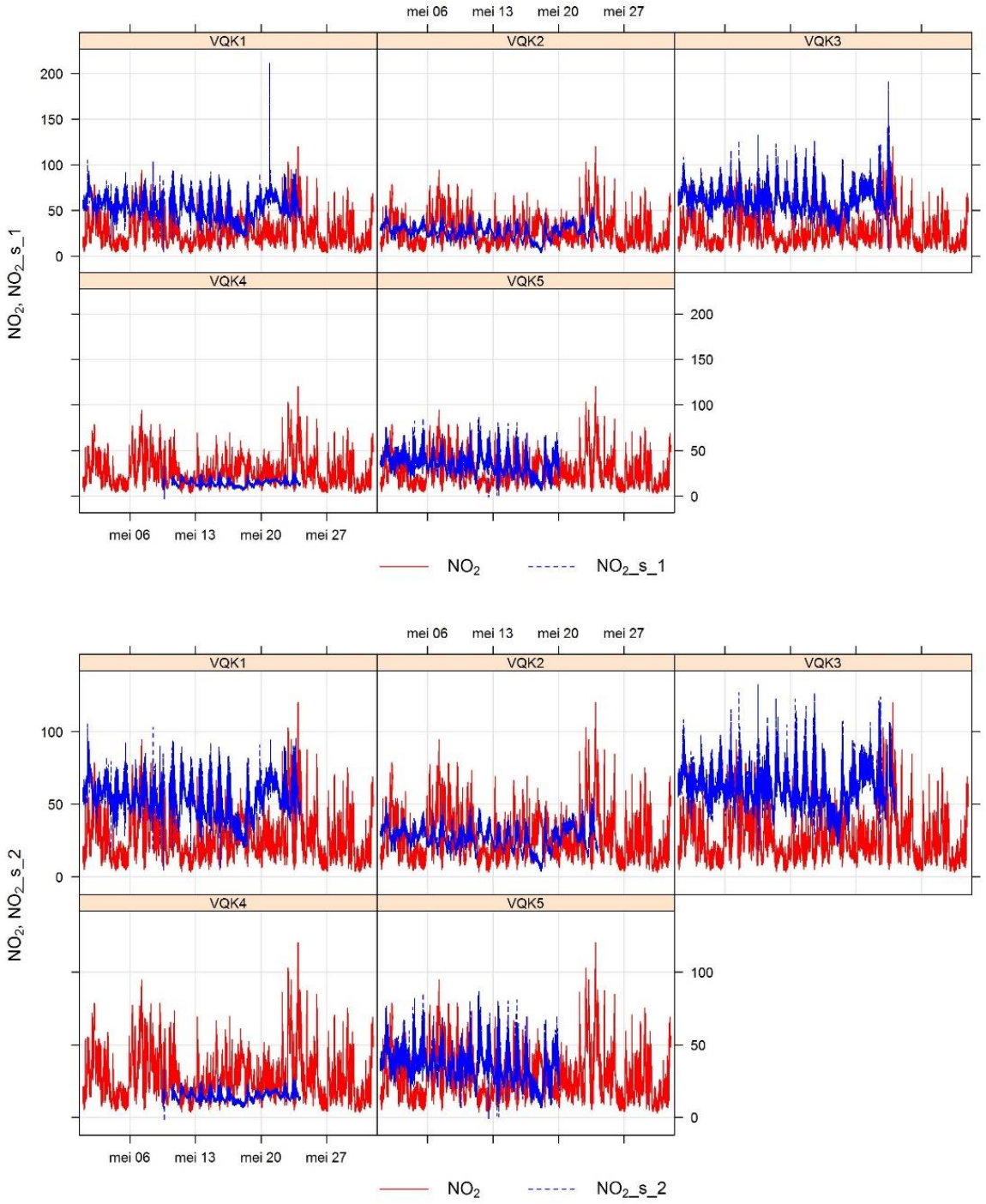



Figure 18: *Example of Membrapor C1 $NO_2$ sensors (minute values) where some spikes (figure above) were marked as suspicious (figure below)*

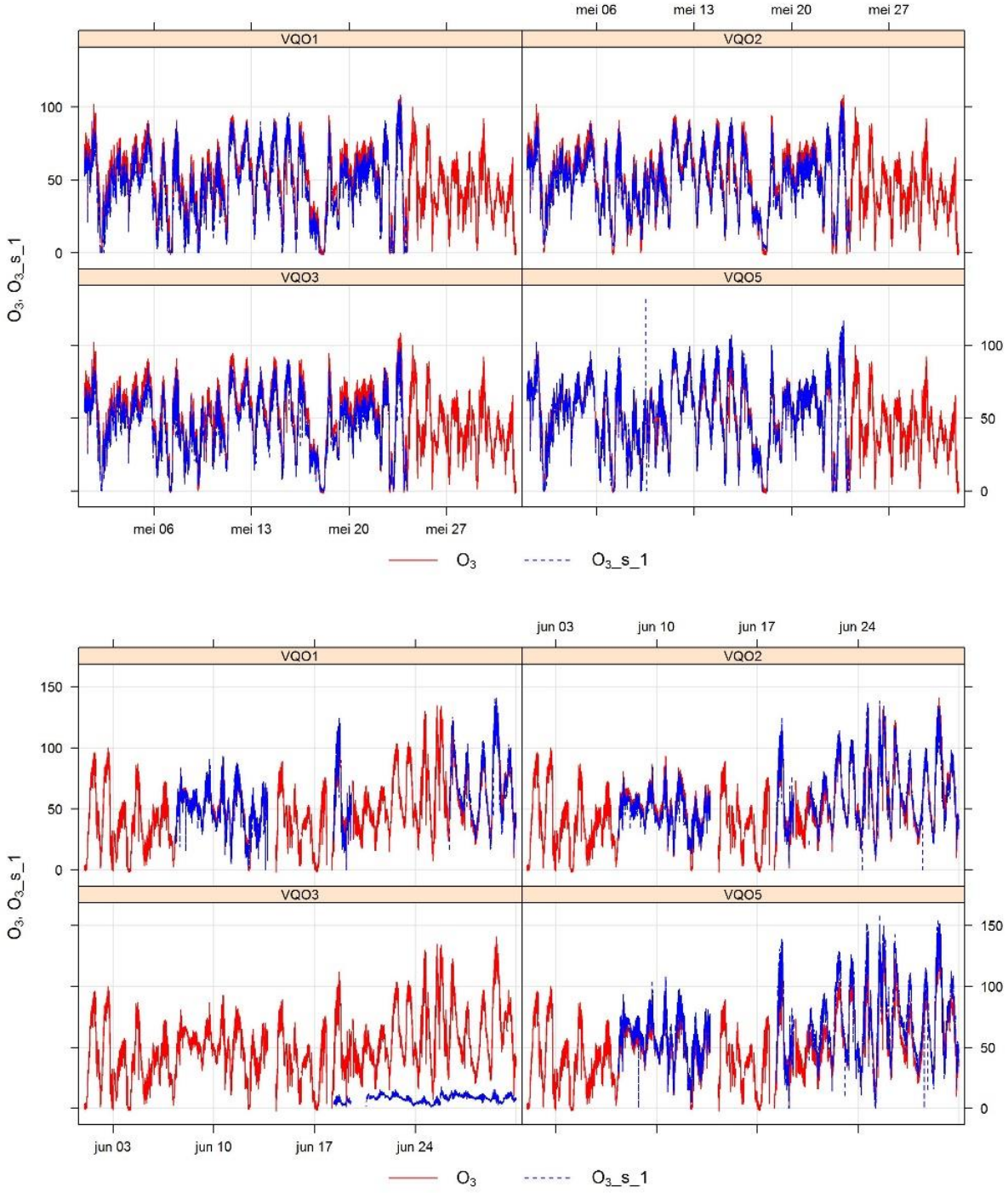



Figure 19: *Example of Aeroqual $O_3$ sensors (minute values) where the sensor VQO3 gave much lower values after restart of the measurement in June (figure above). The data were marked as suspicious (figure below).*

# Appendices

## Appendix 1: Sensor selection procedure

For the LIFE VAQUUMS project we aimed to select an array of low to medium budget air quality sensors to measure particle matter, nitrogen dioxide and ozone. These sensors will be tested in lab conditions and in a field experiment where they will be compared with reference instruments. In this way we can test the reliability of the sensors and give advice to governments and citizen science groups.

We took the following steps to select a workable number of relevant sensors.

**Step 1. Inventory build-up: desktop research**

In a first step we gathered relevant information about the existing know-how of air quality sensors. This knowledge can be available in several forms: scientific literature, other literature, project results of monitoring networks, results of European and worldwide projects, etc.

We started our desktop research in google scholar by combining different search entries (and/or): 'Sensor, PM, Particulate matter, fine dust, O3, ozone, NO2, nitrogen dioxide, comparison, low cost'. In this search, literature up to September 2017 was taken into account. In addition, the work of AQ SPEC was included. When all literature was gathered, we checked our literature database against the AirMonTech database, to avoid missing references. Moreover, all partners checked if there were sensors missing which would possibly indicate missing literature about these specific sensors.

**Step 2. Inventory build-up: Literature review**

The available literature was then reviewed by the project partners. Besides the results ($R^2$) of the studies we also gathered information like initiating body, design of project (lab or field tests), type of sensors, scale and duration, comparative testing at official stations, conclusions and lessons learned.

**Step 3. Longlist sensors**

Based on the literature review we selected those sensors that fulfilled certain quality requirements. Preferably, we based our selection on field tests. If a sensor was tested in the field it was selected for the longlist when the $R^2$ value was above 0.8 for PM sensors and above 0.6 for gas sensors. If a sensor was only tested in lab conditions, it was considered good enough when $R^2$ values were above 0.8 for both PM and gas sensors. These values were based on the partnership's expert opinion and reflect the overall poorer performance of gas sensors compared to particle sensor that was observed in literature. Additionally we did not use too stringent requirements as to not exclude too many sensors. As we see it, we will identify several use cases with varying quality requirements and thus also sensors with mediocre performance can ultimately have their place in some of these cases. Furthermore we recognize that sensors can display a wide range of $R^2$-values in different experiments. Again to not exclude sensors that would prove worth testing later on, we considered the highest reported $R^2$ for each sensor.

**Step 4. Internal expert consultation**

As a fourth step the longlist based on the literature was only circulated to the experts within the partner institutions. They were asked if the sensors in our longlist were either 'not worth testing', 'worth testing due to (expected) high quality', 'worth testing since they are widely used' or 'obsolete'. Secondly, the experts were also asked to indicate which sensors were missing in our

longlist and why they would be interesting to include them in the tests. It was also possible to give additional remarks regarding the sensor selection.

Based on the input of the internal experts, additional sensors were added to the longlist. A new desktop search and literature review were performed, specifically aiming to check available knowledge about these additional sensors. This way of working allowed us to further capture all relevant experience regarding the sensor selection within the partnership.

**Step 5. External expert consultation**

The new version of the longlist was then further distributed amongst other experts in this field, including experts of JRC, WG15 and WG42. They were asked the same two questions as the internal experts: 1. Do you think this sensor is worth testing? Why?; 2. Are there sensors missing in our longlist?

The experts' opinions were summarized in an extended version of the longlist.

**Step 6. Scoring the sensors**

Next, we scored the different sensors based on the experts' opinions, both internal and external. Every time a sensor was recommended ('worth testing due to high quality', 'worth testing since they are widely used') by an expert one point was added to the score. While the score was reduced by one point for every expert that discommended a sensor. This resulted in a new version of the longlist were every senor had a certain expert score.

**Step 7. Sensor selection**

Finally, a shortlist of sensors to test during the LIFE VAQUUMS project was selected based on all the information gathered.

The LIFE-program demanded knowledge build-up on mobile and portable devices. Since several system solutions for measuring air quality are fixed, they were excluded from this project. Furthermore, after consulting JRC it also became clear that many of these more expensive system sensors would be tested during the AQUILA project, wherefore it is not necessary to also test them in our project. In general, testing loose sensors was preferred over testing the same sensors included in boxes. Note that these boxes and system solutions are typically more expensive, making them less suitable for citizen science.

We followed several steps to select the sensors we will test during the VAQUUMS project.

1. A sensor from the longlist was selected based on the expert consultation if it scored 2 or more points. If two sensor types of the same manufacturer scored 2 points or higher, we selected only one to include in the tests. This was for example the case for the 'Plantower PMS sensors type 7003 and A003'. Since the 'Plantower PMS 7003' had a higher score and the models are identical in all ways but their physical size, we selected this one.
2. Next, the sensors on the longlist which were not selected by their expert score were re-evaluated by the project partners. In this way we ensured that no sensors which were worth testing were excluded from the shortlist due to a low or absent expert score.
3. The missing sensors that were advised by the experts could not be evaluated by our scoring system, since they were not scored by all experts. Therefore, these missing sensors were evaluated by the project partners. This resulted in the addition of two extra sensors to the shortlist. The 'Shinyei PPD42' was selected since it is widely used and the 'Membrapor NO2/C-20' was recommended and considered interesting to include in the tests.

Appendix 2: Assemblage and programming of sensors

## Alphasense NO2-B43F


### General information

Meet: NO2 in mV
Aansluiting: 0..5 Vout
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft twee meetkanalen via een spanningsuitgang (0..5 V). WE (Working Electrode) en Aux. Elk kanaal heeft een eigen offset in mV, Electronic Zero. De Sensitivity (mV/ppb) is voor beide kanalen dezelfde.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

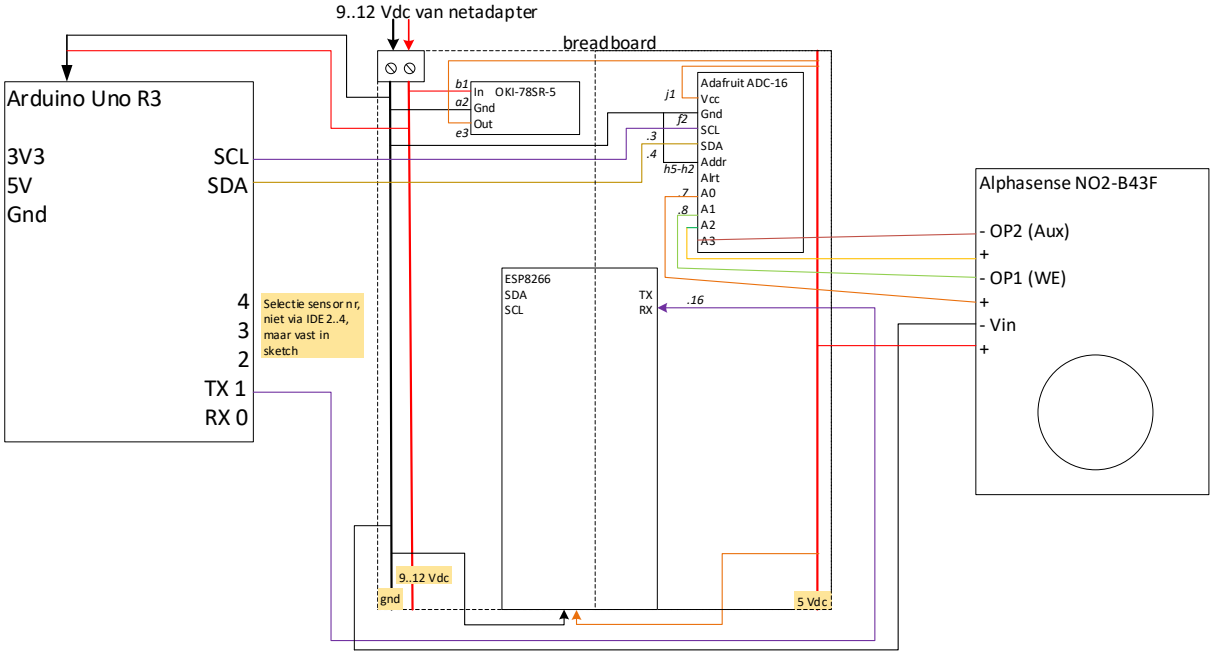
- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning WE kanaal in mV
- Gemeten spanning Aux kanaal in mV
- Concentratie NO2 WE kanaal in ppb
- Concentratie NO2 Aux kanaal in ppb
- Standaard afwijking van metingen gedurende meetperiode van 1 seconde voor WE kanaal

### Instelling

Sensor ID: 'H' geeft type sensor aan
Sensor nummer: Daar de Electronic Zero en Sensitivity verschillend zijn per s/n is er een uniek sketch per sensor. De mogelijkheid om het Sensor nummer in te stellen met jumper wires is dan ook niet relevant meer. Het Sensor nummer voor deze sensor is gedeclareerd als variabele in het sketch.

### Assemblage scheme

### Aansluitschema-instelling sensor



## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[WE_mV];[Aux_mV];[WE_ppb];[Aux_ppb];
[Stdev_WE];[<ETX>[CheckSum]
```

| | |
|---|---|
| <STX> | vast, Start of TeXt, 0x02 |
| VQ | vast, hoofding VaQuums |
| [SensorID] | sensor ID, voor Alphasense NO2 is dit 'H' (0x48) |
| [SensorNr] | sensor nummer zoals ingesteld in sketch |
| ; | vast scheidingskarakter ';' (0x3b) |
| [AantalInGem] | aantal sensormetingen voor de berekening van de volgende gemiddelden |
| [WE_mv] | gemiddelde meting WE in mV, '.' (0x2e) als decimaalteken |
| [Aux_mV] | gemiddelde meting Aux in mV |
| [WE_ppb] | gemiddelde concentratie NO2 op basis van WE mV gemiddelde |
| [Aux_ppb] | gemiddelde concentratie NO2 op basis van Aux mV gemiddelde |
| [Stdev_WE] | standaard afwijking van WE in mV op basis van WE metingen in mV |
| <ETX> | vast, End of TeXt, 0x03 |
| [CheckSum] | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch

***Op basis van sensor specificaties bij levering:***

| Sketches | s/n | | Sensitivity (mV/ppb) | Electr Zero WE (mV) | Electr Zero Aux (mV) |
|---|---|---|---|---|---|
| 180713VQ_AlphasenseOX_v100_H0_sn0661 | 20288 | 0661 | 0.280 | 235 | 239 |
| 180713VQ_AlphasenseOX_v100_H1_sn0659 | 20288 | 0659 | 0.266 | 247 | 239 |
| 180713VQ_AlphasenseOX_v100_H2_sn0658 | 20288 | 0658 | 0.260 | 218 | 227 |
| 180713VQ_AlphasenseOX_v100_H3_sn0656 | 20288 | 0656 | 0.264 | 225 | 225 |
| 180713VQ_AlphasenseOX_v100_H4_sn0657 | 20288 | 0657 | 0.256 | 228 | 240 |
| 180713VQ_AlphasenseOX_v100_H5_sn0662 | 20288 | 0662 | 0.269 | 230 | 226 |
| 180713VQ_AlphasenseOX_v100_H6_sn0660 | 20288 | 0660 | 0.272 | 236 | 228 |

***Voorbeeld sketch: 180713VQ_AlphasenseNO2_v100_H6_sn0660.ino***

```
/* 180713VQ_AlphasenseNO2_v100 - sn 20288 0660  - sens.nr. 6
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean Working Electrode and Aux concentration every 1sec to serial TX
 * based on read input voltages from Alphasense NO2 sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
 * in GAIN_ONE mode -> ADC range +/- 4.096V 0.125V/bit
 * Working Electrode OP1 sensor -> ADC A0..A1
 * Aux Electrode OP2 sensor -> ADC A2..A3
 *   -> max accesptable NO2 concentration 963ppb minimum
 *   -> for connected sensor 20288 0660: max conc NO2 =1024ppb, 0.46ppb/bit
 * SensorId fixed in sketch (not with IDE 2..4) as Sensitivity and Electronic zero
 * are sensor dependent and set in sketch
 *
 * 13/07/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// variables to set for every individual connected gassensor
```

```
unsigned int uiSensNr = 6;       // fixed in sketch 0..7, replaces selection with IDE 2..4
String sSnSens ="202880660";     // serial number of connected sensor
float fSensitivity =0.272;       // sensor Sensitivity (mV/ppb) from spec sheet
int iElectZeroWE =236;           // electronic zero Working Electrode (mV)
int iElectZeroAux = 228;         // electronic zero Aux (mV)

// variables to set for every connected gassensor type
const char cSensId = 'H';              // Sensor Id : G = Alphasense OX sensor
const float fVadcMultiplier = 0.125;   // multiplier of ADC, Voltage of 1 bit of ADC (ADC used
in GAIN_ONE)

// global variables, not to be set for individual sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings

unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSumWE;    // sum sensor measurement Working Electrode
unsigned long ulSensMeasSumAux;   // sum sensor measurement Aux electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
                                  // Stdev only calculated for WE
char cCheckSum;
String dataString = "";

void setup(void)
{
  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  Serial.print("180713VQ_AlphasenseNO2_v100 sensorNr: ");
  Serial.print(uiSensNr);
  Serial.print("  sn: ");
  Serial.println(sSnSens);
  Serial.print(" electr zero WE: ");
  Serial.print(iElectZeroWE);
  Serial.print("mV  electr zero Aux: ");
  Serial.print(iElectZeroAux);
  Serial.println("mV");
  Serial.print(" sensitivity in spec: ");
  Serial.print(fSensitivity);
  Serial.print("mV/ppb  1-bit= ");
  Serial.print((1/fSensitivity)/(1/fVadcMultiplier));
  Serial.println("ppb");

  Serial.println();
  Serial.println("capturing data stream..");

  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range to GAIN_ONE: +/- 4.096V (1 bit = 0,125V) and start reading
  ads1115.setGain(GAIN_ONE);
  ads1115.begin();

  // init variables
  ulSensMeasSumWE =0;
  ulSensMeasSumAux =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  int16_t i16ResultADC23;
  float fSensMeasMeanWE;
  float fSensMeasMeanAux;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdevWE;

  ulActMilli = millis();
```

```
  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values, 1bit = 0,125V
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSumWE +=i16ResultADC01;
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier;

  // read sensor measurement Aux - OP2 connected to A2 - A3
  i16ResultADC23 =ads1115.readADC_Differential_2_3();
  ulSensMeasSumAux +=i16ResultADC23;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg) and convert to concentration
    fSensMeasMeanWE =(float(ulSensMeasSumWE)/uiNrInAvg) *fVadcMultiplier;
    fSensMeasMeanAux =(float(ulSensMeasSumAux)/uiNrInAvg) *fVadcMultiplier;

    // calculate stdev for WE
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMeanWE;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdevWE =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMeanWE);
    dataString += ";";
    dataString += String(fSensMeasMeanAux);
    dataString += ";";
    dataString += String((fSensMeasMeanWE -iElectZeroWE) /fSensitivity);
    dataString += ";";
    dataString += String((fSensMeasMeanAux -iElectZeroAux) /fSensitivity);
    dataString += ";";
    dataString += String(fStdevWE);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
  ulSensMeasSumWE =0;
  ulSensMeasSumAux =0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

## Citytech NO2 3E50


### General information

Meet: $NO_2$
Aansluiting: 4..20mA
Voeding: afzonderlijke 24 Vdc voeding

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft één meetkanaal via een stroomuitgang (4..20mA). Geeft ook negatieve waardes, waarden kleiner dan 4mA.
Testen en ontwikkeling met Citytech NO2 3E50 sensor met serienummer: 12994872-068
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning door ADC in mV, na omzetting van mA naar mV met precisie weerstand
- Concentratie $NO_2$ in ppb (omzetting spanning ADC)
- Standaard afwijking van [spanningsmetingen -1000mV] gedurende meetperiode van 1 sec.

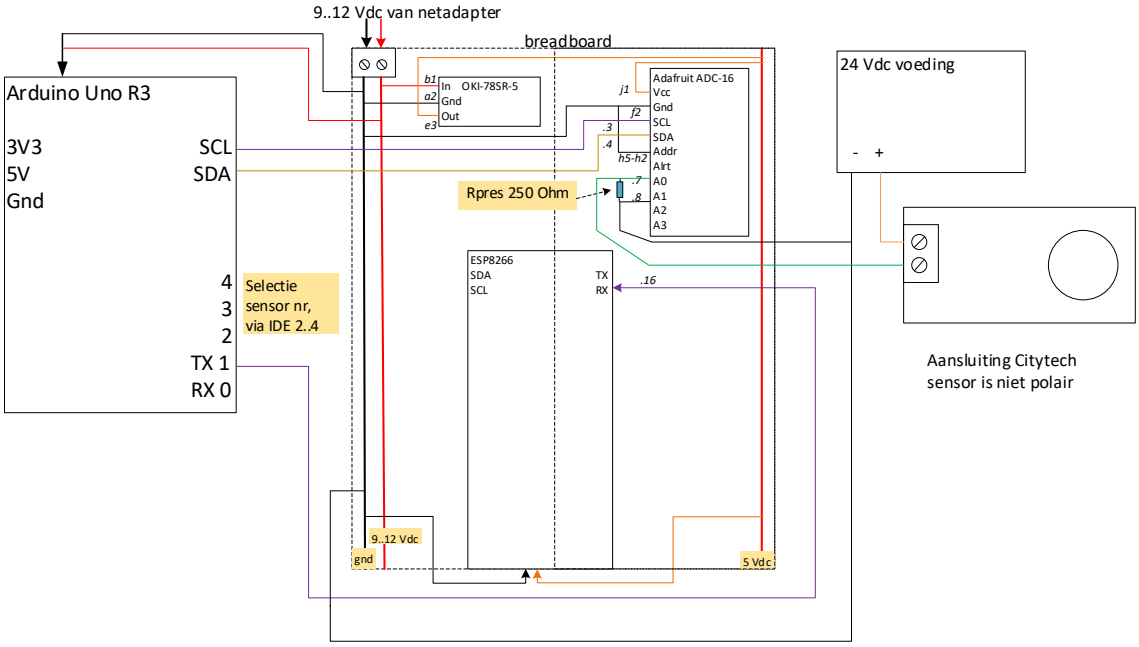
### Instelling

Sensor ID: 'I' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

### Assemblage scheme

**Aansluitschema – instelling sensor**




De sensoren zijn volledig gekalibreerd verscheept. De fabriekskalibratie werd behouden.

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[NO2_mV];[NO2_ppb];[Stdev_NO2_mV];
[<ETX>];[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Citytech NO2-3E50 is dit 'I' (0x49) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ';' (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[NO2_mV]` | gemiddelde meting in mV, na omzetting mA -> mV via Rpres (250 Ohm) |
| `[NO2_ppb]` | gemiddelde concentratie $NO_2$ in ppb op basis van mV gemiddelde |
| `[Stdev_NO2_mV]` | standaard afwijking van `[NO2_mV] -1000mV`, de offset wordt tijdelijk afgetrokken om een relevantere Stdev waarde te krijgen |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180817VQ_CitytechNO2_v100.ino

```
/* 180817VQ_CitytechNO2-3E50_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean O3 concentration every 1sec to serial TX
 * based on read 4..20mA input current, converted to voltage with precission resistor, from
Citytech O3 3E 1 F sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
 * in GAIN_TWO 0.0625mV/bit  -> typical 0.781ppb/bit, max conc 12500ppb   to obtain better
resolution,
 * max. conc with higher GAIN would be too low
 * sensor 4..20mA current, measurements can go below 4mA -> 0..5V voltage -> ADC A0..A1
```

```
 * sensor output = 21mA is case of error:
 *   - output < 3.8mA
 *   - output > 20.5mA
 *   - operated > 5Â°C outside temperature range -40..+50Â°C
 * error only supported in GAIN_TWOTHIRD in other GAIN_ settings the error will blow the ADC..
 *
 * 17/08/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// sensor depending variables
unsigned int uiFS =50000;       // sensor full scale (ppb) from spec sheet NO2 3E 50
float fResistIU =250.0;         // resistor used for current -> voltage (Ohm)

// global variables, not to be set for individual sensor
const char cSensId = 'I';              // Sensor Id : I = Citytech NO2 3E 50 sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings
const float fVadcMultiplier = 0.0625;         // multiplier of ADC, mVoltage of 1 bit of ADC
(ADC used in GAIN_TWO)

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSum;      // sum sensor measurement Working Electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
char cCheckSum;
String dataString = "";

void setup(void)
{
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180817VQ_CitytechNO2-3E50_v100 sensorNr: ");
  Serial.println(uiSensNr);

  Serial.print(" ADC ");
  Serial.print(fVadcMultiplier);
  Serial.print(" mV/bit,  ");
  Serial.print(" 1-bit= ");
  Serial.print( fVadcMultiplier*uiFS/((20-4)*fResistIU) );
  Serial.println("ppb");
  Serial.println(" max. NO2 conc. with ADC settings= 12500ppb");

  Serial.println();
  Serial.println("capturing data stream..");

  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range and start reading
  //  GAIN_TWOTHIRDS +/- 6.144V   1 bit = 0,1875mV = fVadcMultiplier
  //  GAIN_ONE       +/- 4.096V           0.125mV
  //  GAIN_TWO       +/- 2.048V           0.0625mV   <---
  //  GAIN_FOUR      +/- 1.024V           0.03125mV
  //  GAIN_EIGHT     +/- 0.512V           0.015625mV
  //  GAIN_SIXTEEN   +/- 0.256V           0.0078125mV
  //
  ads1115.setGain(GAIN_TWO);
```

```
  ads1115.begin();

  // init variables
  ulSensMeasSum =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  float fSensMeasMean;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdev;
  float fSensConcPPB;

  ulActMilli = millis();

  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSum +=i16ResultADC01;
  // substract offset 4mA: -1000.0mV for better Stdev calculation
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier -1000.0;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg), substract offest 4mA: -1000.0mV for better Stdev calculation
    fSensMeasMean =(float(ulSensMeasSum)/uiNrInAvg) *fVadcMultiplier -1000.0;

    // calculate stdev for raw mV measures
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMean;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdev =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // convert raw mA to ppb:  fResist (mV -> mA), scaling to FS = 20mA, offset = 4mA -> range
16mA
    fSensConcPPB =(fSensMeasMean/fResistIU) *(uiFS/16);
    // add offest (1000.mV) back to mean for string output
    fSensMeasMean =fSensMeasMean +1000.0;

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMean);
    dataString += ";";
    dataString += String(fSensConcPPB);
    dataString += ";";
    dataString += String(fStdev);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulSensMeasSum =0;

    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}
```

```
// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

## Envea Cairclip NO2

This is a plug and play sensor.

## Membrapor NO2/C-1


### General information

Meet: stikstofdioxide $NO_2$
Aansluiting: 4..20mA
Voeding: afzonderlijke 24 Vdc voeding

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft één meetkanaal via een stroomuitgang (4..20mA).
Testen en ontwikkeling met Membrapor NO2-C1 sensor met serienummer: 18227377.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning door ADC in mV, na omzetting van mA naar mV met precisie weerstand
- Concentratie $O_3$ in ppb (omzetting spanning ADC)
- Standaard afwijking van [spanningsmetingen -1000mV] gedurende meetperiode van 1 sec.

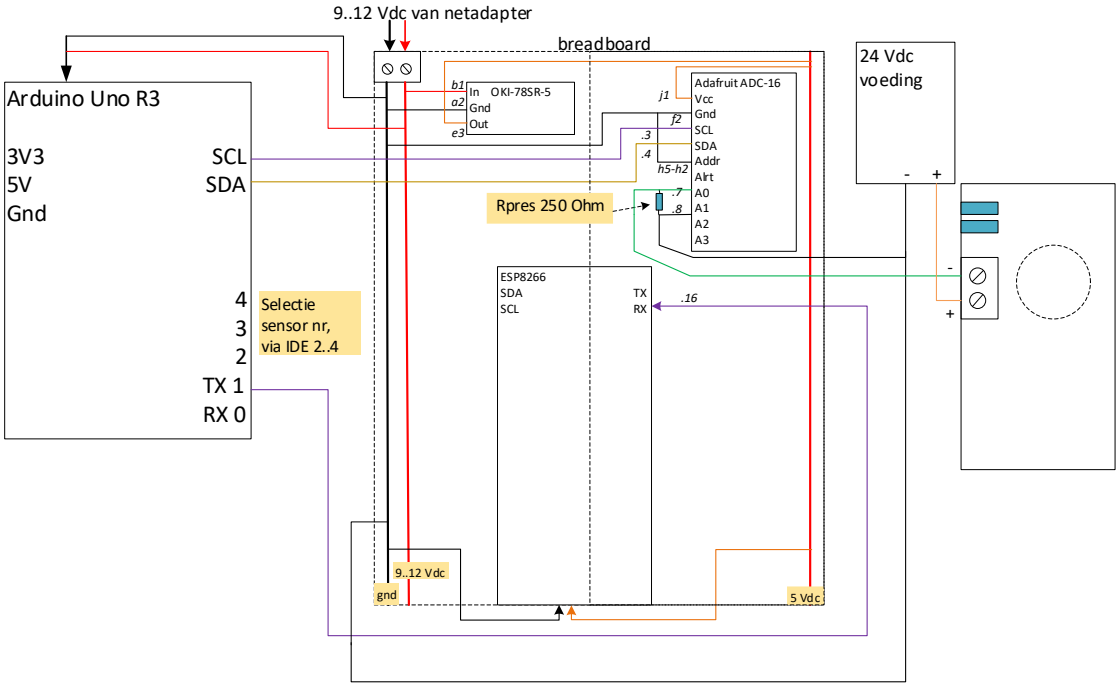
### Instelling

Sensor ID: 'K' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

### Assemblage scheme

**Aansluitschema – instelling sensor**




De fabriekskalibratie werd behouden.

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[NO2_mV];[NO2_ppb];[Stdev_NO2_mV];
[<ETX>];[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Membrapor NO2-C1 is dit ‘K’ (0x4B) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[NO2_mV]` | gemiddelde meting in mV, na omzetting mA -> mV via Rpres (250 Ohm) |
| `[NO2_ppb]` | gemiddelde concentratie $NO_2$ in ppb op basis van mV gemiddelde |
| `[Stdev_NO2_mV]` | standaard afwijking van `[NO2_mV] -1000mV`, de offset wordt tijdelijk afgetrokken om een relevantere Stdev waarde te krijgen |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180823VQ_MembraporNO2-C1_v100.ino

```
/* 180823VQ_MembraporNO2-C1_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean NO2 concentration every 1sec to serial TX
 * based on read 4..20mA input current, converted to voltage with precision resistor, from
Membrapor NO2 C1 sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
 * in GAIN_ONE 0.125mV/bit  -> typical 0.031ppb/bit, max conc 770ppb   to obtain better
resolution
```

```
 * at startup the sensor output peaks above 2.1V, a higher GAIN could damage the ADC at
startup
 * sensor 4..20mA current, measurements can go below 4mA -> 0..5V voltage -> ADC A0..A1
 *
 * 23/08/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// sensor depending variables
unsigned int uiFS =1000;        // sensor full scale (ppb) from spec sheet NO2 C1
float fResistIU =250.0;         // resistor used for current -> voltage (Ohm)

// global variables, not to be set for individual sensor
const char cSensId = 'K';              // Sensor Id : K = Membrapor NO2 C1 sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings
const float fVadcMultiplier = 0.125;          // multiplier of ADC, mVoltage of 1 bit of ADC
(ADC used in GAIN_ONE)

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSum;      // sum sensor measurement Working Electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
char cCheckSum;
String dataString = "";

void setup(void)
{
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180823VQ_MembraporNO2-C1_v100 sensorNr: ");
  Serial.println(uiSensNr);

  Serial.print(" ADC ");
  Serial.print(fVadcMultiplier);
  Serial.print(" mV/bit,  ");
  Serial.print(" 1-bit= ");
  Serial.print( fVadcMultiplier*uiFS/((20-4)*fResistIU) );
  Serial.println("ppb");
  Serial.println(" max. NO2 conc. with ADC settings= 770ppb");

  Serial.println();
  Serial.println("capturing data stream..");

  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range and start reading
  //  GAIN_TWOTHIRDS +/- 6.144V   1 bit = 0,1875mV = fVadcMultiplier
  //  GAIN_ONE       +/- 4.096V           0.125mV   <---
  //  GAIN_TWO       +/- 2.048V           0.0625mV
  //  GAIN_FOUR      +/- 1.024V           0.03125mV
  //  GAIN_EIGHT     +/- 0.512V           0.015625mV
  //  GAIN_SIXTEEN   +/- 0.256V           0.0078125mV
  //
  ads1115.setGain(GAIN_ONE);
  ads1115.begin();
```

```
  // init variables
  ulSensMeasSum =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  float fSensMeasMean;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdev;
  float fSensConcPPB;

  ulActMilli = millis();

  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSum +=i16ResultADC01;
  // substract offset 4mA: -1000.0mV for better Stdev calculation
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier -1000.0;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg), substract offest 4mA: -1000.0mV for better Stdev calculation
    fSensMeasMean =(float(ulSensMeasSum)/uiNrInAvg) *fVadcMultiplier -1000.0;

    // calculate stdev for raw mV measures
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMean;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdev =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // convert raw mA to ppb:  fResist (mV -> mA), scaling to FS = 20mA, offset = 4mA -> range
16mA
    fSensConcPPB =(fSensMeasMean/fResistIU) *(uiFS/16);
    // add offest (1000.mV) back to mean for string output
    fSensMeasMean =fSensMeasMean +1000.0;

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMean);
    dataString += ";";
    dataString += String(fSensConcPPB);
    dataString += ";";
    dataString += String(fStdev);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulSensMeasSum =0;

    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}
```

```
// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

## Membrapor NO2/C-20


### General information

Meet: stikstofdioxide $NO_2$
Aansluiting: 4..20mA
Voeding: afzonderlijke 24 Vdc voeding

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft één meetkanaal via een stroomuitgang (4..20mA).
Testen en ontwikkeling met Membrapor NO2-C20 sensor met serienummer: 18129120.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning door ADC in mV, na omzetting van mA naar mV met precisie weerstand
- Concentratie $O_3$ in ppb (omzetting spanning ADC)
- Standaard afwijking van [spanningsmetingen -1000mV] gedurende meetperiode van 1 sec.

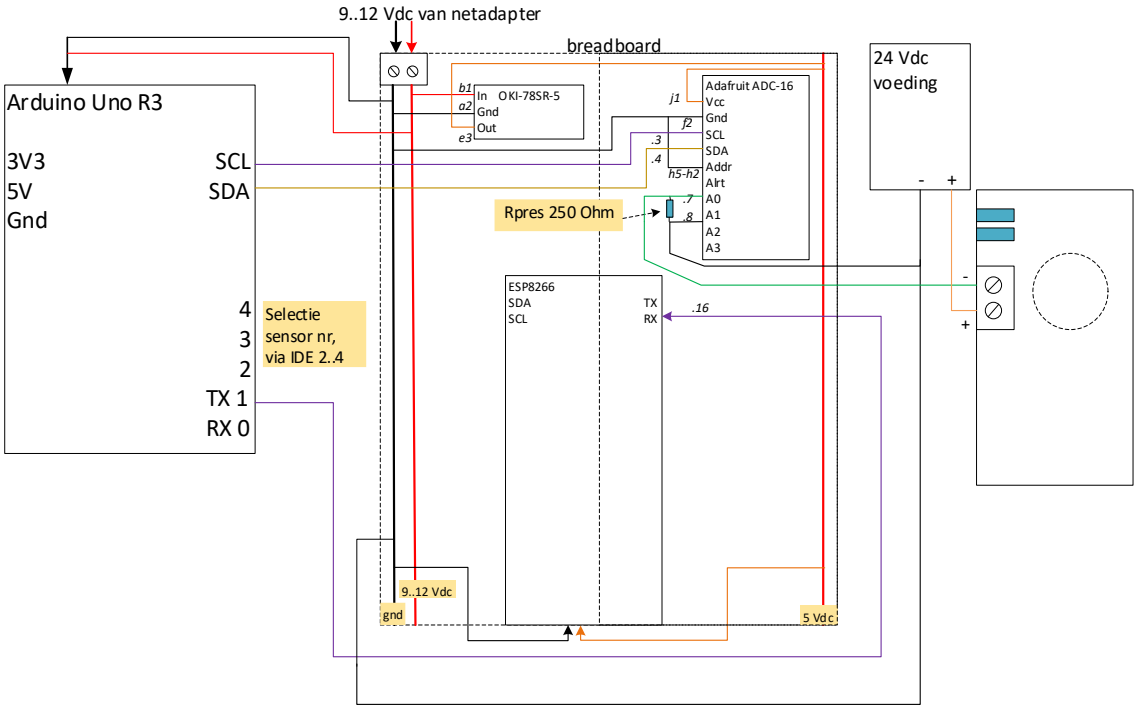
### Instelling

Sensor ID: 'L' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

### Assemblage scheme

**Aansluitschema – instelling sensor**




De fabriekskalibratie werd behouden.

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[NO2_mV];[NO2_ppb];[Stdev_NO2_mV];
[<ETX>];[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Membrapor NO2-C20 is dit ‘L’ (0x4C) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[NO2_mV]` | gemiddelde meting in mV, na omzetting mA -> mV via Rpres (250 Ohm) |
| `[NO2_ppb]` | gemiddelde concentratie $NO_2$ in ppb op basis van mV gemiddelde |
| `[Stdev_NO2_mV]` | standaard afwijking van `[NO2_mV] -1000mV`, de offset wordt tijdelijk afgetrokken om een relevantere Stdev waarde te krijgen |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180823VQ_MembraporNO2-C20_v100.ino

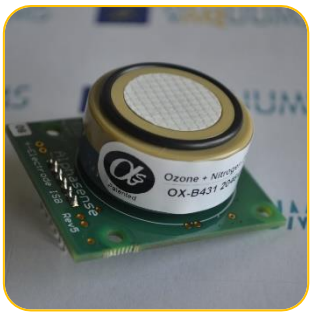
```
/* 180823VQ_MembraporNO2-C20_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean NO2 concentration every 1sec to serial TX
 * based on read 4..20mA input current, converted to voltage with precision resistor, from
Membrapor NO2 C20 sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
```

```
 * in GAIN_TWO 0.0625mV/bit  -> typical 0.016ppb/bit, max conc 5200ppb   to obtain better
resolution
 * sensor 4..20mA current, measurements can go below 4mA -> 0..5V voltage -> ADC A0..A1
 *
 * 23/08/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// sensor depending variables
unsigned int uiFS =20000;        // sensor full scale (ppb) from spec sheet NO2 C20
float fResistIU =250.0;         // resistor used for current -> voltage (Ohm)

// global variables, not to be set for individual sensor
const char cSensId = 'L';              // Sensor Id : L = Membrapor NO2 C20 sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings
const float fVadcMultiplier = 0.0625;         // multiplier of ADC, mVoltage of 1 bit of ADC
(ADC used in GAIN_TWO)

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSum;      // sum sensor measurement Working Electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
char cCheckSum;
String dataString = "";

void setup(void)
{
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180823VQ_MembraporNO2-C20_v100 sensorNr: ");
  Serial.println(uiSensNr);

  Serial.print(" ADC ");
  Serial.print(fVadcMultiplier);
  Serial.print(" mV/bit,  ");
  Serial.print(" 1-bit= ");
  Serial.print( fVadcMultiplier*uiFS/((20-4)*fResistIU) );
  Serial.println("ppb");
  Serial.println(" max. NO2 conc. with ADC settings= 5200ppb");

  Serial.println();
  Serial.println("capturing data stream..");

  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range and start reading
  //  GAIN_TWOTHIRDS +/- 6.144V   1 bit = 0,1875mV = fVadcMultiplier
  //  GAIN_ONE       +/- 4.096V           0.125mV
  //  GAIN_TWO       +/- 2.048V           0.0625mV   <---
  //  GAIN_FOUR      +/- 1.024V           0.03125mV
  //  GAIN_EIGHT     +/- 0.512V           0.015625mV
  //  GAIN_SIXTEEN   +/- 0.256V           0.0078125mV
  //
  ads1115.setGain(GAIN_TWO);
  ads1115.begin();
```

```
  // init variables
  ulSensMeasSum =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  float fSensMeasMean;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdev;
  float fSensConcPPB;

  ulActMilli = millis();

  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSum +=i16ResultADC01;
  // substract offset 4mA: -1000.0mV for better Stdev calculation
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier -1000.0;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg), substract offest 4mA: -1000.0mV for better Stdev calculation
    fSensMeasMean =(float(ulSensMeasSum)/uiNrInAvg) *fVadcMultiplier -1000.0;

    // calculate stdev for raw mV measures
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMean;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdev =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // convert raw mA to ppb:  fResist (mV -> mA), scaling to FS = 20mA, offset = 4mA -> range
16mA
    fSensConcPPB =(fSensMeasMean/fResistIU) *(uiFS/16);
    // add offest (1000.mV) back to mean for string output
    fSensMeasMean =fSensMeasMean +1000.0;

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMean);
    dataString += ";";
    dataString += String(fSensConcPPB);
    dataString += ";";
    dataString += String(fStdev);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulSensMeasSum =0;

    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}
```

```
// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

Alphasense OX-B431



## General information

Meet: Ox in mV
Aansluiting: 0..5 Vout
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft twee meetkanalen via een spanningsuitgang (0..5 V). WE (Working Electrode) en Aux. Elk kanaal heeft een eigen offset in mV, Electronic Zero. De Sensitivity (mV/ppb) is voor beide kanalen dezelfde.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

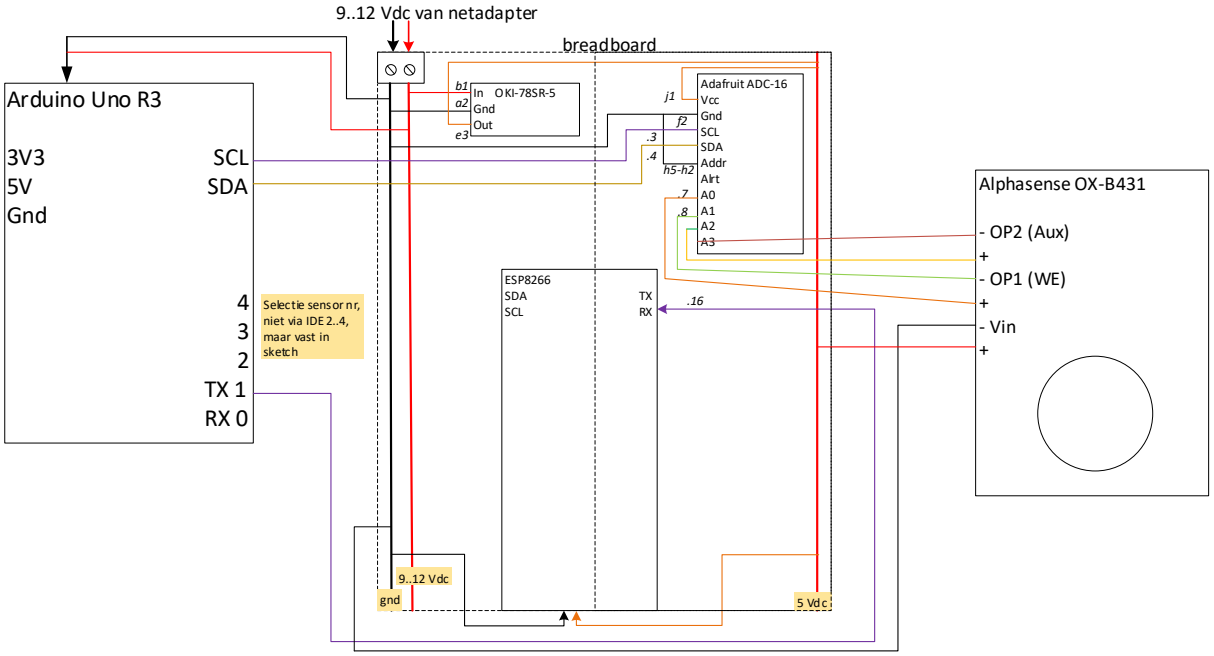
- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning WE kanaal in mV
- Gemeten spanning Aux kanaal in mV
- Concentratie Ox WE kanaal in ppb
- Concentratie Ox Aux kanaal in ppb
- Standaard afwijking van metingen gedurende meetperiode van 1 seconde voor WE kanaal

### Instelling

Sensor ID: 'G' geeft type sensor aan
Sensor nummer: Daar de Electronic Zero en Sensitivity verschillend zijn per s/n is er een uniek sketch per sensor. De mogelijkheid om het Sensor nummer in te stellen met jumper wires is dan ook niet relevant meer. Het Sensor nummer voor deze sensor is gedeclareerd als variabele in het sketch.

## Assemblage scheme

### Aansluitschema – instelling sensor




## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[WE_mV];[Aux_mV];[WE_ppb];[Aux_ppb];
[Stdev_WE];[<ETX>[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Alphasense OX is dit ‘G’ (0x47) |
| `[SensorNr]` | sensor nummer zoals ingesteld in sketch |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal sensormetingen voor de berekening van de volgende gemiddelden |
| `[WE_mv]` | gemiddelde meting WE in mV, ‘.’ (0x2e) als decimaalteken |
| `[Aux_mV]` | gemiddelde meting Aux in mV |
| `[WE_ppb]` | gemiddelde concentratie Ox op basis van WE mV gemiddelde |
| `[Aux_ppb]` | gemiddelde concentratie Ox op basis van Aux mV gemiddelde |
| `[Stdev_WE]` | standaard afwijking van WE in mV op basis van WE metingen in mV |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch:

***Op basis van sensor specificaties bij levering:***

| Sketches | s/n | | Sensitivity (mV/ppb) | Electr Zero WE (mV) | Electr Zero Aux (mV) |
|---|---|---|---|---|---|
| 180712VQ_AlphasenseOX_v100_G0_sn1716 | 20464 | 1716 | 0.355 | 224 | 233 |
| 180712VQ_AlphasenseOX_v100_G1_sn1720 | 20464 | 1720 | 0.369 | 236 | 232 |
| 180712VQ_AlphasenseOX_v100_G2_sn1715 | 20464 | 1715 | 0.345 | 227 | 240 |
| 180712VQ_AlphasenseOX_v100_G3_sn1717 | 20464 | 1717 | 0.385 | 231 | 213 |
| 180712VQ_AlphasenseOX_v100_G4_sn1719 | 20464 | 1719 | 0.377 | 229 | 239 |
| 180712VQ_AlphasenseOX_v100_G5_sn1714 | 20464 | 1714 | 0.368 | 227 | 217 |
| 180712VQ_AlphasenseOX_v100_G6_sn1713 | 20464 | 1713 | 0.369 | 225 | 232 |

## *Voorbeeld sketch: 180712VQ_AlphasenseOX_v100_G1_sn1720.ino*

```
/* 180712VQ_AlphasenseOX_v100 - sn 20464 1720  - sens.nr. 1
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean Working Electrode and Aux concentration every 1sec to serial TX
 * based on read input voltages from Alphasense OX sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
 * in GAIN_TWO mode -> ADC range +/- 2.048V 0.0625V/bit
 * Working Electrode OP1 sensor -> ADC A0..A1
 * Aux Electrode OP2 sensor -> ADC A2..A3
 *   -> max accesptable O3 concentration 609ppb minimum
 *   -> for connected sensor 20464 1720: max conc O3 =651ppb, 0.17ppb/bit
 * SensorId fixed in sketch (not with IDE 2..4) as Sensitivity and Electronic zero
 * are sensor dependent and set in sketch
 *
 * 12/07/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// variables to set for every individual connected gassensor
unsigned int uiSensNr = 1;      // fixed in sketch 0..7, replaces selection with IDE 2..4
String sSnSens ="204641720";    // serial number of connected sensor
float fSensitivity =0.369;      // sensor Sensitivity (mV/ppb) from spec sheet
int iElectZeroWE =236;          // electronic zero Working Electrode (mV)
int iElectZeroAux = 232;        // electronic zero Aux (mV)

// variables to set for every connected gassensor type
const char cSensId = 'G';              // Sensor Id : G = Alphasense OX sensor
const float fVadcMultiplier = 0.0625;  // multiplier of ADC, Voltage of 1 bit of ADC (ADC used
in GAIN_TWO)

// global variables, not to be set for individual sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings

unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSumWE;    // sum sensor measurement Working Electrode
unsigned long ulSensMeasSumAux;   // sum sensor measurement Aux electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
                                  // Stdev only calculated for WE
char cCheckSum;
String dataString = "";

void setup(void)
{
  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  Serial.print("180712VQ_AlphasenseOX_v100 sensorNr: ");
  Serial.print(uiSensNr);
  Serial.print("  sn: ");
  Serial.println(sSnSens);
  Serial.print(" electr zero WE: ");
  Serial.print(iElectZeroWE);
  Serial.print("mV  electr zero Aux: ");
  Serial.print(iElectZeroAux);
  Serial.println("mV");
  Serial.print(" sensitivity in spec: ");
  Serial.print(fSensitivity);
  Serial.print("mV/ppb  1-bit= ");
  Serial.print((1/fSensitivity)/(1/fVadcMultiplier));
  Serial.println("ppb");

  Serial.println();
  Serial.println("capturing data stream..");
```

```
  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range to GAIN_TWO: +/- 2.048V (1 bit = 0,0625V) and start reading
  ads1115.setGain(GAIN_TWO);
  ads1115.begin();

  // init variables
  ulSensMeasSumWE =0;
  ulSensMeasSumAux =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  int16_t i16ResultADC23;
  float fSensMeasMeanWE;
  float fSensMeasMeanAux;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdevWE;

  ulActMilli = millis();

  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values, 1bit = 0,0625V
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSumWE +=i16ResultADC01;
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier;

  // read sensor measurement Aux - OP2 connected to A2 - A3
  i16ResultADC23 =ads1115.readADC_Differential_2_3();
  ulSensMeasSumAux +=i16ResultADC23;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg) and convert to concentration
    fSensMeasMeanWE =(float(ulSensMeasSumWE)/uiNrInAvg) *fVadcMultiplier;
    fSensMeasMeanAux =(float(ulSensMeasSumAux)/uiNrInAvg) *fVadcMultiplier;

    // calculate stdev for WE
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMeanWE;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdevWE =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMeanWE);
    dataString += ";";
    dataString += String(fSensMeasMeanAux);
    dataString += ";";
    dataString += String((fSensMeasMeanWE -iElectZeroWE) /fSensitivity);
    dataString += ";";
    dataString += String((fSensMeasMeanAux -iElectZeroAux) /fSensitivity);
    dataString += ";";
    dataString += String(fStdevWE);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);
```

```
    // reset variables for next output
    uiNrInAvg =0;
  ulSensMeasSumWE =0;
  ulSensMeasSumAux =0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

## Citytech O3 3E1F


### General information

Meet: ozon
Aansluiting: 4..20mA
Voeding: afzonderlijke 24 Vdc voeding

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft één meetkanaal via een stroomuitgang (4..20mA). Geeft ook negatieve waardes, waarden kleiner dan 4mA.
Testen en ontwikkeling met Citytech O3 3E1F sensor met serienummer: 13015985-068
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning door ADC in mV, na omzetting van mA naar mV met precisie weerstand
- Concentratie $O_3$ in ppb (omzetting spanning ADC)
- Standaard afwijking van [spanningsmetingen -1000mV] gedurende meetperiode van 1 sec.

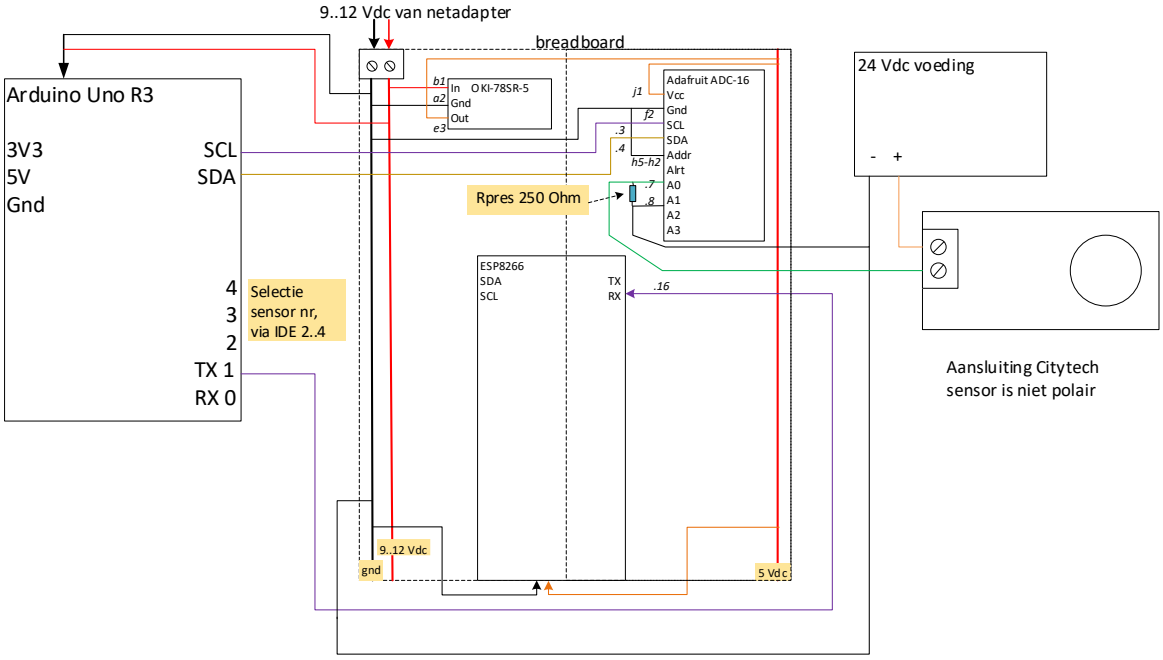
### Instelling

Sensor ID: 'J' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

### Assemblage scheme

**Aansluitschema – instelling sensor**




De sensoren zijn volledig gekalibreerd verscheept. De fabriekskalibratie werd behouden.

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[O3_mV];[O3_ppb];[Stdev_O3_mV];[<ETX>]
;[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Citytech O3-3E1F is dit ‘J’ (0x4A) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[O3_mV]` | gemiddelde meting in mV, na omzetting mA -> mV via Rpres (250 Ohm) |
| `[O3_ppb]` | gemiddelde concentratie $O_3$ in ppb op basis van mV gemiddelde |
| `[Stdev_O3_mV]` | standaard afwijking van `[O3_mV] -1000mV`, de offset wordt tijdelijk afgetrokken om een relevantere Stdev waarde te krijgen |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180817VQ_CitytechO3_v100.ino

```
/* 180817VQ_CitytechO3-3E1F_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean O3 concentration every 1sec to serial TX
 * based on read 4..20mA input current, converted to voltage with precission resistor, from
Citytech O3 3E 1 F sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
 * in GAIN_ONE 0.125mV/bit  -> typical 0.031ppb/bit, max conc 819ppb   to obtain better
resolution,
 * max. conc with higher GAIN would be too low
 * sensor 4..20mA current, measurements can go below 4mA -> 0..5V voltage -> ADC A0..A1
 * sensor output = 21mA is case of error:
```

```
 *   - output < 3.8mA
 *   - output > 20.5mA
 *   - operated > 5Â°C outside temperature range -40..+50Â°C
 * error only supported in GAIN_TWOTHIRD in other GAIN_ settings the error will blow the ADC..
 *
 * 17/08/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// sensor depending variables
unsigned int uiFS =1000;        // sensor full scale (ppb) from spec sheet O3 3E 1 F
float fResistIU =250.0;         // resistor used for current -> voltage (Ohm)

// global variables, not to be set for individual sensor
const char cSensId = 'J';              // Sensor Id : J = Citytech O3 3E 1 F sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings
const float fVadcMultiplier = 0.125;          // multiplier of ADC, mVoltage of 1 bit of ADC
(ADC used in GAIN_ONE)

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSum;      // sum sensor measurement Working Electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
char cCheckSum;
String dataString = "";

void setup(void)
{
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180817VQ_CitytechO3-3E1F_v100 sensorNr: ");
  Serial.println(uiSensNr);

  Serial.print(" ADC ");
  Serial.print(fVadcMultiplier);
  Serial.print(" mV/bit,  ");
  Serial.print(" 1-bit= ");
  Serial.print( fVadcMultiplier*uiFS/((20-4)*fResistIU) );
  Serial.println("ppb");
  Serial.println(" max. O3 conc. with ADC settings= 819ppb");

  Serial.println();
  Serial.println("capturing data stream..");

  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range and start reading
  //  GAIN_TWOTHIRDS +/- 6.144V   1 bit = 0,1875mV = fVadcMultiplier
  //  GAIN_ONE       +/- 4.096V           0.125mV
  //  GAIN_TWO       +/- 2.048V           0.0625mV
  //  GAIN_FOUR      +/- 1.024V           0.03125mV
  //  GAIN_EIGHT     +/- 0.512V           0.015625mV
  //  GAIN_SIXTEEN   +/- 0.256V           0.0078125mV
  //
  ads1115.setGain(GAIN_ONE);
  ads1115.begin();
```

```
  // init variables
  ulSensMeasSum =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  float fSensMeasMean;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdev;
  float fSensConcPPB;

  ulActMilli = millis();

  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSum +=i16ResultADC01;
  // substract offset 4mA: -1000.0mV for better Stdev calculation
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier -1000.0;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg), substract offest 4mA: -1000.0mV for better Stdev calculation
    fSensMeasMean =(float(ulSensMeasSum)/uiNrInAvg) *fVadcMultiplier -1000.0;

    // calculate stdev for raw mV measures
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMean;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdev =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // convert raw mA to ppb:  fResist (mV -> mA), scaling to FS = 20mA, offset = 4mA -> range
16mA
    fSensConcPPB =(fSensMeasMean/fResistIU) *(uiFS/16);
    // add offest (1000.mV) back to mean for string output
    fSensMeasMean =fSensMeasMean +1000.0;

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMean);
    dataString += ";";
    dataString += String(fSensConcPPB);
    dataString += ";";
    dataString += String(fStdev);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulSensMeasSum =0;

    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}
```

```
// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

## Envea Cairclip O3/NO2

This is a plug & play sensor

Aeroqual SM50

## General information

Meet: ozon
Aansluiting: RS232
Voeding: afzonderlijke 24 Vdc voeding (11..30 Vdc)

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor meet ozon en zet circa elke 70 seconde een binair datarecord op de RS232 poort.
Testen en ontwikkeling uitgevoerd op sensor met serienummer 1712804-201.
Fabrieks-, kalibratie-instellingen niet aangepast.

De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Ozon concentratie in ppb
- Status sensor

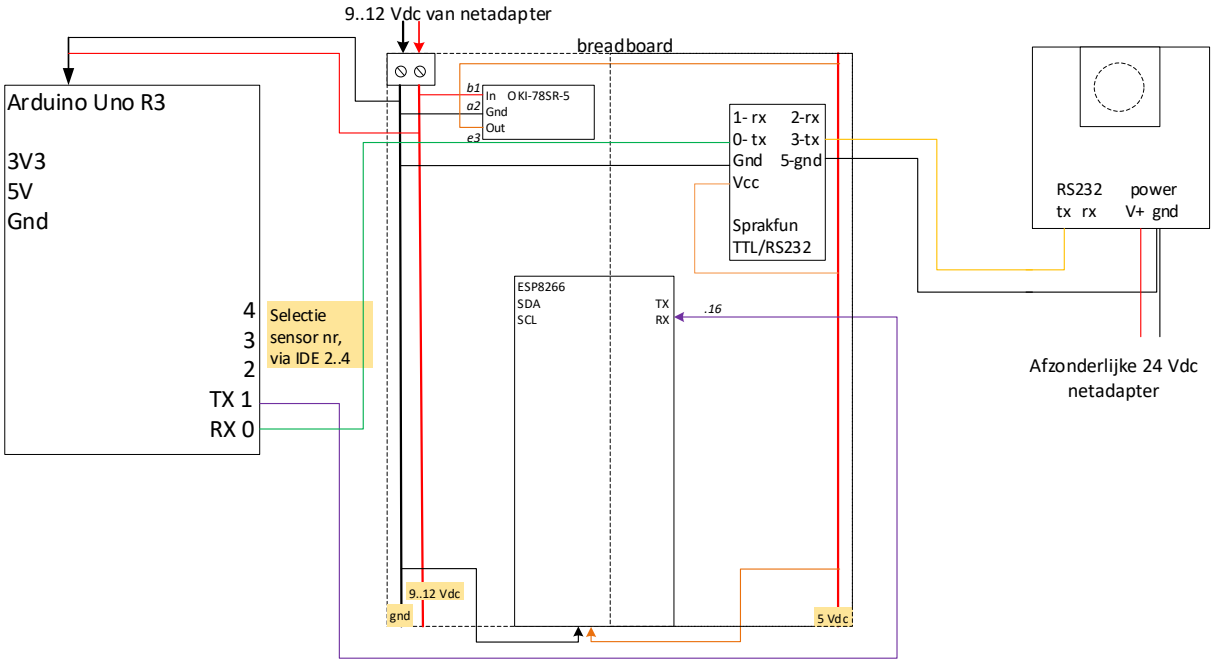
### Instelling

Sensor ID: 'O' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

## Assemblage scheme

### Aansluitschema – instelling sensor




## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[O3];[status];[<ETX>];[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Aeroqual O3 0-0.15 is dit ‘O’ (0x4F) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen, daar de Aeroqual circa elke 70 seconde een meting geeft zijn er meer berichten met AantalInGem=0 en af en toe eentje met echte meting |
| `[O3]` | (gemiddelde) meting ozon in ppb, formaat xx.xx |
| `[status]` | status zoals ontvangen van sensor<br>0 sensor ok<br>1 sensor failure -> sensor defect<br>2 sensor aging -> veroudering sensor |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

**Sketch: 180912VQ_AeroqualO3ul_v100.ino**

```
/* 180912VQ_AeroqualO3ul_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * Outputs O3 concentration every 1sec to serial, from a Aeroqual O3 ultra low sensor.
 * Aeroqual gives a data record every 70 seconds
 *
 * CONNECTING
 * Tx channel on sensor is connected to RS232 convertor pin 3 (TX)
 * TTL-TX RS232 convertor is connected to Arduino RX
```

```
 * Rx channel on sensor is not connected
 *
 * 12/09/2018 VMM - Jan Adams
 */

const char cSensId = 'O';       // Sensor Id : O = Aeroqual O3 ultra low 0-0.15ppm
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
int iStatus =0;
float fO3ConcSum =0.0;
byte bReceive[20];
char cCheckSum;
String dataString = "";
union {
  byte bConc[4];
  float fConc;
} btof;

int iTelC =0;

void setup(){
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180912VQ_AeroqualO3 Ultra Low_v100 sensorNr: ");
  Serial.println(uiSensNr);
  Serial.println();
  Serial.println("capturing data stream..");

  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
  iStatus =0;
}

void loop(){
  byte bRec;
  byte bChecksum =0;
  int iPnt =0;

  ulActMilli = millis();

  // read sensor
  if (Serial.available()) {
    bRec = Serial.read();
    delay( 2);

    // check if received byte is header1-0xAA
    if (bRec == 0xAA) {
      bReceive[iPnt++] =bRec;
      bRec = Serial.read();
      delay( 2);

      // check if received byte is header2-0x10
      if (bRec =0x10) {
        bReceive[iPnt] =bRec;
        for( iPnt =2; iPnt <15; iPnt++) {
          bReceive[iPnt] =Serial.read();
          if (iPnt <6) {      // store conc in union.byte
            btof.bConc[iPnt-2] =bReceive[iPnt];
```

```
          }
          delay( 2);
        }

        // check status
        if ((bReceive[12] & 3) >0) {
          if (bReceive[12] & 1) iStatus =1;   // sensor failure
          if (bReceive[12] & 2) iStatus =2;   // sensor aging
        }

        // checksum in data record is not the same as calculated -> no checksum verification

        // ppm -> ppb, add for average value
        uiNrInAvg++;
        fO3ConcSum += (float) btof.fConc *1000.0;
      }
    }
  }

  // output
  if (ulActMilli > ulSampleMilli) {
    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fO3ConcSum /uiNrInAvg);
    dataString += ";";
    dataString += String(iStatus);;
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
    fO3ConcSum =0.0;
    iStatus =0;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

VAQUUMS

## Membrapor O3/C-5


### General information

Meet: ozon
Aansluiting: 4..20mA
Voeding: afzonderlijke 24 Vdc voeding

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft één meetkanaal via een stroomuitgang (4..20mA).
Testen en ontwikkeling met Membrapor O3-C5 sensor met serienummer: 18083292.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Gemeten spanning door ADC in mV, na omzetting van mA naar mV met precisie weerstand
- Concentratie $O_3$ in ppb (omzetting spanning ADC)
- Standaard afwijking van [spanningsmetingen -1000mV] gedurende meetperiode van 1 sec.

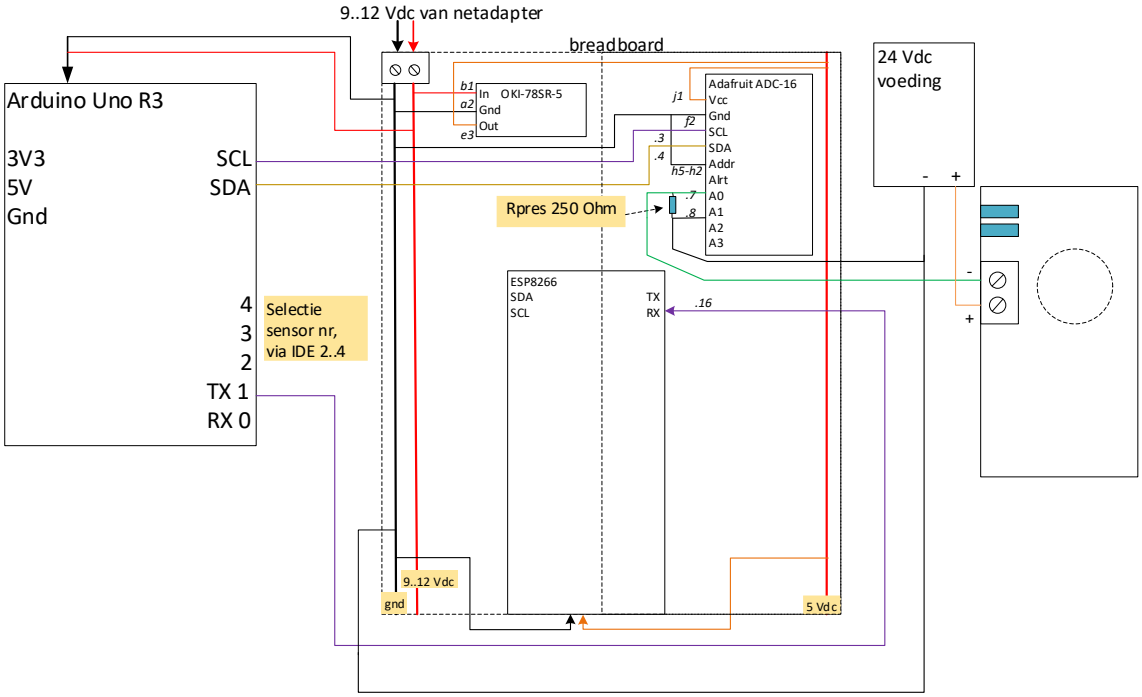
### Instelling

Sensor ID: 'M' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

### Assemblage scheme

**Aansluitschema – instelling sensor**




De fabriekskalibratie werd behouden.

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[O3_mV];[O3_ppb];[Stdev_O3_mV];[<ETX>]
;[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Membrapor O3-C5 is dit ‘M’ (0x4D) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[O3_mV]` | gemiddelde meting in mV, na omzetting mA -> mV via Rpres (250 Ohm) |
| `[O3_ppb]` | gemiddelde concentratie $O_3$ in ppb op basis van mV gemiddelde |
| `[Stdev_O3_mV]` | standaard afwijking van `[O3_mV] -1000mV`, de offset wordt tijdelijk afgetrokken om een relevantere Stdev waarde te krijgen |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180821VQ_MembraporO3-C5_v100.ino

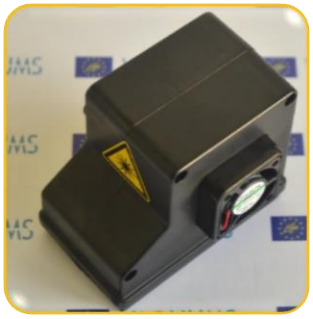
```
/* 180821VQ_MembraporO3-C5_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * output string with mean O3 concentration every 1sec to serial TX
 * based on read 4..20mA input current, converted to voltage with precision resistor, from
Membrapor O3 C5 sensor
 *
 * uses adafruit ADC-16bit in differential mode for best analog to digital conversion
 * in GAIN_ONE 0.125mV/bit  -> typical 0.156ppb/bit, max conc 4080ppb   to obtain better
resolution
```

```
 * at startup the sensor output peaks above 2.1V, a higher GAIN could damage the ADC at
startup
 * sensor 4..20mA current, measurements can go below 4mA -> 0..5V voltage -> ADC A0..A1
 *
 * 21/08/2018 VMM - Jan Adams
 */

#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads1115;

// sensor depending variables
unsigned int uiFS =5000;        // sensor full scale (ppb) from spec sheet O3 C5
float fResistIU =250.0;         // resistor used for current -> voltage (Ohm)

// global variables, not to be set for individual sensor
const char cSensId = 'M';              // Sensor Id : M = Membrapor O3 C5 sensor
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s
const unsigned long ulGasSendInterv = 48;     // interval (ms) between 2 sensor readings
const float fVadcMultiplier = 0.125;          // multiplier of ADC, mVoltage of 1 bit of ADC
(ADC used in GAIN_ONE)

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulSensMeasSum;      // sum sensor measurement Working Electrode
unsigned long ulActMilli;
unsigned long ulSampleMilli;
float fSensMeas[22];              // array for calculation of Stdev, size: ulSampleInterv /
ulGasSendInterv + 2 spare
char cCheckSum;
String dataString = "";

void setup(void)
{
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180821VQ_MembraporO3-C5_v100 sensorNr: ");
  Serial.println(uiSensNr);

  Serial.print(" ADC ");
  Serial.print(fVadcMultiplier);
  Serial.print(" mV/bit,  ");
  Serial.print(" 1-bit= ");
  Serial.print( fVadcMultiplier*uiFS/((20-4)*fResistIU) );
  Serial.println("ppb");
  Serial.println(" max. O3 conc. with ADC settings= 4080ppb");

  Serial.println();
  Serial.println("capturing data stream..");

  // adaFruit 16-bit ADC, getting differential reading from AIN0 (Pos) and AIN1 (Neg - Gnd)
  // set ADC Range and start reading
  //  GAIN_TWOTHIRDS +/- 6.144V   1 bit = 0,1875mV = fVadcMultiplier
  //  GAIN_ONE       +/- 4.096V           0.125mV   <---
  //  GAIN_TWO       +/- 2.048V           0.0625mV
  //  GAIN_FOUR      +/- 1.024V           0.03125mV
  //  GAIN_EIGHT     +/- 0.512V           0.015625mV
  //  GAIN_SIXTEEN   +/- 0.256V           0.0078125mV
  //
  ads1115.setGain(GAIN_ONE);
  ads1115.begin();
```

```
  // init variables
  ulSensMeasSum =0;
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}


void loop(void)
{
  int iPnt;
  int16_t i16ResultADC01;
  float fSensMeasMean;
  float fDiffMeasMean;
  float fSumRtDiffMeasMean =0;
  float fStdev;
  float fSensConcPPB;

  ulActMilli = millis();

  // read sensor measurement WE - OP1 connected to A0 - A1, output of ADC 16bit -> 65536
possible values
  i16ResultADC01 =ads1115.readADC_Differential_0_1();
  ulSensMeasSum +=i16ResultADC01;
  // substract offset 4mA: -1000.0mV for better Stdev calculation
  fSensMeas[uiNrInAvg++] =float(i16ResultADC01) *fVadcMultiplier -1000.0;

  if (ulActMilli < ulSampleMilli) {
    if ((ulActMilli +ulGasSendInterv) >ulSampleMilli) {
      delay(abs(ulSampleMilli-ulActMilli));
    } else {
      delay(ulGasSendInterv);     // wait x msec for next reading
    }
  } else {
    // output String
    // calculate mean (avg), substract offest 4mA: -1000.0mV for better Stdev calculation
    fSensMeasMean =(float(ulSensMeasSum)/uiNrInAvg) *fVadcMultiplier -1000.0;

    // calculate stdev for raw mV measures
    for (iPnt =0; iPnt <uiNrInAvg; iPnt++) {
      fDiffMeasMean =fSensMeas[iPnt] -fSensMeasMean;
      fSumRtDiffMeasMean +=fDiffMeasMean *fDiffMeasMean;
    }
    fStdev =sqrt(fSumRtDiffMeasMean /(uiNrInAvg-1));

    // convert raw mA to ppb:  fResist (mV -> mA), scaling to FS = 20mA, offset = 4mA -> range
16mA
    fSensConcPPB =(fSensMeasMean/fResistIU) *(uiFS/16);
    // add offest (1000.mV) back to mean for string output
    fSensMeasMean =fSensMeasMean +1000.0;

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(fSensMeasMean);
    dataString += ";";
    dataString += String(fSensConcPPB);
    dataString += ";";
    dataString += String(fStdev);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulSensMeasSum =0;

    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}
```

```
// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

## Alphasense OPC-N2



Programmed according to: https://github.com/dhhagan/opcn2

Dylos DC1700


## General information

Meet: PM2.5, PM10
Aansluiting: RS232
Voeding: afzonderlijke 9 Vdc voeding

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor meet PM2.5 en PM10 en zet elke 1 minuut een string op de RS232 poort.
Testen en ontwikkeling met Dylos 1700 v2.09m. De Dylos sensoren hebben geen afzonderlijk serienummer.
De standaard bijgeleverde 9V voeding is met een Amerikaanse wandaansluiting voor 120Vac 60Hz en niet bruikbaar. Met de bestelde 9Vdc voeding voor de Arduino lukt het wel.
De Dylos instelling werden niet gewijzigd en zijn:

- PM correction factor =1.00
- Continuos mode PM2.5 PM10

De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- PM2.5 concentratie in µg/$m^3$
- PM10 concentratie in µg/$m^3$

### Instelling

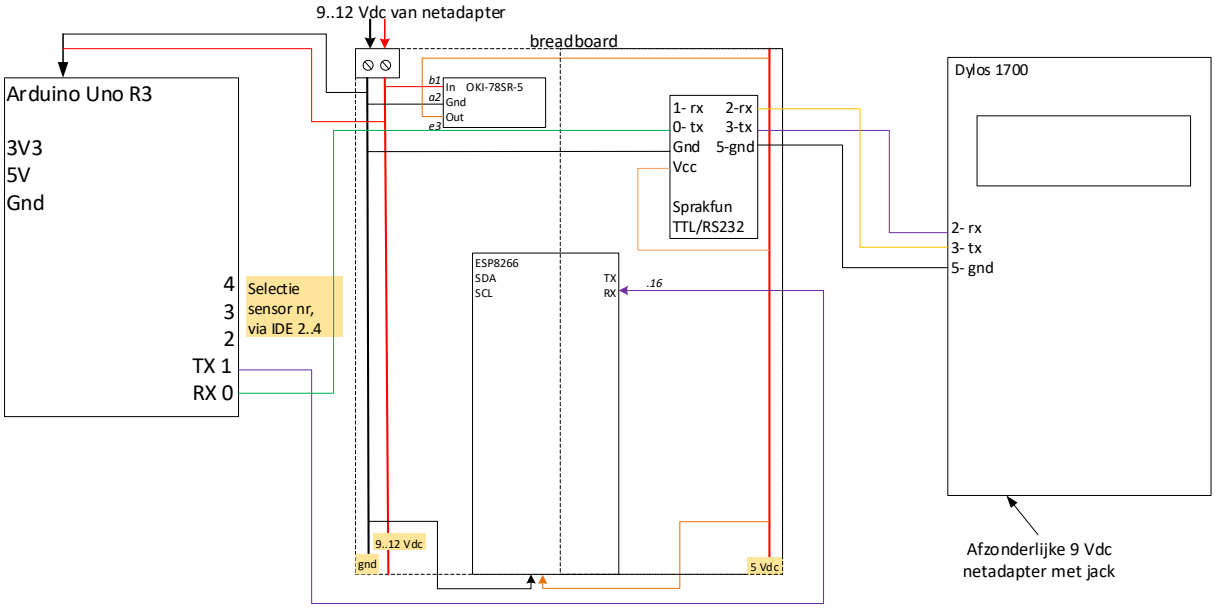
Sensor ID: 'N' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

-

## Assemblage scheme

**Aansluitschema – instelling sensor**




## Data output

### Structuur

`<STX>VQ[SensorID][SensorNr];[AantalInGem];[PM2.5];[PM10];[<ETX>];[CheckSum]`

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Dylos 1700 is dit 'N' (0x4E) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ';' (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen, daar de Dylos elke 1 minuut een meting geeft zijn er 59 berichten met AantalInGem=0 en eentje met echte metingen |
| `[PM2.5]` | (gemiddelde) meting PM2.5 in µg/m$^3$, formaat xxx.xx |
| `[PM10]` | (gemiddelde) meting PM10 in µg/m$^3$, formaat xxx.xx |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180829VQ_Dylos_v100.ino

```
/* 180829_Dylos_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * Outputs mass concentration (Âµg/m3) every 1sec to serial, from a Dylos 1700 sensor.
 *
 * CONNECTING
 * Tx channel on sensor is connected to digital 0 (Rx)
 * Rx channel on sensor is only connected to digital 1 (Tx) on startup
 * afterwards when measuring the Rx is not connected
 *
 * 24/08/2018 VMM - Jan Adams
 *
 */

const char cSensId = 'N';       // Sensor Id : N = Dylos 1700
const char STX = 2;
const char ETX = 3;
```

```
const char LF = 10;
const char CR = 13;
const char ASCII_0 = 48;                        // ASCII value of "0"
const char ASCII_PNT = 46;                      // ASCII value of "."
const char SEPARATOR = 44;                      // values in stream received from Dylos are
comma "," separated
const unsigned long ulSampleInterv = 1000;     // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned int uiPM10;
unsigned int uiPM25;
unsigned int uiNrInAvg = 0;
unsigned long ulPM10sum;
unsigned long ulPM25sum;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
int iPnt = 0;
byte bReceive[20];
char cCheckSum;
String dataString = "";

void setup(){
  bool bStreaming = false;
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180829VQ_Dylos1700_v100 sensorNr: ");
  Serial.println(uiSensNr);
  Serial.println();
  Serial.println("capturing data stream..");

  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
  uiPM10 = 0;
  uiPM25 = 0;
  ulPM10sum =0;
  ulPM25sum =0;
  iPnt =0;
}

void loop(){
  byte bRec;
  int iDecPower = 1;

  ulActMilli = millis();

  // read sensor
  if (Serial.available()) {
    bRec =Serial.read();
    bReceive[iPnt++] = bRec;

    if (bRec ==LF) {      // if LF received, process received characters in reverse order
      if (iPnt >=5) {
        iPnt -=3;   // skip CR LF

        // start with PM10 until separator, format xxx.x
        while (bReceive[iPnt] !=SEPARATOR) {
          if (bReceive[iPnt] != ASCII_PNT) {
            uiPM10 += (bReceive[iPnt] -ASCII_0) *iDecPower;
            iDecPower *= 10;
          }
          iPnt--;
        }
        iPnt--;   //skip separator
```

```
        // continue with PM2.5, format xx.x
        iDecPower = 1;
        while (iPnt >=0) {
          if (bReceive[iPnt] != ASCII_PNT) {
            uiPM25 +=(bReceive[iPnt] -ASCII_0) *iDecPower;
            iDecPower *= 10;
          }
          iPnt--;
        }
        iPnt = 0;

        // add for average value
        uiNrInAvg++;
        ulPM10sum += uiPM10;
        ulPM25sum += uiPM25;
      }
    }
  }

  // output
  if (ulActMilli > ulSampleMilli) {
    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(float(ulPM25sum)/(uiNrInAvg *10));
    dataString += ";";
    dataString += String(float(ulPM10sum)/(uiNrInAvg *10));
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg = 0;
    uiPM10 = 0;
    uiPM25 = 0;
    ulPM10sum = 0;
    ulPM25sum = 0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

Honeywell HPMA115S0

## General information

Meet: PM2.5 en PM10
Aansluiting: RS232 TTL op 9600 baud, 8 bits, geen pariteit, 1 stopbit
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).

De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

Sensor ID
Sensor nummer
Aantal in gemiddelde
Concentratie PM2.5 (µg/m3)
Concentratie PM10 (µg/m3)

### Instelling

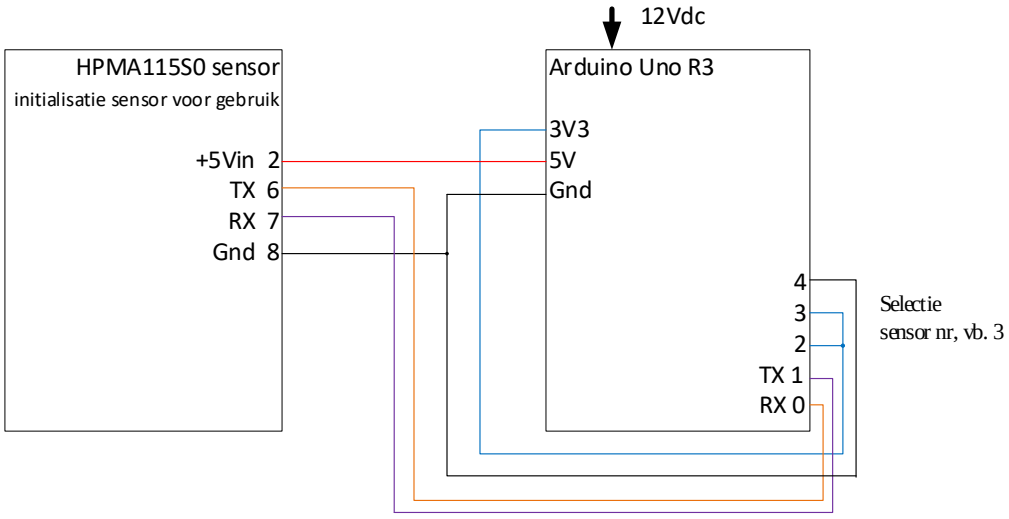
Sensor ID: 'D' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

## Assemblage scheme

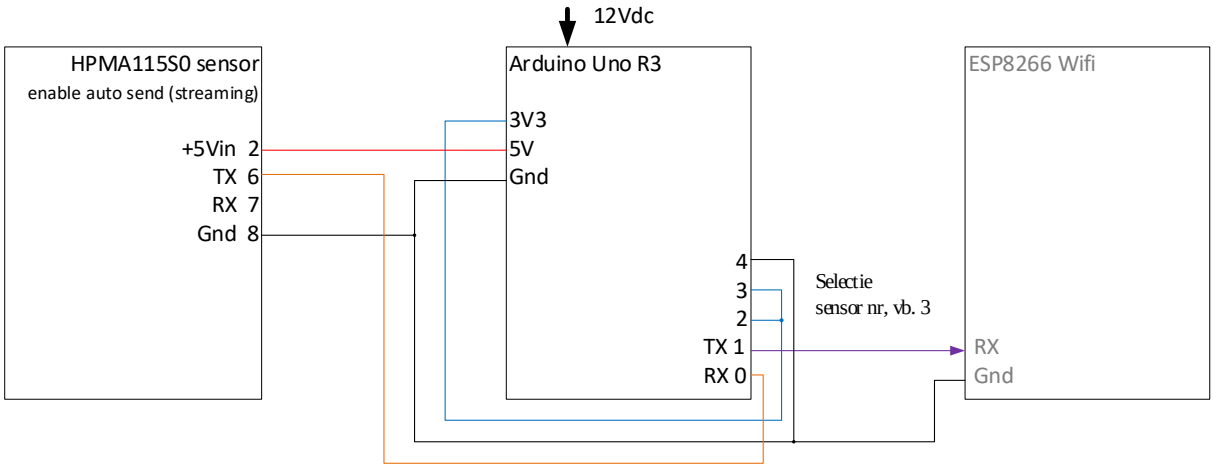
### Aansluitschema

***Aansluitschema initialisatie sensor voor gebruik***



VAQUUMS

***Aansluitschema sensor bij meting in streaming mode***




**Instelling sensor**

De Customer adjustment coefficient van alle HPMA115S0 sensoren werd uitgelezen. Voor alle sensoren staat dit op de default waarde van 100.

## Data output

### Structuur

```
STX>VQ[SensorID][SensorNr];[AantalInGem];[PM2.5];[PM10]<ETX>[CheckSum]
```

| | |
|---|---|
| <STX> | vast, Start of TeXt, 0x02 |
| VQ | vast, hoofding VaQuums |
| [SensorID] | sensor ID, voor HPMA is dit ‘D’ (0x44) |
| [SensorNr] | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| ; | vast scheidingskarakter ‘;’ (0x3b) |
| [AantalInGem] | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, in deze versie van het sketch is dit steeds 1 daar de Arduino elke poll doorgeeft en geen gemiddelde berekent |
| [PM2.5] | PM2.5 meting van sensor, ‘.’ (0x2e) als decimaalteken |
| [PM10] | PM10 meting van sensor |
| <ETX> | vast, End of TeXt, 0x03 |
| [CheckSum] | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

**Sketch: 180620VQ_HPMA_v101.ino**

```
/* 180620VQ_HPMA_v101
 * VAQUUMS project
 *
 * DESCRIPTION
 * Outputs mass concentration (Âµg/m3) every 1sec to serial, from a Honeywell HPMA115S0
sensor.
 *
 * CONNECTING
 * Tx channel on sensor is connected to digital 0 (Rx)
 * Rx channel on sensor is only connected to digital 1 (Tx) on startup
 * afterwards when measuring the Rx is not connected
 *
 * 20/06/2018 VMM - Jan Adams
 * v101  improved reading of stream from sensor, increase rx-time/out when receiving data
 *
 */

const char cSensId = 'D';       // Sensor Id : D = HPMA115S0
```

```
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned int uiPM10;
unsigned int uiPM25;
unsigned int uiNrInAvg = 0;
unsigned long ulPM10sum;
unsigned long ulPM25sum;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
char cCheckSum;
String dataString = "";

void setup(){
  bool bStreaming = false;
  int iPow2 = 1;
  int iPin;
  int iAdjCoeff;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180620VQ_HPMA115S0_v101 sensorNr: ");
  Serial.println(uiSensNr);
  Serial.println();

  // Read and check Customer Adjust Coeff
  iAdjCoeff = HPMA_ReadCustomerAdjustCoeff();
  switch (iAdjCoeff) {
    case 0:
      Serial.println(" time-out, no answer from sensor, check connection");
      break;
    case 1:
      Serial.println(" answer from sensor not as expected, check connection/reconnect");
      break;
    default:
       Serial.print(" customer adjustment coeff value for the connected sensor is: ");
       Serial.print(iAdjCoeff);
       if (iAdjCoeff ==100) {
        Serial.println("  -> DEFAULT");
       } else {
        Serial.println("  -> coeff is NON default");
       }
       delay(200);
  }

  if (iAdjCoeff >29) {
    // Enable Auto Send (streaming) of sensor
    if (HPMA_EnableAutoSend() ==3) {
      Serial.println(" auto send (streaming) enabled");
      delay(200);
      HPMA_StartMeasurement();    // ==3 ACK, but returns ==2 NACK ??
        Serial.println(" start measurement");
        bStreaming = true;

    }
  }

  if (bStreaming) {
    Serial.println(" sensor initialised");

  } else {
    Serial.println();
    Serial.println(" initialisation failed, fix & retry");
  }
  Serial.println(" disconnect Sensor_RX from Arduino_TX (disturbs HPMA measurements), connect
Arduino_TX to ESP8266_RX");
```

```
  delay(10000);       // wait 10s before starting to read auto send stream
  Serial.println("capturing data stream..");

  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
  ulPM10sum =0;
  ulPM25sum =0;
}

void loop(){
  ulActMilli = millis();

  // read sensor
  if (HPMA_ReadStream() ==2) {      // check for HPMA streaming data
    uiNrInAvg++;
    ulPM10sum = ulPM10sum +uiPM10;
    ulPM25sum = ulPM25sum +uiPM25;
  }

  // output
  if (ulActMilli > ulSampleMilli) {
    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(float(ulPM25sum)/uiNrInAvg);
    dataString += ";";
    dataString += String(float(ulPM10sum)/uiNrInAvg);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg = 0;
    ulPM10sum =0;
    ulPM25sum =0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}


// Read Customer Adjust Coeff from HPMA sensor connected to RX/TX pins
// receive time-out of 100ms
// returns int 0 - time-out, nothing received
//             1 - received header not as expected
//            30..200 - customer adjust coeff of connected sensor
int HPMA_ReadCustomerAdjustCoeff() {
  byte bToSend[] ={0x68, 0x01, 0x10, 0x87};
  byte bReceive[5];
  int iPnt =0;
  int iReturn =0;
  unsigned long ulRxTO;
  ulRxTO =millis() +250;      // rx time-out 250msec

  Serial.write(bToSend, sizeof(bToSend));
  while (millis() <ulRxTO && iPnt <5) {
```

```
    if (Serial.available()) {
      bReceive[iPnt++] =Serial.read();
    }
  }

  // if bytes received check content
  if (iPnt >0) {
    if ((bReceive[0] ==0x40) && (bReceive[1] == 0x02) && (bReceive[2] == 0x10)) {
      iReturn =int(bReceive[3]);
    } else {
      // received header not as expected
      iReturn =1;
    }
  }
  return iReturn;
}


// Enable Auto Send
// returns int 0 - time-out, nothing received
//             1 - not as expectd
//             2 - NACK
//             3 - ACK
int HPMA_EnableAutoSend() {
  byte bToSend[] ={0x68, 0x01, 0x40, 0x57};
  byte bReceive[5];
  int iPnt =0;
  int iReturn =0;
  unsigned long ulRxTO;
  ulRxTO =millis() +200;      // rx time-out 200msec

  Serial.write(bToSend, sizeof(bToSend));
  while (millis() <ulRxTO && iPnt <2) {
    if (Serial.available()) {
      bReceive[iPnt++] =Serial.read();
    }
  }

  // if bytes received check content
  if (iPnt >0) {
    if ((bReceive[0] ==0xA5) && (bReceive[1] == 0xA5)) {
      // NACK
      iReturn =2;
    } else if ((bReceive[0] ==0x96) && (bReceive[1] ==0x96)) {
      // ACK
      iReturn =3;
    } else {
      iReturn =1;
    }
  }
  return iReturn;
}


// Start measurement
// returns int 0 - time-out, nothing received
//             1 - not as expectd
//             2 - NACK
//             3 - ACK
// note: protocol logging shows that sensor always returns NACK although it starts measuring
int HPMA_StartMeasurement() {
  byte bToSend[] ={0x68, 0x01, 0x01, 0x96};
  byte bReceive[5];
  int iPnt =0;
  int iReturn =0;
  unsigned long ulRxTO;
  ulRxTO =millis() +200;      // rx time-out 200msec

  Serial.write(bToSend, sizeof(bToSend));
  while (millis() <ulRxTO && iPnt <2) {
    if (Serial.available()) {
      bReceive[iPnt++] =Serial.read();
    }
  }

  // if bytes received check content
  if (iPnt >0) {
```

```
    if ((bReceive[0] ==0xA5) && (bReceive[1] == 0xA5)) {
      // NACK
      iReturn =2;
    } else if ((bReceive[0] ==0x96)&& (bReceive[1] ==0x96)) {
      // ACK
      iReturn =3;
    } else {
      iReturn =1;
    }
  }
  return iReturn;
}


// Check for HPMA streaming and read data
// time-out of 200msec
// returns int: 0 - no data, time-out
//              1 - checksum failure
//              2 - data, PM2.5 and PM10 into variables uiPM25 and uiPM10
int HPMA_ReadStream() {
  byte bRecSer =false;
  byte bReceive[33];
  int iPnt =0;
  int iReturn =0;
  unsigned int uiChecksum =0;
  unsigned long ulRxTO;
  ulRxTO =millis() +251;      // rx time-out 251msec = (1000/4)+1

  while (millis() <ulRxTO && (millis() <ulSampleMilli || bRecSer)  && iPnt <32) {
    if (Serial.available()) {
      bReceive[iPnt++] =Serial.read();
      ++ulRxTO;                          // increase Rx time/out with 2msec so complete message
will be received
      ++ulRxTO;
      bRecSer =true;
    }
  }

  // if bytes received check content
  if (iPnt >0) {
    if ((bReceive[0] ==0x42) && (bReceive[1] == 0x4D) && (bReceive[2] == 0x00) && (bReceive[3]
== 0x1C)) {
      uiPM25 =int(bReceive[6])*256 +int(bReceive[7]);
      uiPM10 =int(bReceive[5])*256 +int(bReceive[9]);

      // calculate checksum of received data
      for (iPnt =0; iPnt <30; iPnt++) {
        uiChecksum = uiChecksum +int(bReceive[iPnt]);
      }
      if (uiChecksum ==(int(bReceive[30])*256+int(bReceive[31])) ) {
        // correct CS
        iReturn =2;
      } else {
        // CS failure
        Serial.println("Checksum failure");
        iReturn =1;
      }
    }
  }
  return iReturn;
}
```

VAQUUMS

Nova fitness SDS011

## General information

Meet: PM
Aansluiting: RS232 TTL op 9600 baud, 8 bits, geen pariteit, 1 stopbit
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Concentratie PM2.5
- Concentratie PM10

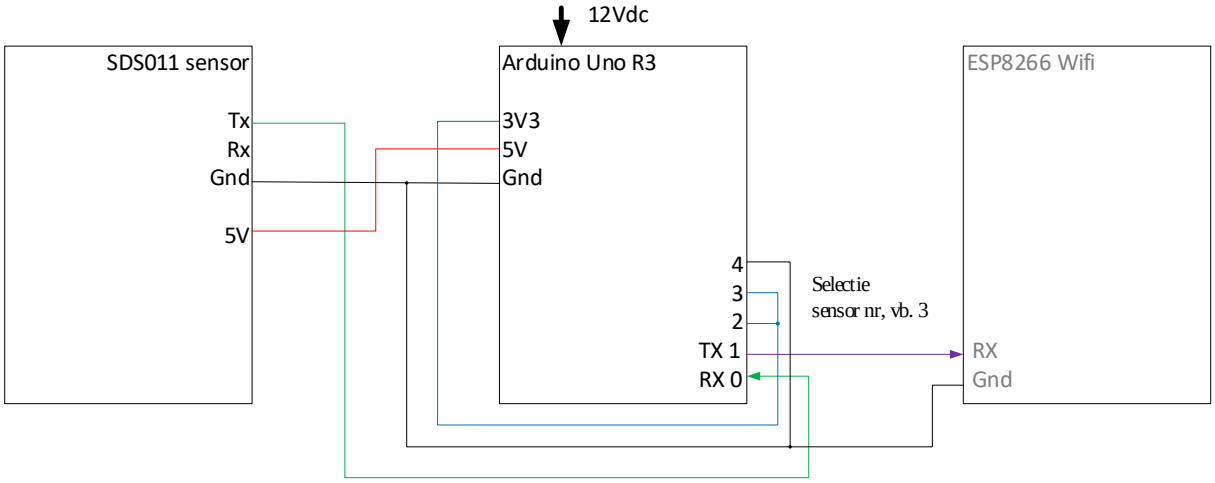
### Instelling

Sensor ID: 'A' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

## Assemblage scheme

### Aansluitschema




## Data output

### Structuur

<STX>VQ[SensorID][SensorNr];[AantalInGem];[PM2.5];[PM10]<ETX>[CheckSum]

| | |
|---|---|
| <STX> | vast, Start of TeXt, 0x02 |
| VQ | vast, hoofding VaQuums |
| [SensorID] | sensor ID, voor SDS011 is dit ‘A’ (0x41) |
| [SensorNr] | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| ; | vast scheidingskarakter ‘;’ (0x3b) |
| [AantalInGem] | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, in deze versie van het sketch is dit steeds 1 daar de Arduino elke poll doorgeeft en geen gemiddelde berekent |
| [PM2.5] | PM2.5 meting van sensor, ‘.’ (0x2e) als decimaalteken |
| [PM10] | PM10 meting van sensor |
| <ETX> | vast, End of TeXt, 0x03 |
| [CheckSum] | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

**Sketch: 180531VQ_SDS011_v100.ino**

```
/* 180530VQ_SDS011_V100
 * VAQUUMS project,
 * read PM values every 1s from SDS011 sensor and put them on the TX of teh serial port
 *
 * read sensor SDS011 measurements with function ProcessSerialSDS011()
 * code of ProcessSerialSDS011() based on www.inovafitness.com
 * and www.dfrobot.com
 *
 * 30/05/2018 VMM - Jan Adams
 */

#include <Wire.h>
const char cSensId = 'A';                       // Sensor Id : A = SDS011
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;        // interval (ms) of sampling sensor on RX
pin 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 1;
unsigned int uiPM25;
unsigned int uiPM10;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
char cCheckSum;
String dataString = "";


void setup() {
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);  // RX0,TX0: 9600,8,N,1
  uiPM25 =0;
  uiPM10 =0;

  // read Arduino pin 2..4, use as Sensor Number
  //  4  3  2   SensorNr
  // -------------------
  //  0  0  0     0
  //  0  0  1     1
  //  0  1  0     2
  //  ..
  //  1  1  1     7
  // converting output of function pow() to int introduces error
  for (iPin=2; iPin<5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  // display header, version, sensor number
  Serial.print("180530VQ_SDS011_v100 sensorNr: ");
```

```
  Serial.print(uiSensNr);

  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}

void loop() {
  ulActMilli = millis();

  if (ulActMilli > ulSampleMilli) {
    // read sensor
    ProcessSerialSDS011();

    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(float(uiPM25)/10.0);
    dataString += ";";
    dataString += String(float(uiPM10)/10.0);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    ulSampleMilli =ulActMilli +ulSampleInterv;
  }
}


//***  functions ***

// code of ProcessSerialSDS011() based on www.inovafitness.com and www.dfrobot.com
void ProcessSerialSDS011() {
  uint8_t mData=0;
  uint8_t i=0;
  uint8_t mPkt[10]={0};
  uint8_t mCheck=0;
  while(Serial.available()) {
    // packet format: AA C0 PM25_low PM25_high PM10_low PM10_high 00 00 CheckSum AB
    //
    mData = Serial.read();
    // wait (2 ms) until data packet received
    delay(1);
    if (mData==0xAA) {    // header 1
      mPkt[0] = mData;
      mData = Serial.read();
      if (mData==0xC0) {    // header 2
        mPkt[1] = mData;
        mCheck = 0;
        for (i=2; i<8; i++) {
          mPkt[i]=Serial.read();
          delay(2);
          mCheck += mPkt[i];
        }
        mPkt[8] = Serial.read();
        delay(1);
        mPkt[9] =Serial.read();
        if (mCheck==mPkt[8]) {    // crc ok ?
          Serial.flush();
          uiPM25 = (uint16_t)mPkt[2]|(uint16_t)(mPkt[3]<<8);
          uiPM10 = (uint16_t)mPkt[4]|(uint16_t)(mPkt[5]<<8);

          return;
        }
      }
    }
  }
}

// calculate checksum with XOR of passed String until end '\0' is found
```

```
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor=0;
  int i=0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

VAQUUMS

Plantower PMS 7003


## General information

Meet: PM1, PM2.5, PM10 en deeltjes aantallen in 6 klassen
Aansluiting: RS232 TTL op 9600 baud, 8 bits, geen pariteit, 1 stopbit
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft zelf maar elke 2 seconde een datarecord. Het aantal in gemiddelde in de datarecords vanuit Arduino waarvoor er geen sensor metingen zijn is 0.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Concentratie PM1 (µg/m3) (CF=1, standard particle)
- Concentratie PM2.5 (µg/m3) (CF=1, standard particle)
- Concentratie PM10 (µg/m3) (CF=1, standard particle)
- Concentratie PM1 (µg/m3) (atmospheric environment)
- Concentratie PM2.5 (µg/m3) (atmospheric environment)
- Concentratie PM10 (µg/m3) (atmospheric environment)
- Aantal deeltjes met diameter >0,3 µm in 0,1 liter lucht
- Aantal deeltjes met diameter >0,5 µm in 0,1 liter lucht
- Aantal deeltjes met diameter >1,0 µm in 0,1 liter lucht
- Aantal deeltjes met diameter >2,5 µm in 0,1 liter lucht
- Aantal deeltjes met diameter >5,0 µm in 0,1 liter lucht
- Aantal deeltjes met diameter >10 µm in 0,1 liter lucht

### Instelling

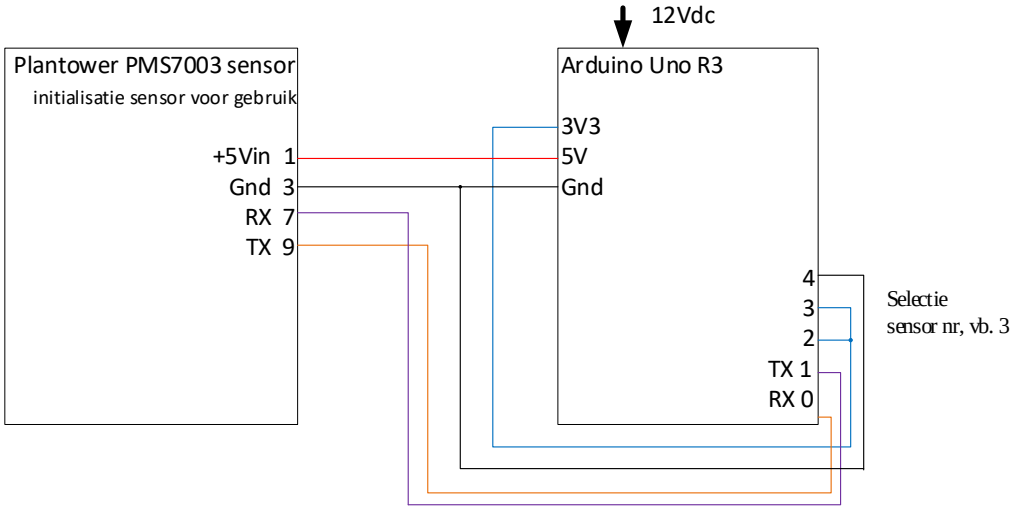
Sensor ID: ‘E’ geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

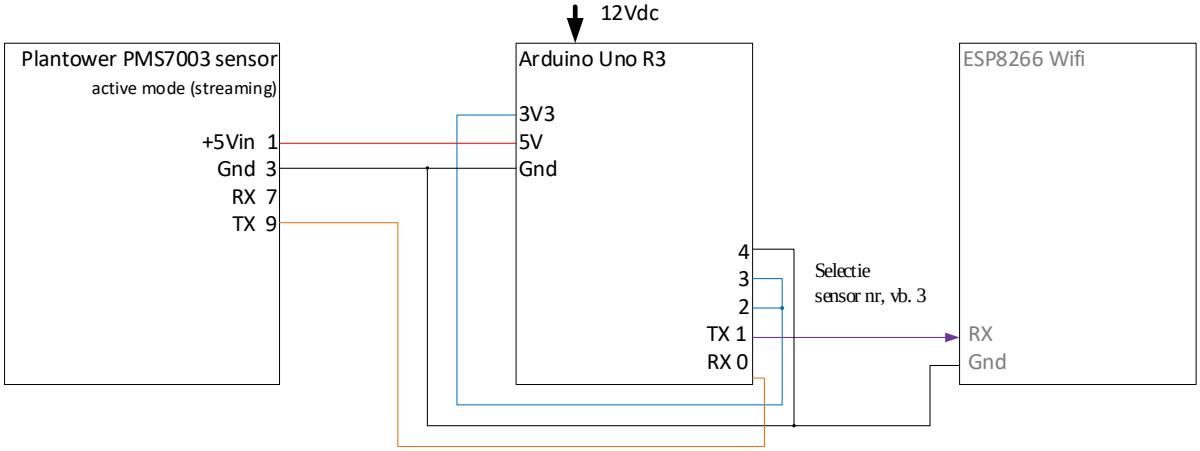
## Assemblage scheme

### Aansluitschema

***Aansluitschema initialisatie sensor voor gebruik***




***Aansluitschema sensor bij meting in streaming mode***




**Instelling sensor**

De sensoren zijn default ingesteld voor streaming (active mode).

Data output

**Structuur**

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[PM1];[PM2.5];[PM10];[PM1atm];
[PM2.5atm];[PM10atm];[part >0.3];[part >0.5];[part >1.0];[part >2.5];
[part >5.0];[part >10];[<ETX>[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Plantower PMS7003 is dit ‘E’ (0x45) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[PM1]` | PM1 concentratiemeting van sensor bij standaard omstandigheden (µg/m3), ‘.’ (0x2e) als decimaalteken |
| `[PM2.5]` | PM2.5 concentratiemeting van sensor bij standaard omstandigheden (µg/m3) |
| `[PM10]` | PM10 concentratiemeting van sensor bij standaard omstandigheden (µg/m3) |

| | |
|---|---|
| [PM1atm] | PM1 concentratiemeting van sensor bij omgeving omstandigheden (µg/m3) |
| [PM2.5atm] | PM2.5 concentratiemeting van sensor bij omgeving omstandigheden (µg/m3) |
| [PM10atm] | PM10 concentratiemeting van sensor bij omgeving omstandigheden (µg/m3) |
| [part >0.3] | aantal deeltjes met diameter > 0,3 µm in 0,1 liter lucht |
| [part >0.5] | aantal deeltjes met diameter > 0,5 µm in 0,1 liter lucht |
| [part >1.0] | aantal deeltjes met diameter > 1,0 µm in 0,1 liter lucht |
| [part >2.5] | aantal deeltjes met diameter > 2,5 µm in 0,1 liter lucht |
| [part >5.0] | aantal deeltjes met diameter > 5,0 µm in 0,1 liter lucht |
| [part >10] | aantal deeltjes met diameter > 10 µm in 0,1 liter lucht |
| <ETX> | vast, End of TeXt, 0x03 |
| [CheckSum] | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

**Sketch: 180628VQ_PMS7003_v100.ino**

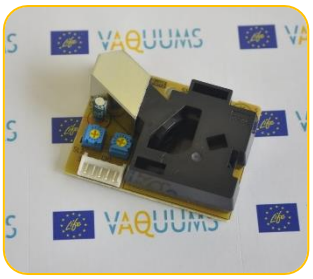
```
/* 180626VQ_PMS7003_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * Outputs mass concentration PM1, PM2.5, PM10 (Âµg/m3) and 6 particles number classes
 * every 1sec to serial, from a Plantower PMS7003 sensor.
 *
 * CONNECTING
 * Tx channel on sensor is connected to digital 0 (Rx)
 * Rx channel on sensor is only connected to digital 1 (Tx) on startup
 * afterwards when measuring the Rx is not connected
 *
 * 26/06/2018 VMM - Jan Adams
 */

const char cSensId = 'E';       // Sensor Id : E = Plantower PMS7003
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned int uiNrInAvg = 0;
unsigned long ulPMCnt[13];  // array for results from sensor, use unsigned long because the
particle counts can give big numbers
                            // [0]- PM1.0 conc standard   [3]- PM1.0 conc atmosph   [6]- cnt
diam >0.3Âµm   [9]- cnt diam >2.5Âµm
                            // [1]- PM2.5 conc standard   [4]- PM2.5 conc atmosph   [7]- cnt
diam >0.5Âµm   [10]- cnt diam >5Âµm
                            // [2]- PM10 conc standard    [5]- PM10 conc atmosph    [8]- cnt
diam >1.0Âµm   [11]- cnt diam >10Âµm
                            // [12]- reserved, not used
unsigned long ulPMCntSum[12]; // array for sum of measurements, same layout as ulPM_Cnt[],
[12] not included
unsigned long ulActMilli;
unsigned long ulSampleMilli;
char cCheckSum;
String dataString = "";

void setup(){
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180626VQ_Plantower_v100 sensorNr: ");
  Serial.println(uiSensNr);
```

```
  Serial.println();

  // set Initiaive upload mode - streaming, on by default
  PMS7003_InitActiveMode();
  Serial.println(" auto send streaming enabled, sensor initialised");
  Serial.println(" disconnect Sensor_RX from Arduino_TX, connect Arduino_TX to ESP8266_RX");
  delay(1000);       // wait 1s before starting to read auto send stream
  Serial.println("capturing data stream..");

  // init variables
  for (iPin =0; iPin <12; iPin++) {
    ulPMCntSum[iPin] =0;
  }
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}

void loop(){
  int iPnt;

  ulActMilli = millis();

  // read sensor
  if (PMS7003_ReadStream() ==2) {      // check for PMS7003 streaming data
    uiNrInAvg++;
    for (iPnt =0; iPnt <12; iPnt++) {
      ulPMCntSum[iPnt] = ulPMCntSum[iPnt] + ulPMCnt[iPnt];
    }
  }

  // output
  if (ulActMilli > ulSampleMilli) {
    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    for (iPnt =0; iPnt <6; iPnt++) {      // PM concentrations
      dataString += ";";
      dataString += String(float(ulPMCntSum[iPnt])/uiNrInAvg);
    }
    for (iPnt =6; iPnt <12; iPnt++) {     // particles counts
      dataString += ";";
      dataString += String(ulPMCntSum[iPnt]);
    }
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    for (iPnt =0; iPnt <12; iPnt++) {
      ulPMCntSum[iPnt] =0;
    }
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

```
// Set active mode in sensor connected to RX/TX pins
// sensor send no confirmation
void PMS7003_InitActiveMode() {
  byte bToSend[] ={0x42, 0x4D, 0xE1, 0x00, 0x01, 0x00, 0x8F};

  Serial.write(bToSend, sizeof(bToSend));
}


// Check for PMS7003 streaming and read data with time-out
// returns int: 0 - no data, time-out
//              1 - checksum failure
//              2 - data into array uiPMCnt[]
int PMS7003_ReadStream() {
  byte bReceive[33];
  byte bRecSer =false;
  int iPntArr =0;
  int iPntBit =0;
  int iReturn =0;
  unsigned int uiChecksum =0;
  unsigned long ulRxTO;
  ulRxTO =millis() +501;      // rx time-out 500msec = (1000/2)+1

  while (millis() <ulRxTO && (millis() <ulSampleMilli || bRecSer) && iPntBit <32) {
    if (Serial.available()) {
      bReceive[iPntBit++] =Serial.read();
      ulRxTO++;                          // increase Rx time/out with 2msec so complete message
will be received
      ulRxTO++;
      bRecSer =true;
    }
  }

  // if bytes received check content
  if (iPntBit ==32) {
    if ((bReceive[0] ==0x42) && (bReceive[1] == 0x4D) && (bReceive[2] == 0x00) && (bReceive[3]
== 0x1C)) {
      // transfer received bytes to PM-particles measurement array
      iPntBit =4;

      for (iPntArr =0; iPntArr <12; ++iPntArr) {
        ulPMCnt[iPntArr] =int(bReceive[iPntBit++])*256 +int(bReceive[iPntBit++]);
      }

      // calculate checksum of received data
      for (iPntBit =0; iPntBit <30; iPntBit++) {
        uiChecksum = uiChecksum +int(bReceive[iPntBit]);
      }
      if (uiChecksum ==(int(bReceive[30])*256+int(bReceive[31])) ) {
        // correct CS
        iReturn =2;
      } else {
        // CS failure
        Serial.print(" Checksum failure:");
        iReturn =1;
      }
    }
  }
  return iReturn;
}
```

VAQUUMS

Shinyei PPD42NJ



## General information

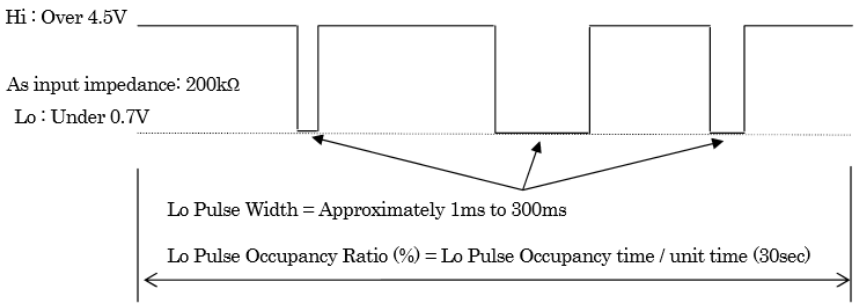
Meet: PM
Aansluiting: Pulse Width Modulation
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

Sensor ID
Sensor nummer
Aantal gedetecteerde pulsen in gemiddelde P1
Ratio P1 (%)
Aantal gedetecteerde pulsen in gemiddelde P2
Ratio P2 (%)

P1 uitgang geeft signaal voor deeltjes met een diameter van circa 1µm of groter
P2 uitgang geeft signaal voor deeltjes met een diameter van circa 2,5µm of groter




Afbeelding uit specificatie met voorbeeld uitgang P1 of P2 van sensor. Merk op dat uitgegaan is van een uitmiddelingtijd van 30sec. Ook in de voorbeeld sketches online gevonden werd uitgegaan van een uitmiddelingtijd van 30sec. Bij de omzetting naar een PM concentratie dient hiermee rekening gehouden te worden.

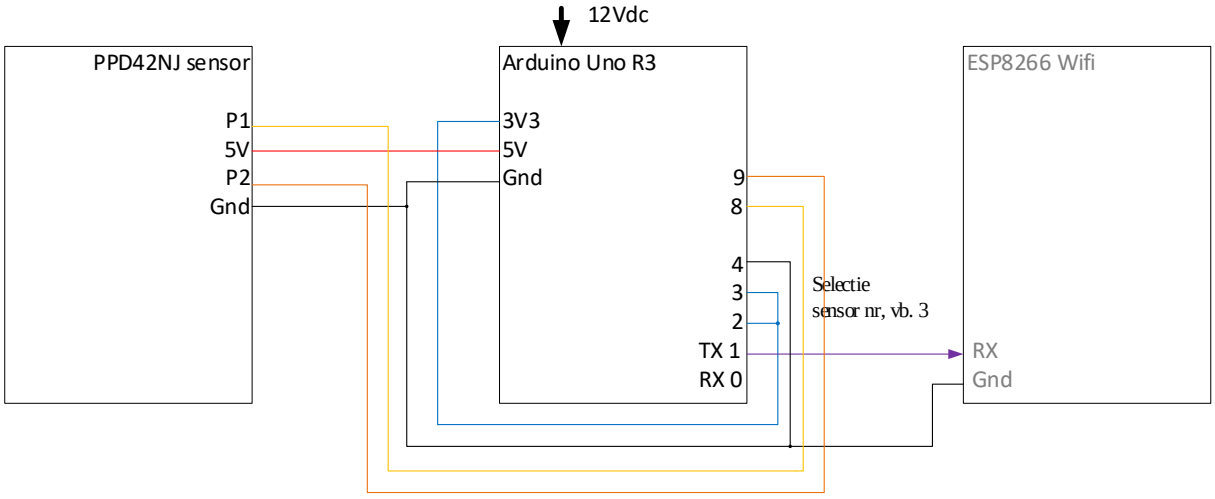
### Instelling

Sensor ID: 'B' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

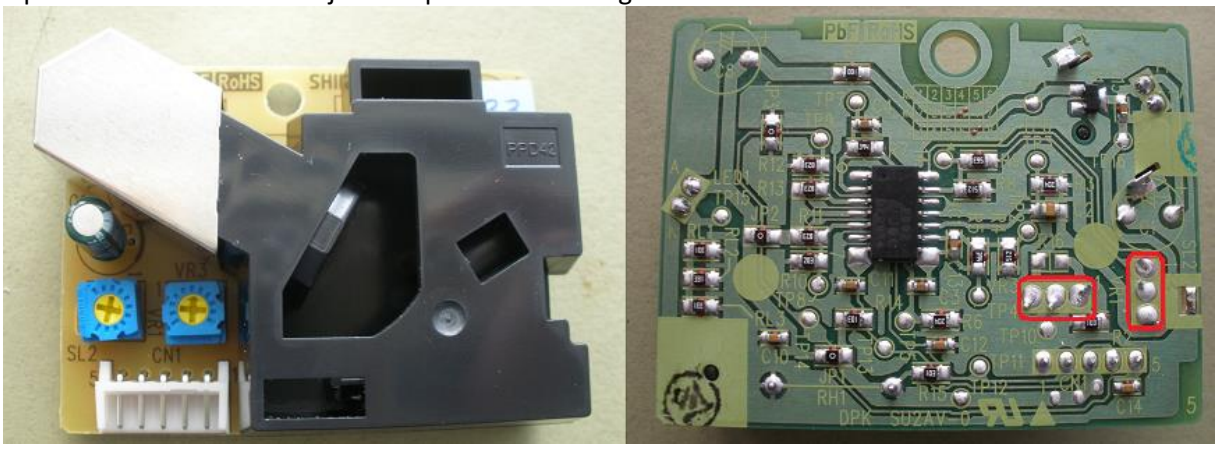
## Assemblage scheme

### Aansluitschema

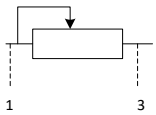



## Instelling sensor

Op de PPD42NJ sensoren zijn twee potentiometers gemonteerd.



VR1 : gain, versterking
VR3 : sensitivity, gevoeligheid
Volgens het document “ShinyeiPPD42NS_Deconstruction_TracyAllen.pdf” worden bij levering VR1 op ~60 kOhm en VR3 op ~54 kOhm gezet.
Om het te controleren op de geleverde sensoren kregen ze een uniek nummer : 0..6.
Bij het nameten van de geleverde sensoren tussen aansluiting 1 en 3 kwamen we op volgende waarden:




| Sensor | gemeten VR3 (kOhm) | gemeten VR1 (kOhm) |
|---|---|---|
| B0 | 77,64 | 94,96 |
| B1 | 110,46 | 121,79 |
| B2 | 107,36 | 94,06 |
| B3 | 102,68 | 110,28 |
| B4 | 108,38 | 114,48 |
| B5 | 125,81 | 113,68 |
| B6 | 93,43 | 99,49 |

## Data output

## Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalP1Gem];[RatioP1];[AantalP2Gem];[RatioP2];
  <ETX>[CheckSum]
```

| | |
|---|---|
| <STX> | vast, Start of TeXt, 0x02 |
| VQ | vast, hoofding VaQuums |
| [SensorID] | sensor ID, voor PPD42NJ is dit ‘B’ (0x42) |
| [SensorNr] | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| ; | vast scheidingskarakter ‘;’ (0x3b) |
| [AantalP1Gem] | aantal gedetecteerde pulsen van de sensor voor de berekening van het P1 ratio (%) |
| [RatioP1] | P1 ratio (%), ‘.’ (0x2e) als decimaalteken |
| [AantalP2Gem] | aantal gedetecteerde pulsen van de sensor voor de berekening van het P2 ratio (%) |
| [RatioP2] | P2 ratio (%), ‘.’ (0x2e) als decimaalteken |
| <ETX> | vast, End of TeXt, 0x03 |
| [CheckSum] | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

**Sketch: 180608VQ_PPD42NJ_v100.ino**

```
/* 180608VQ_PPD42NJ_v100
 * VAQUUMS project
 *
 * based on DustDuino Serial By Matthew Schroyer, MentalMunition.com
 * DESCRIPTION
 * Outputs ratios to serial, from a Shinyei PPD42NJ sensor.
 *
 * CONNECTING THE DUSTDUINO
 * P1 channel on sensor is connected to digital 8
 * P2 channel on sensor is connected to digital 9
 *
 * THEORY OF OPERATION
 * Measures the width of pulses through boolean triggers, on both channels.
 * Pulse widths are converted into a percent integer of time on.
 * No further calculations are done in sketch to minimise processing time and to deliver raw
data.
 * Calculation of particle counts and mass concentration must be done on next level.
 *
 * 08/06/2018 VMM - Jan Adams
 */

const char cSensId = 'B';       // Sensor Id : B = PPD42NJ
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
char cCheckSum;
String dataString = "";

unsigned int uiNrInAvgP1 = 0;
unsigned long ulTriggerOnP1;
unsigned long ulSumDurationP1 = 0;
boolean bValP1 = HIGH;
boolean bTriggerP1 = false;
float fRatioP1 = 0;

unsigned int uiNrInAvgP2 = 0;
unsigned long ulTriggerOnP2;
unsigned long ulSumDurationP2 = 0;
boolean bValP2 = HIGH;
boolean bTriggerP2 = false;
float fRatioP2 = 0;
```

```
void setup(){
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  // display header, version, sensor number
  Serial.print("180608VQ_PPD42NJ_v100 sensorNr: ");
  Serial.println(uiSensNr);

  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}

void loop(){
  ulActMilli = millis();

  // read sensor
  bValP1 = digitalRead(8);    // P1
  bValP2 = digitalRead(9);    // P2

  if(bValP1 == LOW && bTriggerP1 == false){
    // particle start detected for P1 channel, save start time Âµs
    bTriggerP1 = true;
    ulTriggerOnP1 = micros();
  }

  if (bValP1 == HIGH && bTriggerP1 == true){
    // particle end detected for P1 channel, save end time Âµs
    // calculate duration and add to sum
    ulSumDurationP1 += (micros() - ulTriggerOnP1);
    bTriggerP1 = false;
    uiNrInAvgP1++;
  }

  if(bValP2 == LOW && bTriggerP2 == false){
    // particle start detected for P2 channel, save start time Âµs
    bTriggerP2 = true;
    ulTriggerOnP2 = micros();
  }

  if (bValP2 == HIGH && bTriggerP2 == true){
    // particle end detected for P1 channel, save end time Âµs
    // calculate duration and add to sum
    ulSumDurationP2 += (micros() - ulTriggerOnP2);
    bTriggerP2 = false;
    uiNrInAvgP2++;
  }

  // output
  if (ulActMilli > ulSampleMilli) {
      // calculate ratios 0..100
      fRatioP1 = ulSumDurationP1/(ulSampleInterv*10.0);
      fRatioP2 = ulSumDurationP2/(ulSampleInterv*10.0);

      // calculation of counts, in comment to speed up sketch
      // float countP1 = 1.1*pow(fRatioP1,3)-3.8*pow(fRatioP1,2)+520*fRatioP1+0.62;
      // float countP2 = 1.1*pow(fRatioP2,3)-3.8*pow(fRatioP2,2)+520*fRatioP2+0.62;
      // float PM10count = countP2;
      // float PM25count = countP1 - countP2;

      // prepare output string and send to serial TX
      dataString = STX;
      dataString += "VQ";
      dataString += cSensId;
      dataString += String(uiSensNr);
      dataString += ";";
      dataString += String(uiNrInAvgP1);
```

```
    dataString += ";";
    dataString += String(fRatioP1);
    dataString += ";";
    dataString += String(uiNrInAvgP2);
    dataString += ";";
    dataString += String(fRatioP2);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    ulSumDurationP1 = 0;
    ulSumDurationP2 = 0;
    uiNrInAvgP1 = 0;
    uiNrInAvgP2 = 0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}
```

Shinyei PPD60PV-T2

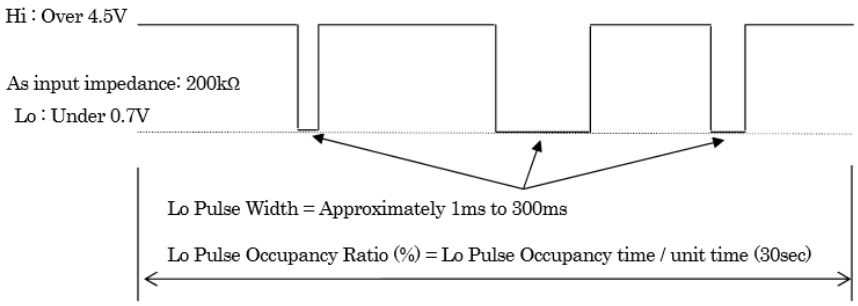
## General information

Meet: PM
Aansluiting: Pulse Width Modulation
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output
- Sensor ID
- Sensor nummer
- Aantal gedetecteerde pulsen in gemiddelde P1
- Ratio P1 (%)

P1 uitgang geeft signaal voor deeltjes met een diameter van circa 0,5µm of groter




Afbeelding uit specificatie met voorbeeld uitgang P1 van sensor. Merk op dat uitgegaan is van een uitmiddelingtijd van 30sec. Ook in de voorbeeld sketches online gevonden werd uitgegaan van een uitmiddelingtijd van 30sec. Bij de omzetting naar een PM concentratie dient hiermee rekening gehouden te worden.

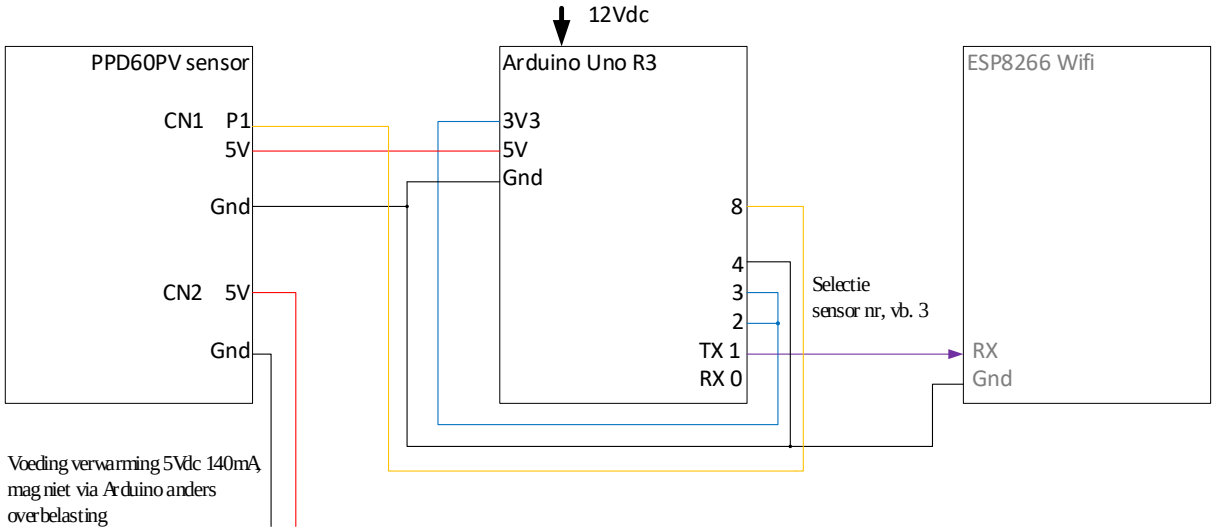
### Instelling

Sensor ID: 'C' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

## Assemblage scheme

### Aansluitschema



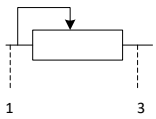

## Instelling sensor

Op de PPD60PV sensoren zijn twee potentiometers gemonteerd: VR1 en VR3.


In de documentatie staat geen informatie wat VR1 en VR3 juist doen. In analogie met de PPD42NJ mogen we veronderstellen dat: VR1 de gain, versterking regelt en VR3 de sensitivity, gevoeligheid.

Om het te controleren op de geleverde sensoren kregen ze een uniek nummer : 0..6.

Bij het nameten van de geleverde sensoren tussen aansluiting 1 en 3 kwamen we op volgende waarden:




| Sensor | gemeten VR3 (kOhm) | gemeten VR1 (kOhm) |
|---|---|---|
| C0 | 34,43 | 115,56 |
| C1 | 33,79 | 108,82 |
| C2 | 37,37 | 110,96 |
| C3 | 29,13 | 118,47 |
| C4 | 27,27 | 117,77 |
| C5 | 38,67 | 129,05 |
| C6 | 36,66 | 120,56 |

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalP1Gem];[RatioP1];<ETX>[CheckSum]
```

| | |
|---|---|
| <STX> | vast, Start of TeXt, 0x02 |
| VQ | vast, hoofding VaQuums |
| [SensorID] | sensor ID, voor PPD60PV is dit ‘C’ (0x43) |
| [SensorNr] | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| ; | vast scheidingskarakter ‘;’ (0x3b) |
| [AantalP1Gem] | aantal gedetecteerde pulsen van de sensor voor de berekening van het P1 ratio (%) |
| [RatioP1] | P1 ratio (%), ‘.’ (0x2e) als decimaalteken |
| <ETX> | vast, End of TeXt, 0x03 |
| [CheckSum] | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

**Sketch: 180613VQ_PPD60PV_v100.ino**

```
/* 180613VQ_PPD60PV_v100
 * VAQUUMS project
 *
 * based on DustDuino Serial By Matthew Schroyer, MentalMunition.com
 * DESCRIPTION
 * Outputs ratio to serial, from a Shinyei PPD60PV sensor.
 *
 * CONNECTING THE DUSTDUINO
 * P1 channel on sensor is connected to digital 8
 *
 * THEORY OF OPERATION
 * Measures the width of pulses through boolean triggers, on both channels.
 * Pulse widths are converted into a percent integer of time on.
 * No further calculations are done in sketch to minimise processing time and to deliver raw
data.
 * Calculation of particle counts and mass concentration must be done on next level.
 *
 * 13/06/2018 VMM - Jan Adams
 */

const char cSensId = 'C';       // Sensor Id : C = PPD60PV
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
char cCheckSum;
String dataString = "";

unsigned int uiNrInAvgP1 = 0;
unsigned long ulTriggerOnP1;
unsigned long ulSumDurationP1 = 0;
boolean bValP1 = HIGH;
boolean bTriggerP1 = false;
float fRatioP1 = 0;

void setup(){
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  // display header, version, sensor number
  Serial.print("180613VQ_PPD60PV_v100 sensorNr: ");
  Serial.println(uiSensNr);
```

```
  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
}

void loop(){
  ulActMilli = millis();

  // read sensor
  bValP1 = digitalRead(8);    // P1

  if(bValP1 == LOW && bTriggerP1 == false){
    // particle start detected for P1 channel, save start time Âµs
    bTriggerP1 = true;
    ulTriggerOnP1 = micros();
  }

  if (bValP1 == HIGH && bTriggerP1 == true){
    // particle end detected for P1 channel, save end time Âµs
    // calculate duration and add to sum
    ulSumDurationP1 += (micros() - ulTriggerOnP1);
    bTriggerP1 = false;
    uiNrInAvgP1++;
  }

  // output
  if (ulActMilli > ulSampleMilli) {
      // calculate ratios 0..100
      fRatioP1 = ulSumDurationP1/(ulSampleInterv*10.0);

      // prepare output string and send to serial TX
      dataString = STX;
      dataString += "VQ";
      dataString += cSensId;
      dataString += String(uiSensNr);
      dataString += ";";
      dataString += String(uiNrInAvgP1);
      dataString += ";";
      dataString += String(fRatioP1);
      dataString += ETX;
      cCheckSum = GetCheckSum( dataString);
      dataString += cCheckSum;
      Serial.print(dataString);

      // reset variables for next output
      ulSumDurationP1 = 0;
      uiNrInAvgP1 = 0;
      ulSampleMilli = ulActMilli + ulSampleInterv;
    }
  }


  // *** functions ***

  // calculate checkum with XOR of passed String until end '\0' is found
  // if result is 0 ('\0') change it to 1 to avoid faulty String end
  char GetCheckSum(String pIn) {
    int iXor =0;
    int i =0;
    while (pIn[i] !='\0') {
      iXor ^= pIn[i++];
    }
    if (iXor ==0) iXor =1;
    return char(iXor);
  }
```

Winsen ZH03B

## General information

Meet: PM1, PM2.5 en PM10
Aansluiting: RS232 TTL op 9600 baud, 8 bits, geen pariteit, 1 stopbit
Voeding: vanuit Arduino

### Output Arduino

Arduino Uno R3 zet string met gegevens met tussentijd van 1 seconde op TX (digital pin 1).
De sensor geeft zelf maar elke 2 seconde een datarecord. Het aantal in gemiddelde in de datarecords vanuit Arduino waarvoor er geen sensor metingen zijn is 0.
De opbouw van de string en een voorbeeld staat beschreven onder 4. Data output

- Sensor ID
- Sensor nummer
- Aantal in gemiddelde
- Concentratie PM1 (µg/m3)
- Concentratie PM2.5 (µg/m3)
- Concentratie PM10 (µg/m3)

### Instelling

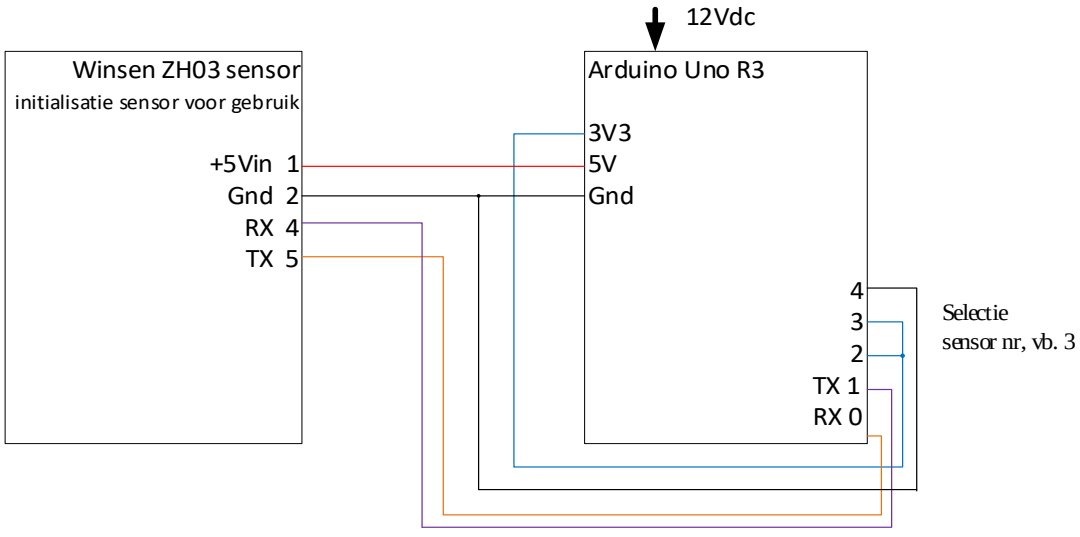
Sensor ID: 'F' geeft type sensor aan
Sensor nummer: In te stellen dmv. Arduino digital input 2..4
Laat toe om elke sensor van hetzelfde type een nummer te geven

| 4 | 3 | 2 | sensorNr |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

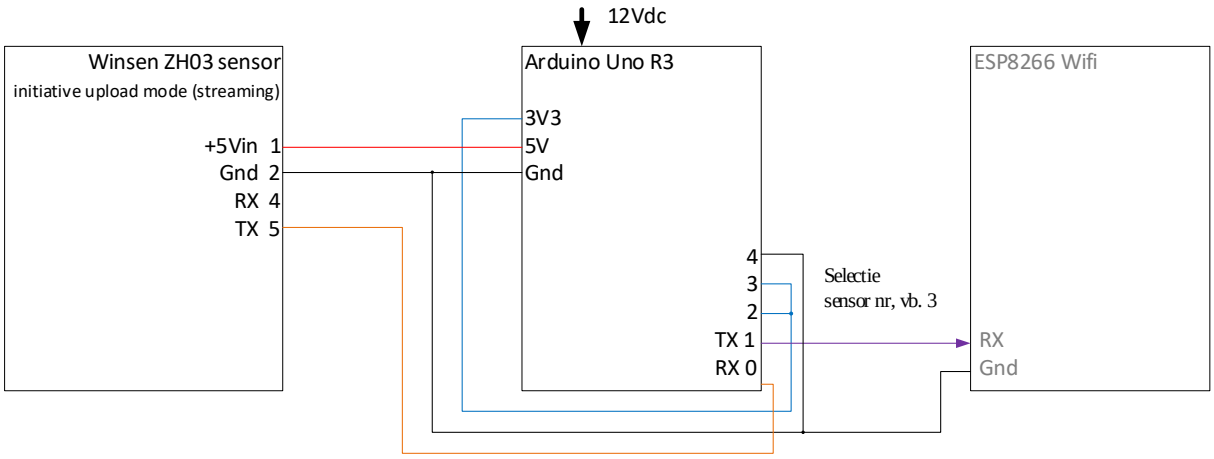
## Assemblage scheme

### Aansluitschema – instelling sensor

***Aansluitschema initialisatie sensor voor gebruik***



### *Aansluitschema sensor bij meting in streaming mode*




### Instelling sensor

De sensoren zijn default ingesteld voor streaming (initiative upload mode).

## Data output

### Structuur

```
<STX>VQ[SensorID][SensorNr];[AantalInGem];[PM1];[PM2.5];[PM10]<ETX>[CheckSum]
```

| | |
|---|---|
| `<STX>` | vast, Start of TeXt, 0x02 |
| `VQ` | vast, hoofding VaQuums |
| `[SensorID]` | sensor ID, voor Winsen ZH03 is dit ‘F’ (0x46) |
| `[SensorNr]` | sensor nummer zoals ingesteld met digitale ingang 2..4 van Arduino |
| `;` | vast scheidingskarakter ‘;’ (0x3b) |
| `[AantalInGem]` | aantal polls van de sensor voor de berekening van de gemiddelde metingen die volgt, 0 = geen metingen |
| `[PM1]` | PM1 concentratiemeting van sensor (µg/m3), ‘.’ (0x2e) als decimaalteken |
| `[PM2.5]` | PM2.5 concentratiemeting van sensor (µg/m3) |
| `[PM10]` | PM10 concentratiemeting van sensor (µg/m3) |
| `<ETX>` | vast, End of TeXt, 0x03 |
| `[CheckSum]` | XOR van alle voorgaande karakters inclusief <STX> en <ETX> |

## Programming sketch

### Sketch: 180625VQ_WinsenZH03_v100.ino

```
/* 180625VQ_WinsenZH03_v100
 * VAQUUMS project
 *
 * DESCRIPTION
 * Outputs mass concentration (Âµg/m3) every 1sec to serial, from a Winsen ZH03 sensor.
 * As the Winsen sensor sends data every 2sec half of the records don't contain data (NAN) and
N=0
 * N: number of measurments in a record
 *
 * CONNECTING
 * Tx channel on sensor is connected to digital 0 (Rx)
 * Rx channel on sensor is only connected to digital 1 (Tx) on startup
 * afterwards when measuring the Rx is not connected
 *
 * 25/06/2018 VMM - Jan Adams
 */
```

```
const char cSensId = 'F';       // Sensor Id : F = Winsen ZH03
const char STX = 2;
const char ETX = 3;
const unsigned long ulSampleInterv = 1000;    // interval (ms) sending sampled mean values on
TX pin, 1000ms - 1s

unsigned int uiSensNr = 0;
unsigned int uiPM1;
unsigned int uiPM10;
unsigned int uiPM25;
unsigned int uiNrInAvg = 0;
unsigned long ulPM1sum;
unsigned long ulPM10sum;
unsigned long ulPM25sum;
unsigned long ulActMilli;
unsigned long ulSampleMilli;
char cCheckSum;
String dataString = "";

void setup(){
  int iPow2 = 1;
  int iPin;

  Serial.begin(9600);     // RX0, TX0:  9600,8,N,1

  // read Arduino pin 2..4, use as Sensor Number
  // ex. pin 4 3 2 = 1 0 0 -> 4
  for (iPin =2; iPin <5; iPin++) {
    pinMode(iPin, INPUT);
    uiSensNr += digitalRead(iPin) * iPow2;
    iPow2 *= 2;
  }

  Serial.print("180625VQ_WinsenZH03_v100 sensorNr: ");
  Serial.println(uiSensNr);
  Serial.println();

  // set Initiaive upload mode - streaming, on by default
  ZH03_InitUploadMode();
  Serial.println(" auto send streaming enabled, sensor initialised");
  Serial.println(" disconnect Sensor_RX from Arduino_TX, connect Arduino_TX to ESP8266_RX");
  delay(1000);       // wait 1s before starting to read auto send stream
  Serial.println("capturing data stream..");

  // init variables
  ulActMilli =millis();
  ulSampleMilli =ulActMilli +ulSampleInterv;
  ulPM1sum =0;
  ulPM10sum =0;
  ulPM25sum =0;
}

void loop(){
  ulActMilli = millis();

  // read sensor
  if (ZH03_ReadStream() ==2) {      // check for ZH03 streaming data
    uiNrInAvg++;
    ulPM1sum = ulPM1sum + uiPM1;
    ulPM10sum = ulPM10sum +uiPM10;
    ulPM25sum = ulPM25sum +uiPM25;
  }

  // output
  if (ulActMilli > ulSampleMilli) {
    // prepare output string and send to serial TX
    dataString = STX;
    dataString += "VQ";
    dataString += cSensId;
    dataString += String(uiSensNr);
    dataString += ";";
    dataString += String(uiNrInAvg);
    dataString += ";";
    dataString += String(float(ulPM1sum)/uiNrInAvg);
    dataString += ";";
    dataString += String(float(ulPM25sum)/uiNrInAvg);
    dataString += ";";
```

```
    dataString += String(float(ulPM10sum)/uiNrInAvg);
    dataString += ETX;
    cCheckSum = GetCheckSum( dataString);
    dataString += cCheckSum;
    Serial.print(dataString);

    // reset variables for next output
    uiNrInAvg =0;
    ulPM1sum =0;
    ulPM10sum =0;
    ulPM25sum =0;
    ulSampleMilli = ulActMilli + ulSampleInterv;
  }
}


// *** functions ***

// calculate checkum with XOR of passed String until end '\0' is found
// if result is 0 ('\0') change it to 1 to avoid faulty String end
char GetCheckSum(String pIn) {
  int iXor =0;
  int i =0;
  while (pIn[i] !='\0') {
    iXor ^= pIn[i++];
  }
  if (iXor ==0) iXor =1;
  return char(iXor);
}


// Set initiative upload mode in sensor connected to RX/TX pins
// sensor send no confirmation
void ZH03_InitUploadMode() {
  byte bToSend[] ={0xFF, 0x01, 0x78, 0x40, 0x00, 0x00, 0x00, 0x00, 0x47};

  Serial.write(bToSend, sizeof(bToSend));
}


// Check for ZH03 streaming and read data
// time-out of 251msec
// returns int: 0 - no data, time-out
//              1 - checksum failure
//              2 - data, PM1, PM2.5 and PM10 into variables uiPM1, uiPM25 and uiPM10
int ZH03_ReadStream() {
  byte bRecSer =false;
  byte bReceive[25];
  int iPnt =0;
  int iReturn =0;
  unsigned int uiChecksum =0;
  unsigned long ulRxTO;
  ulRxTO =millis() +251;      // rx time-out 251msec = (1000/4)+1

  while (millis() <ulRxTO && (millis() <ulSampleMilli || bRecSer) && iPnt <24) {
    if (Serial.available()) {
      bReceive[iPnt++] =Serial.read();
      ulRxTO++;                          // increase Rx time/out with 2msec so complete message
will be received
      ulRxTO++;
      bRecSer =true;
    }
  }

  // if bytes received check content
  if (iPnt >0) {
    if ((bReceive[0] ==0x42) && (bReceive[1] == 0x4D) && (bReceive[2] == 0x00) && (bReceive[3]
== 0x14)) {
      uiPM1 =int(bReceive[10])*256 +int(bReceive[11]);
      uiPM25 =int(bReceive[12])*256 +int(bReceive[13]);
      uiPM10 =int(bReceive[14])*256 +int(bReceive[15]);

      // calculate checksum of received data
      for (iPnt =0; iPnt <22; iPnt++) {
        uiChecksum = uiChecksum +int(bReceive[iPnt]);
      }
      if (uiChecksum ==(int(bReceive[22])*256+int(bReceive[23])) ) {
```

```
        // correct CS
        iReturn =2;
      } else {
        // CS failure
        Serial.print(" Checksum failure:");
        iReturn =1;
      }
    }
  }
  return iReturn;
}
```