Sensors UAV – SW project specifications

Sommario

[01. Description 1](#_Toc139931930)

[02. Configuration file 1](#_Toc139931931)

[03. Software systems components of Raspberry 2](#_Toc139931932)

[03.1. Configurator 3](#_Toc139931933)

[03.2. Arduinomanager 3](#_Toc139931934)

[03.3. Scddetectionmanager 3](#_Toc139931935)

[03.4. Csvwritermanager 3](#_Toc139931936)

[03.5. Simulator 3](#_Toc139931937)

[03.6. Utilitiesmanager 3](#_Toc139931938)

[03.7. Sensorscsv 3](#_Toc139931939)

[03.8. Mainsensors 3](#_Toc139931940)

[04. Reading logic 3](#_Toc139931941)

# Description

Aim of the project is controlling and coordinating the system onboard of the UAV. In the system different sensors are used for monitoring the air quality.

MQ sensors are resides in Arduino system. Selected sensors are **MQ-4** (detection of CH4), **MQ-7** (detection of CO), **MQ-5** (general monitor of air quality), **MQ-3** (alcohol sources in air), **MQ-135** (detection of Ammonia) and **MQ-2** (detection of combustible sources). Those sensors are linked to analog ports of Arduino.

Onboard of Raspberry the **SCD41** sensor took place. This component can detect **CO, Humidity (RH)** and **Temperature**. Ticks for the last two mentioned variables are also detected. The sensor is linked to the controller using the **I2C protocol**.

Arduino and Raspberry communicates with the **UART** protocol by the mean of a serial port.

# Configuration file

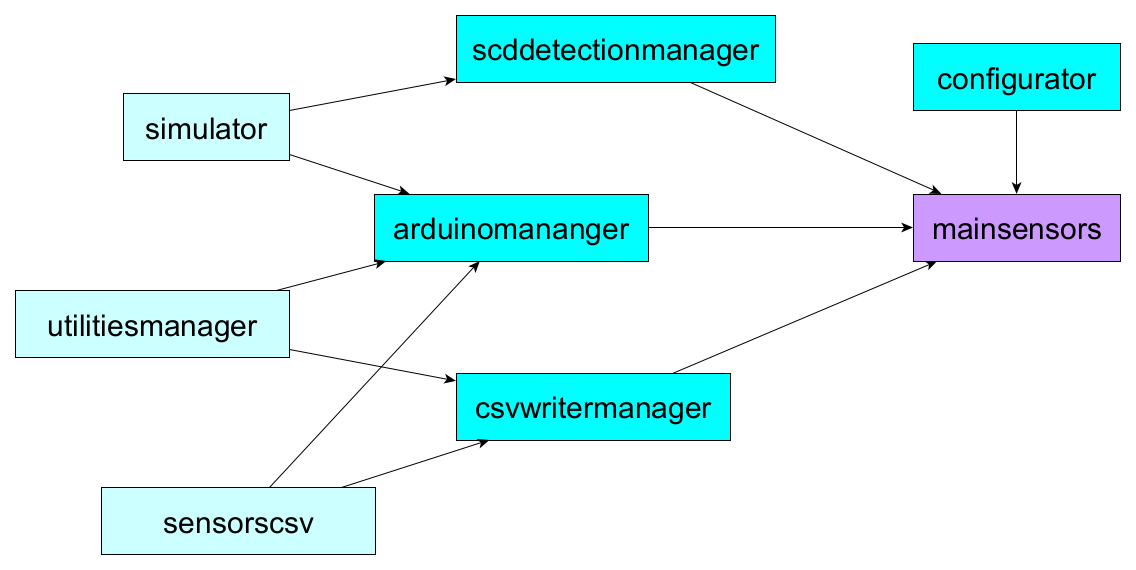
Since there are variable elements in the system, the software project is provided with a Configuration file (**analysisConf**). This file resides in same folder of the main procedure and is read before starting the program. Depending of the inputs, different effects can occur.

Configuration file is an xml document and here are shown the main parameters which is possible to give to the procedure:

1. **csv\_max\_line:** it dictates the maximum number of rows that is possible to write in an output CSV file. This file is produced collecting the different sensors data and extra useful information for the overall interpretation. When the maximum number of lines indicated here is reached, the file is moved in the download folder (see below) and a new file will be generated.
2. **main\_procedure\_path:** it is the output principal path. From here different folders are eventually created for content diversification, and from here starts the files management (see below)
3. **sensors\_folder\_out:** from the above path **(2)**, this folder will be eventually created by the procedure. In this folder it is possible to find the current CSV file elaborated by the program. As already mentioned, lines of sensors measurements will be appended to this file until the **csv\_max\_line** number is reached for the content.
4. **download\_folder\_out:** this folder will be created from the **main\_procedure\_path** too. In this folder the completed CSV will be moved from the **sensors\_folder\_out**. File will be unique, since a date timestamp identifier will be prepended in the name of each of them.
5. **csv\_basic\_name:** the static part of every produced output data file. For be distinguishable, the CSV will be created as **‘yyyyMMddhhmmss\_csv\_basic\_name.csv’**.
6. **arduino\_activated:** this Boolean input is responsible for the Arduino activation. If True in fact, it will allow the procedure to read directly from the board, on the contrary case a simulation method will be launched. Those simulation method allows to maintain the coherence of read data in the single row of the CSV file.
7. **arduino\_serial\_port:** it reports the name of the serial port of the machine to which the Arduino board is connected. Responsible functionality for the read will ask to this port for the data in case the above flag **(6)** is set to True.
8. **arduino\_baud\_rate:** the rate at which information are read from the Arduino serial port **(7)** in case the flag for Arduino activation **(6)** is set to True.
9. **arduino\_duty\_time:** always in case in which Arduino is activated, it expresses the delay to give for each read from the Arduino board using serial port module.
10. **scd\_activated:** this flag allows to use the **SCD41** sensor in case of connection. If the configuration is set to False instead, the data generally read from this sensor will be replaced by simulation parameters again, for maintaining information coherence in the CSV.
11. **scd\_i2c\_port:** the port to which **SCD41** measurements are read using the **I2C protocol** in case the component is activated by **(10).**
12. **scd\_duty\_cycle:** the time in seconds for reading the data originated by the **SCD41** sensor in case of activation. This parameter is important because of functional characteristics of adopted modules.
13. **sys\_analysis:** write here the system in which the scripts are running. Selecting **‘WIN’** the system is assumed to be Windows, selecting **‘RASP’** it will be Raspberry instead (this value will be used by default in any case). This characteristic is important to the software system for specializing some specific lines of code that are peculiar to one of the 2 mentioned systems.

# Software systems components of Raspberry

Following diagram shows the modules used for programming Raspberry:



*Image 1: relations in the Python sensors project.*

In the image the **mainsensors** is the unique purple module because is the responsible of starting the program and of coordination of principal actions.

The darker azure modules are the ones used from parallel tasks for the realization of actions (**scddetectionmanager, arduinomanager, csvwritermanager**) or are elements directly related to the start of the procedure (**configurator**).

The lighter azure means modules that are of support of the entire elaboration (**simulator, utilitiesmanager, sensorscsv**).

## Configurator

This module reads the above described configurations (see the chapter 2) and offers them in an object visible to all the other components via **mainsensors** module.

In the mentioned object some attributes realize the parallel communication between parallel tasks. A multi process **queue** is available for the Arduino sensors data and another one for the SCD sensor.

Once the header of the CSV file (it does not change from file to file) is available, it will be placed in an array of this object too for a newer creation of output file.

## Arduinomanager

The functionalities for the Arduino interaction (or simulation) are here contained. The component is also responsible for the starting of the CSV creation: from here the primary header line is created and placed in the configuration object, adding the extra information for the other sensors data too.

## Scddetectionmanager

The module responsible for the SCD data read or simulation either. Those data are separated from the read of the other inputs, another queue is used for them.

## Csvwritermanager

Main procedure for the writing and moving of the CSV files. Enqueued information for the sensors are here dequeued for be placed (after an eventual processing) inside the principal CSV file. Once the document reaches the maximum row length, it will be moved in the download folder and another file will be generated.

## Simulator

Logic for the replacement of real sensor data with some random data is here developed. As already written, those “padding” information are import for not loosing the CSV data coherence. Methods for Arduino and SCD simulations are here provided.

## Utilitiesmanager

Lists of functionalities that helps in obtaining the correct format of data in different contexts.

## Sensorscsv

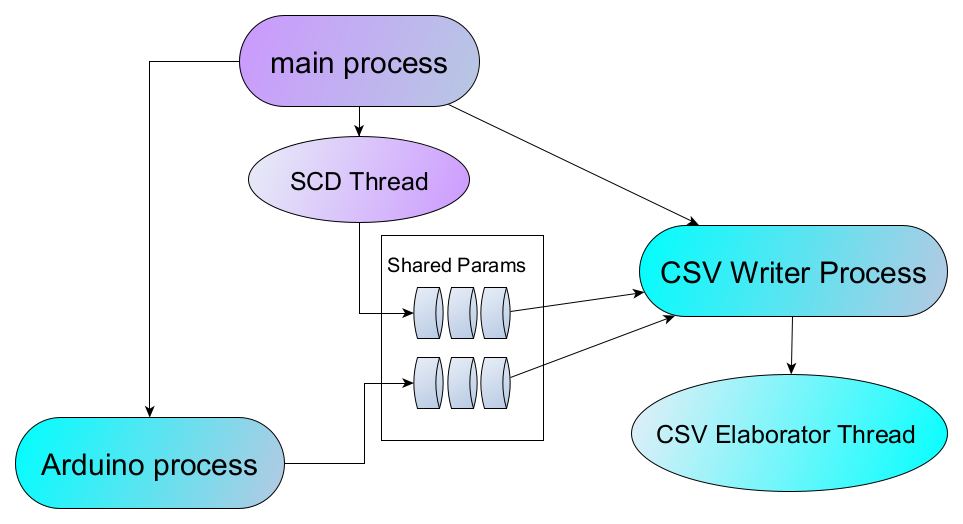
It helps the formatting of some information for the Arduino context.

## Mainsensors

Program entrance: It is responsible for giving the correct initial directives in the procedure.

# Reading logic

Image of this chapter illustrates how the different tasks works:



*Image 2: the elaboration scheme.*

The main process generates the SCD Thread: this thread is necessary for setting correctly the **SCD duty cycle** measure managing the correct functioning of the sensor. Data of provenience of this sensor are enqueued in the specific queue.

From the main process 2 child processes are generated: the Arduino process works on the **queue of MQ data** of the shared object whereas the **CSV Writer Process** will produce the document output.

CSV Write Process produces the CSV file **unifying information** from the 2 queues and an extra Thread will be launched from it: this parallel task will **eventually elaborate** the information of all the files moved in the download folder and not affected by multi operations anymore.

It the overall system is important to keep elements coherence from multi process concurrence. For this reason elements in the shared params object are accurately kept safe from eventual parallel operations on them.