Towards data reliability based on triple redundancy and online outlier detection

Sylvain Poupry, Cédrick Béler, Kamal Medjaher

Laboratoire Génie de Production, ENIT Toulouse INP, 47 Avenue d'Azereix, Tarbes, 65000, France
sylvain.poupry@enit.fr
cedrick.beler@enit.fr
kamal.medjaher@enit.fr

ABSTRACT

Today, air quality monitoring is a global concern. The World Health Organization (WHO) defined standards for each pollutant and each member state is committed to monitoring them continuously and reliably to protect the population. This responsibility is delegated to air quality monitoring associations. To achieve the objectives of reliable, accurate, and continuous measurements, these associations rely on conventional measuring stations with demanding specifications to serve as scientific references and decision supports for the authorities. However, because of heavy investments and required qualified staff, there are few stations and the coverage is coarse for territories of several thousand km². To circumvent this difficulty, measurement network architectures using Low-Cost Sensors (LCS) have been deployed. Low cost and requiring less qualification, This alternative technology to conventional measuring stations makes it possible to target local pollution that could not otherwise be detected. Although it is more accurate on the spatial dimension, this technology has several drawbacks, notably in terms of measurement repeatability and hardware quality. It is also subject to measurement drifts over time. To overcome these drawbacks, a resilient and reliable architecture based on LCS and triple redundancy has been proposed. The basic principle is based on the implementation of three smart sensors (SmS) using LCS measuring the same parameters on the same perimeter. These SmS communicate with an Aggregator that aggregates the data sent by SmS. The aggregator includes also detection and voting tasks allowing to compare, cross the data, detect faults of LCS online, and provide data that are ready for processing. In this paper, a pre-processing algorithm in four steps is presented. It identifies hardware faults from one or more LCS and reports outliers for verification by an expert. It is configurable and can identify failure behaviors (LCS or air quality). Finally, the proposed algorithm excludes the outliers data from faulty LCS and presents only reliable ones.

1. INTRODUCTION

Air pollution is the cause of 4.2 million deaths every year, not to mention the impact on wildlife. Based on this fact, air pollution is continuously monitored in a reliable and accurate way by air quality associations. As a scientific reference, these associations allow the authorities to take decisions in case of alert and to protect the population. However, these measuring stations require heavy investments and qualified personnel, and only few stations are deployed. As a consequence, the monitoring coverage is coarse and, despite extrapolations, local pollutant phenomena on territories of several km² are not detected.

In addition to the monitoring of air quality associations, deployments of measurement networks are carried out with low cost sensors (LCS) (Morawska et al., 2018). These deployments were facilitated as the LCS are inexpensive aspect (in the order of x10 to x100) and require less qualified personnel. The spatial dimension of these networks is a strong advantage, in particular for detecting local pollution and specifying the extrapolations of air quality associations (Castell et al., 2017). However, the measurements, at the level of a geographical point, present problems of precision and reliability. Indeed, LCSs have several drawbacks with respect to their material quality, measurement drift, cross-interference with other pollutants and their lifetime (Lewis et al., 2016). As a consequence, the reliability of each point of the network is questioned and the continuity of the measurement depends on the random lifetime of the LCS.

To overcome these problems, a resilient and reliable measuring station based on LCS and triple redundancy was developed. The station monitors the pollutant concentrations at a geographical point of the measurement network. It is located in a measurement perimeter where the environmental parameters do not vary at any point within the perimeter. It is com-

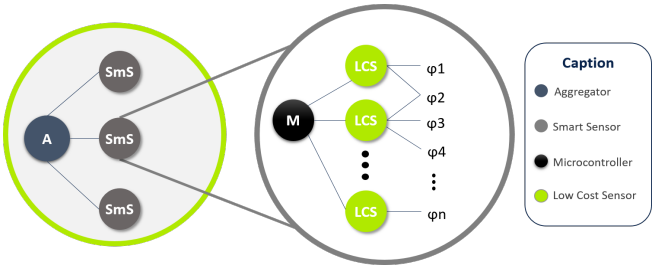


Figure 1. Measuring station composition.

posed of three (and can be extended to more) smart sensors (SmS) and an Aggregator, as shown in the figure 1.

The redundancy is active and is located at the level of the SmS. With a minimum of three, each SmS measures the same number N of parameters under the same environmental conditions. The SmS is composed of a microcontroller and the LCS measuring the N parameters (φi). The microcontroller concatenates and aggregates the LCS measurements into a vector φ and then communicates with the Aggregator and transmits the data at a frequency F, as shown in figure 2.

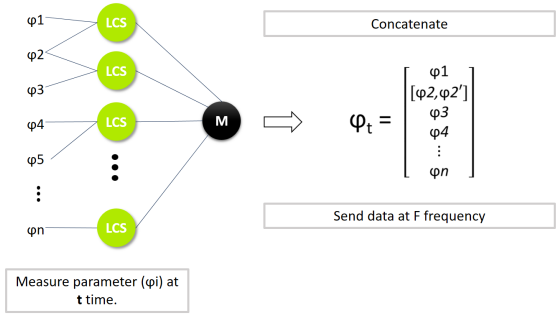


Figure 2. Processing of SmS measurements.

The Aggregator receives the vectors φ from each SmS, restructures the data by parameters (φi), and stores them. This first processing aggregates the raw data of the relevant LCS measurements from each SmS. The figure 3 gives an example of raw data after the restructuration step.

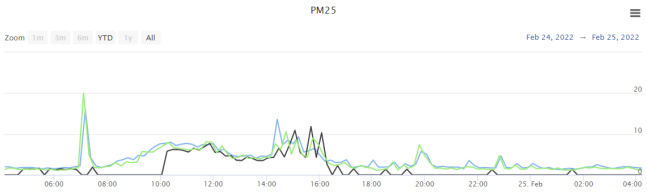


Figure 3. PM2.5 concentration measurements over one day.

However, the sending of data to the Aggregator by the SmS is not synchronized. Indeed, each SmS communicates with its own P-period making comparisons between measurements difficult. Moreover, the variable quality of the sensitive elements leads to repeatability problems. The detection of certain peaks can also be questioned. Finally, the failure of an LCS can affect the global synthesis for the targeted parameter (as shown by the black curve in figure 3). In summary, the problem of the synthesis of the measuring station is at three levels: measurement comparison, outliers detection to keep the most reliable data, and finally, the final synthesis which must be the most faithful image of the air quality measured while targeting the failing LCS or in a drift phase.

To remedy to the above mentioned issues, a method is proposed in this paper. It consists of four steps: Restructuring, Detection, Filtering, and Aggregation. It exploits the independence of the SmS and provides a global synthesis of the measurement station. An algorithm exploiting this method is implemented in the aggregator. At the input, the LCS measurements are grouped by parameters and, at the output, a synthesis exploitable in real-time is provided with the discarded data for identification of the failures. A confidence index Ic, based on the number of LCS errors, is also introduced. It is an indicator of the measuring station health state. Indeed, the station works continuously, despite the state of its components and the difficulty to maintain them because of their difficult access. The confidence index Ic is then a quick way to assess the integrity of the station operation and the reliability of the acquired data.

This paper is structured as follows: the relevant state of the art concerning detection and voting algorithms is presented in section 2, the methodology and the description of the algorithm are presented in section 3, section 4 presents the first results obtained from our case study and section 5 concludes the paper and gives some perspectives about the presented work.

2. State of the art

The architecture of the measuring station is inspired by the triple modular redundancy (TMR) which is widely used in the industry for high availability and reliability of critical applications. The principle behind it is based on three identical and independent modules operating in parallel and having same inputs. The output of these modules is submitted to a voting unit to create an output and generate a synthesis. The aggregations performed by the voting unit are generally the majority vote, the median vote, and the weighted average vote (Lorczak, Caglayan, & Eckhardt, 1989).

The majority voting algorithms make the system fault-tolerant by selecting the output corresponding to the majority of the modules' outputs. Otherwise, the output is a safety code to safely shut down the system. The latest developments of the majority voting algorithm use the historical data to optimize the choice and thus make the system more reliable. However, this kind of majority voting algorithm has two major drawbacks. The first drawback is that when the outputs of the

modules are not close (which is the case for LCS), a threshold must be defined in order to group the close values so that the algorithm considers them identical and, when the majority is not reached, the system is stopped. The second disadvantage comes from the fact that the failures are masked. Indeed, the output corresponds to the most frequent value but the discarded values are not exploited.

Weighted average voting algorithms average the outputs of the modules with weights assigned to each output. These weights are calculated from their respective deviations. The larger the deviation, the smaller the weight. New algorithms are proposed (Latif-Shabgahi, 2004) and are more reliable according to their authors than majority voting when errors are present. Their main advantage is to provide an output whatever the number of modules present (contrary to the voting algorithms). However, when the errors are large, the number of incorrect outputs of the algorithm increases. This observation comes from the use of the average. Indeed, the mean is influenced by the extremes, and the more outliers in the inputs, the more the mean will be influenced (Leys, Ley, Klein, Bernard, & Licata, 2013). Moreover, the performance of these algorithms depends on the choice of weights.

The median voting algorithms are more efficient than those based on the weighted average. Indeed, the use of the median can allow getting rid of the influence of outliers. But the reliability of such an algorithm is diminished when the majority of the values are outliers (Bass, Latif-Shabgahi, & Bennett, 1997).

Each type of voting algorithm performs well when the input errors are a minority of the total output. They are optimized to provide an output with the least error. The combination of algorithms and the use of classification to find the best output increases the reliability but also the computational complexity and the processing time (Kassab, Hashad, Taha, & Shedied, 2013). However, for real-time processing, computation time must be taken into account. Nevertheless, whatever the combination of algorithms, they retain their weakness and remain influenced by the errors in the inputs. Finally, the common point of these algorithms is the masking of errors. This is why, in order to increase the performance of the algorithms, a step of data-driven fault detection is proposed and implemented upstream of the aggregation in the voting unit. Indeed, according to the authors of this study (Kucera, Hyncica, Cidl, & Vasatko, 2006), this configuration allows to make a TMR system more reliable.

For fault detection methods, the closest domain of the application addressed in this paper is the Wireless sensor networks (WSN). Indeed, when several stations are deployed, each station can be assimilated to a sink node where the SmS correspond to sensor nodes. The configuration is even more similar because the SmS transmit the measurements with their period P. There are various fault detection techniques and a qualitative comparison of the latest fault detection algorithms for the deployment of WSN is listed in this reference (Muhammed & Shaikh, 2017). Among all possible techniques, the choice is motivated by a distributed self-fault diagnostic of sensor nodes, which are the SmS in this paper. For a large-scale deployment, each measuring station must identify its faults in order not to increase the computational complexity at the global network level. Thus, in (Panda & Khilar, 2015), an algorithm is proposed using the normalized median standard deviation to detect and discard the outliers. Consisting of two phases, the first phase of the algorithm aims to harvest measurements during an estimation time and associate them with LCS identification. The second phase discards the outliers by a statistical method using the Normalized Median Absolute Deviation (MADN). However, although the first phase is inspiring for the detection step, the calculation of MADN assumes that the distribution of the measurements is normal, which is not the case of measurements related to natural phenomena.

In conclusion, in order to make the Aggregator synthesis more reliable, it is important to incorporate a fault detection step upstream of the voting unit in order to decrease the output errors. Then, for the most reliable data provided at the output, a filtering will be applied to decrease the noise. Finally, a median voting algorithm is proposed. It is less sensitive to the extreme values compared to the average and is also more reliable than the majority voting algorithms. Therefore, the contribution of this paper consists in the development of the algorithm in four steps:

- Data restructuring of SmS vectors φ;
- Fault Detection and storage of outliers;
- Filtering (curve smoothing);
- Aggregation by the median voter.

3. METHOD

For the input data (measurements) of the SmS, the following assumptions are made:

- Being implemented in a measurement perimeter, the LCS measure the same parameters under the same environmental conditions;
- The output data of the SmS from a parameter φi are the result of measurements and uncertainties of the corresponding LCS;
- The air pollution phenomenon is slow by its physical nature. Therefore, a sampling of the order of a minute would be sufficient.

In the propose method, the Aggregator receives the data from each SmS, stores them and restructures them. Then, a detection step is applied to detect faulty components and discard outliers to provide useful data. These date are then filtered and aggregated with the median voting method. As output,

we obtain a reliable synthesis, a list of errors detected with the identified components, and a confidence index correlated to the number of reliable LCS. Figure 4 summarizes the set of steps handling the input/output within the Aggregator.

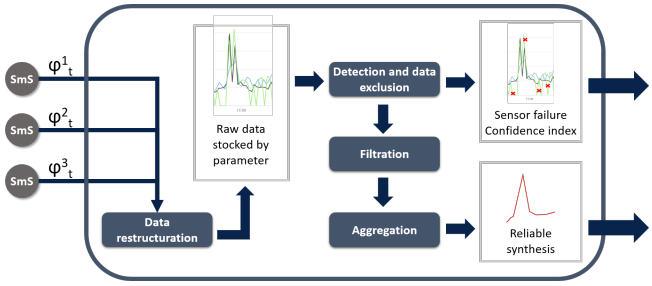


Figure 4. Aggregator's functions.

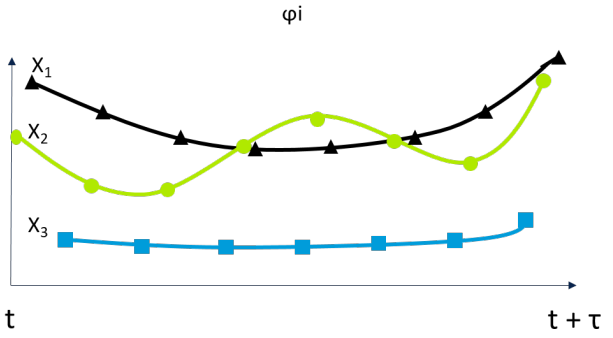
3.1. Data restructuration

Each SmS transmits to the Aggregator a measurement vector φ^j at a period P. This first step consists in reading the values of the vector and then storing them by parameter (φi) by associating the SmS identifier and the time of the measurement. This classification is essential to find the corresponding LCS from the SmS identification since each measured parameter is associated with a specific sensor. Thus, at the end of this restructuration step, for each SmS numbered j and for each parameter (φi), a time-series X_j is created, where x_h is the measurement and t the associated timestamp (Eq. (1)). The obtained time-series are then used by the fault detection step.

$$X_j = [x_{1,t}, x_{2,t+P}, ..., x_{h,t+hP}, ...x_{n,t+nP}] \quad (1)$$

3.2. Fault detection step

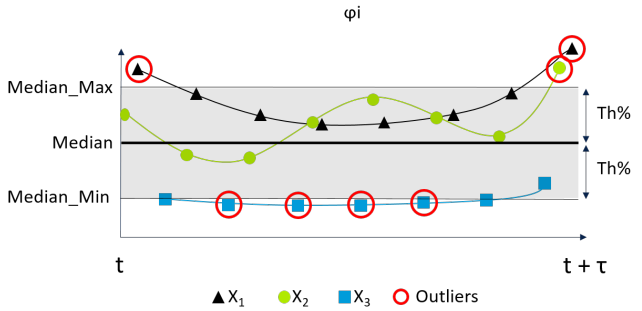
The fault detection step applies for every parameter. It consists of two phases: initialization and detection.



$$X_j = \{x_{1,t}, x_{2,t+P}, ..., x_{h,t+hP}, ..., x_{n,t+nP}\}$$
$$\text{with } \tau > P \text{ and } nP < \tau$$

Figure 5. Raw data after initialisation step

The first phase consists in retrieving the measurements for an estimation time τ corresponding to the desired number of points. During initialization, τ is used to define the window size to apply a rolling window on the data set as a step-time. Its size is defined with respect to the periods of the SmS and the average time of the monitored air pollution evolution. It also must be greater than the largest value of the SmS periods P to group at least one value of each LCS. Figure 5 illustrates raw data sampling for the parameter (φi) with $\tau = 7P$. Due to the difference in the P period between SmS, the third SmS gives seven points instead of eight for the others.



$$Median_{Max} = Median + Th\%$$
$$Median_{Min} = Median - Th\%$$

Figure 6. Raw data after initialisation step.

The second phase (detection) allows to identify the outlier data from the failed LCS. It consists, first of all, in checking the size of the time-series X_j. Indeed, missing data means that a hardware fault occurred on the associated SmS number j leading to a non-transmission of its data. The fault could be due to network failure, power failure or SmS fault. This detection triggers an alert at the output of the Aggregator which is then stored in the $Alert$ series with the corresponding timestamp. The number of alerts in this series influences the confidence index which is correlated to the number of errors of LCS in service. Then, after checking the size of the times-series, the next step consists in calculating the median of all the concatenated measurement values. The median has been chosen in order to follow as closely as possible the pollution peaks, which is not the case of the mean that tends to crush them. Then a threshold deviation Th is set to calculate the $Median_{Max}$ and $Median_{Min}$ values. This deviation, defined in percentage, is estimated from the historical data. It corresponds to the maximum variability allowed between the LCS values and improves the sensitivity of the detection, as indicated in figure 6.

The values outside the area delimited by $Median_{Max}$ and $Median_{Min}$ are considered as outliers. They are stored in a time-series X_{jOut} associated to the SmS number j. The conservation of these values at the output of the Aggregator will allow to contextualize them with the following window

in order to be able to differentiate if they are anomalies due to measurements or to LCS errors. The values in the bounded area are the ones considered reliable. They are stored in a new X_{jr} time series also associated with the SmS. These data are then processed by the filtering step.

3.3. Filtering step

This step allows to attenuate the perturbations or the measurement noise specific to the LCS. The filtering using a kernel regression is applied to the X_{jr} time-series for curve smoothing. The use of Nadaraya-Watson Kernel Regression on these data is motivated by the fact that they are statistically non-parametric (measurements on environmental systems). Moreover, this approach is optimized for small numbers of points and suffers less from bias problems at the extreme points of the time-series (Nadaraya, 1964). Once the data are reliable and filtered, they are stored in a new time-series X_{jrf} associated with the corresponding SmS number and presented as inputs to the aggregation step.

3.4. Aggregation step

This step consists in applying a median voting algorithm on the reliable and filtered data obtained after the four previous steps. It consists of three tasks:

1. The values of the time-series X_{jrf} are concatenated into a vector of values named S_rf;
2. The values of S_rf are arranged in ascending order;
3. If the number of elements of S is odd, the $(n+1)/2$ element is selected for the output. If the number of elements of S is even then the average is calculated between the $n/2$ and $(n+1)/2$ elements.

The output of this voting algorithm is then stored in a time-series Out_{agg} with the timestamp equal to $t+\tau$, as its index. This output corresponds to the rolling window synthesis defined in the detection step. For the whole data, the Aggregator synthesis is the set of output values from the aggregation step stored in the Out_{agg} time-series.

These four subsections describe each step of the proposed method for processing SmS data concluded by the aggregation step. The measurement vectors (φi) constitute the inputs of the algorithm. After these steps, various outputs are produced: the time-series $Alert$ for alerts, the reliable synthesis of the data by the time-series Out_{agg}, and the time-series X_{jOut} to perform post-processing. The whole method leads to an algorithm implemented within the Agreggator which will be presented in the next subsection.

3.5. Algorithm description

The proposed Aggregator algorithm is implemented for three SmS transmitting measurement vectors φ1, φ2, and φ3 with their own period P. For clarity of presentation, the algorithm described hereafter will start after the data restructuring step and has as input the raw data from a single parameter φi.

3.5.1. Rolling windows

The $StartTime\_data$ and $EndTime\_data$ variables correspond to the start and end timestamps of the raw data. Figure 7 shows the main algorithm and, more precisely, the rolling window for the raw data. The Processing and Output steps are detailed in figure 8.

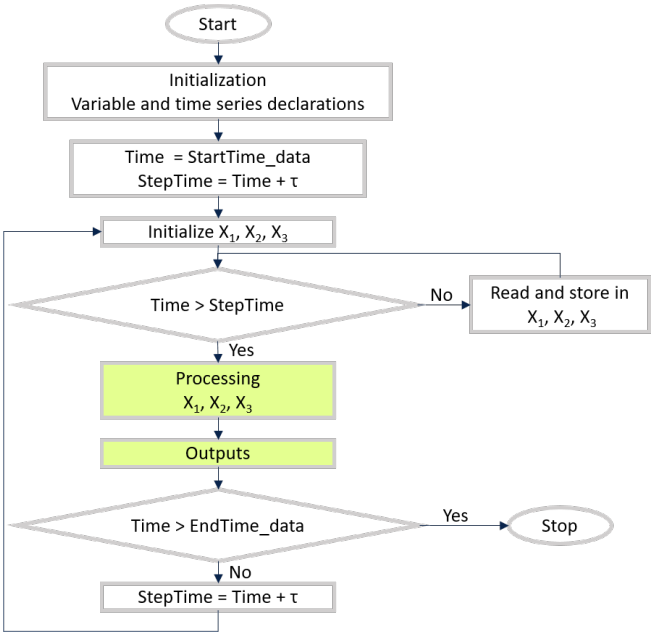


Figure 7. The main algorithm.

The initialization step consists in declaring the variables τ, Th and the time-series described in the method section. The time-series X_{jout}, $Alert$ and Out_{agg} are global and independent of the rolling window. They are used to store the Outputs when running the raw data with the Time variable. Then, the rolling window is initialized and its size is set to τ. The raw data is stored in the time-series X_1, X_2 and X_3 as long as the Time variable is less than τ. When the variable Time reaches the size of the window or the end of the raw data, the treatments on X_1, X_2 and X_3 are done and the outputs are stored. Finally, as long as the Time variable is less than the last Timestamps of the raw data, a new window is set and the times series X_1, X_2 and X_3 are reset to store data again.

3.5.2. Processing and output

This part of the algorithm starts from the second phase of the fault detection step with the data having been stored in X1, X2 and X3. The size of the series is evaluated to detect the lack of data. The alerts are stored in the $Alert$ time-series and the number of elements present is subtracted from the total number of series (3 in this application) for the confidence

index. Then, the remaining values are compared against the thresholds calculated with the median and the deviation (Th) defined upstream. The outliers are stored in $X - jout$ corresponding to the X_j timestamp. The values having passed this first filter are considered reliable and are stored in X_{jr}. They are then smoothed by using a Nadaraya-Watson estimator and stored in X_{jrf}. Finally, the smoothed values are concatenated to evaluate the median which will be saved in Out_{agg}. The algorithm is repeated for each rolling window.

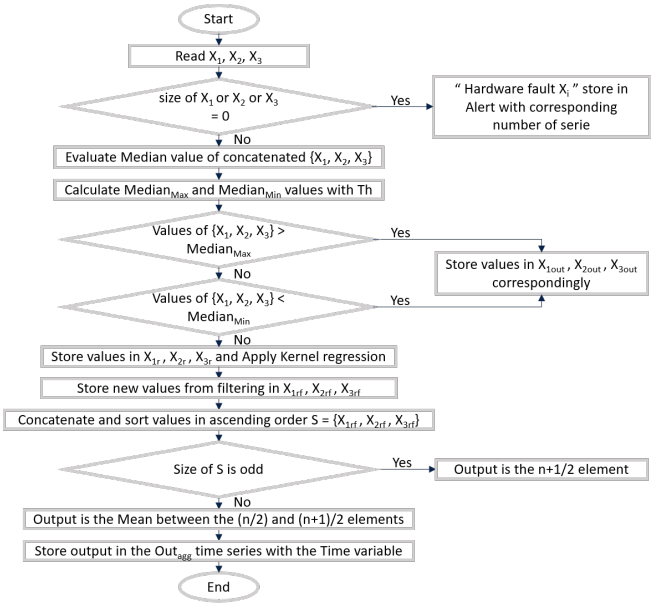


Figure 8. Processing and detailed output functions.

This subsection presents the method and the description of the algorithm. The data from the SmS are presented as input to the algorithm to be restructured by parameters. A rolling window whose size is defined upstream allows browsing the dataset. A detection module recovers the data in the first phase with a rolling window and then uses them to detect hardware faults and discard outliers in a second phase. The remaining data are filtered and presented to the voting algorithm which finalizes the synthesis for a time slot equal to the size of the rolling window.

4. RESULTS

The proposed algorithm is intended to work in real-time. To test it, the data corresponding to the measurements of PM2.5 concentration during one day are used (figure 3). The algorithm steps are programmed under Python and the code is provided in the Appendix section. The maximum period of the SmS is set to six minutes and the window size is ten minutes ($\tau = 10\ min$). The threshold value setting is defined according to the variability of the LCS observed on the data set. Initially, based on the uncertainty defined by the manufacturer, it is refined as observations are made until a threshold is set. Thus, the threshold deviation Th for the detection of aberrations is fixed at 30% for the LCS in charge of the PM2.5 measurements. Indeed, the starting value chosen is 10% based on the uncertainty defined by the manufacturer then it is adjusted to be able to make the first detections. The output of the algorithm using the dataset and based on the previous settings is presented in figure 9. Figure 10 shows the algorithm output when $\tau = 20\ min$ and $Th = 30\%$. The size of the rolling windows has low impact on the number of errors collected by the algorithm. Indeed, the smaller the step size, the more peaks are included in the synthesis of the aggregator. A larger step size will flatten the synthesis curve and increase the number of outliers. The comparison of the two figures shows that when the step size is large, the peaks deviate from the synthesis but the numbers of errors on SmS 1 and 2 are approximately identical. This factor makes it possible to conclude that the anomalies are due to the measured parameter because the errors are grouped in a small time interval. The errors of SmS 3 are more frequent than those of the two others and its higher number indicates a loss of reliability of the component by its return to zero. The confidence index of this LCS can be calculated from the ratio of error to the total number of measurements. The lifting of alerts on hardware faults can also be done on the size of the time-series. Indeed, when a SmS does not send any more its data the size of its time-series will decrease compared to the other SmS.

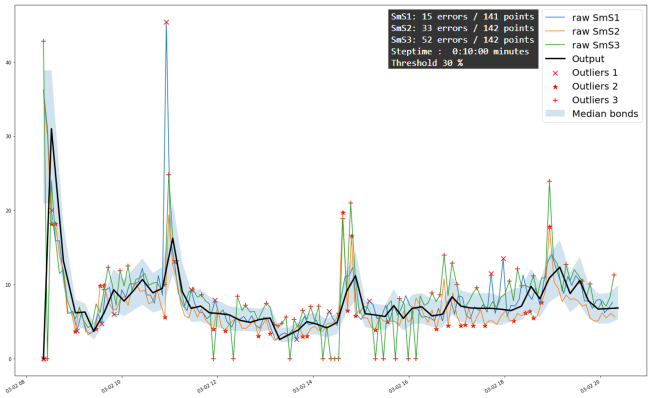


Figure 9. The algorithm output obtained on the raw data by using outliers detection with $\tau = 10\ min$ and $Th = 30\%$.

The parameter Th has a significant impact on the number of outliers. The comparisons between figures 9 and 11 and figures 10 and 12 confirm that the higher the parameter Th is, the more the deviations between measurements will be tolerated. However, this parameter can be decisive in detecting a possible offset of an LCS. Indeed, depending on the setting of Th, a high number of measurements may be detected as outliers compared to other LCS leading in a shift compared to all collected measurements.

For the synthesis of the Aggregator, the step size is an important factor. Depending on the objective of the measurement,

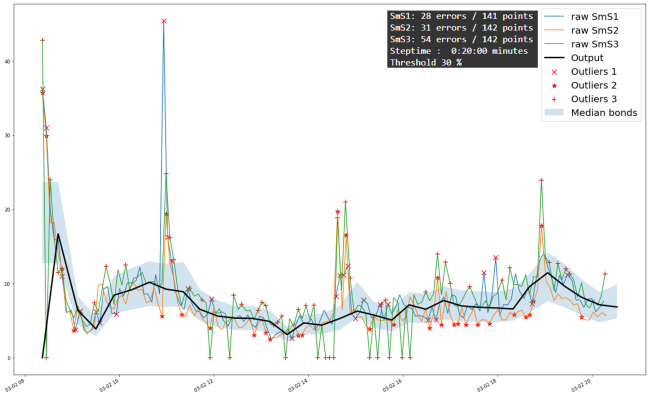


Figure 10. The algorithm output obtained on the raw data by using outliers detection with $\tau = 20\ min$ and $Th = 30\%$

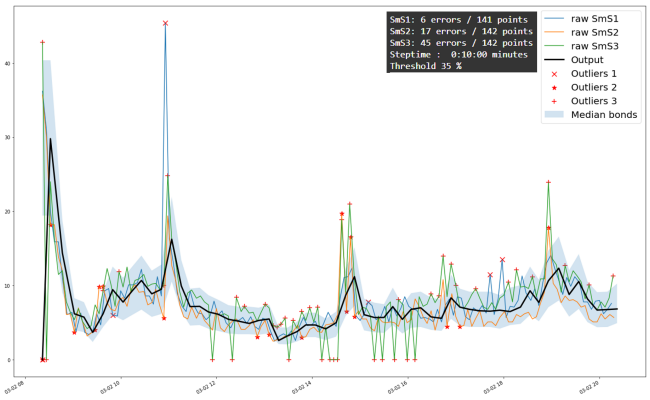


Figure 11. The algorithm output obtained on the raw data by using outliers detection with $\tau = 10\ min$ and $Th = 35\%$

it will be adapted either to avoid peak detection or to obtain a synthesis over a larger time range. This Aggregator's output is only slightly influenced by the τ parameter because of the robustness of the median on the extremes. The combination of the synthesis and the time-series outliers allows us to observe failure typologies at the hardware level but also at the air quality level. A confidence index can therefore be calculated from the number of LCS errors and the size of the input data.

5. Conclusion

In this paper an algorithm allowing to process pollutant concentration measurements provided by a measuring station has been proposed. It allows to have a reliable synthesis of the measurements at a given spacial point and to beyond the hardware faults of the LCS composing the SmS. The algorithm offers the possibility to adapt the purpose of measurement and the outliers detection by adjusting two parameters. The first parameter is the rolling window size τ and the second parameter is the tolerance threshold Th. The setting of Th impacts the outliers detection, which are given in a synthesis

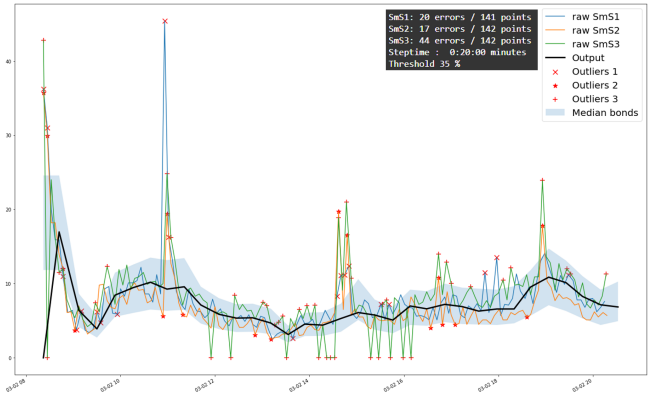


Figure 12. The algorithm output obtained on the raw data by using outliers detection with $\tau = 20\ min$ and $Th = 35\%$

to identify the topology of failures. Moreover, a confidence index based on the number of errors can be calculated and associated with the synthesis. Its purpose is to quickly identify the health state of the LCS in service and thus the state of the measuring station in general. Note that in this version of the algorithm, the Th and τ values are chosen empirically due to lack of long-term observation of the measuring station behavior. Therefore, as a future work, data collection through several seasons will allow to better set the detection threshold and to observe failure typologies. The stored outliers will be processed to differentiate the anomalies due to air pollution or to LCS. The classification of failures and their identification by experts will eventually allow the creation of an automatic failure identification module for preventive and predictive maintenance to be carried out on the measuring stations. The reliable data will be used to predict abnormal behaviors on a larger scale and thus to determine the air quality and eventually to predict future pollution to protect the population.

Acknowledgment

The authors would like to thank Rose-Marie GRENOUILLET in charge of the Environment and Serge DUTHU in charge of operations for their advice and for the setting up of the measuring station on the field.

This work is co-funded by Région Occitanie and Communauté de Communes Pyrénées Vallées des Gaves.

References

Bass, J., Latif-Shabgahi, G. R., & Bennett, S. (1997). *Experimental comparison of voting algorithms in cases of disagreement*. (Pages: 523) doi: 10.1109/EUROMIC.1997.617368

Castell, N., Dauge, F. R., Schneider, P., Vogt, M., Lerner, U., Fishbain, B., ... Bartonova, A. (2017, Febru-

ary). Can commercial low-cost sensor platforms contribute to air quality monitoring and exposure estimates? *Environment International*, *99*, 293–302. doi: 10.1016/j.envint.2016.12.007

Kassab, M., Hashad, A., Taha, H., & Shedied, S. (2013, May). A Novel Voting Algorithm Based on Weighted Average Voting with a Classification Technique. *International Conference on Aerospace Sciences and Aviation Technology*, *15*(AEROSPACE SCIENCES), 1–8. doi: 10.21608/asat.2013.22266

Kucera, P., Hyncica, O., Cidl, J., & Vasatko, J. (2006, February). Realibility model of TMR system with fault detection. *IFAC Proceedings Volumes*, *39*(21), 468–472. doi: 10.1016/S1474-6670(17)30233-1

Latif-Shabgahi, G. R. (2004, September). A novel algorithm for weighted average voting used in fault tolerant computing systems. *Microprocessors and Microsystems*, *28*(7), 357–361. doi: 10.1016/j.micpro.2004.02.006

Lewis, A. C., Lee, J. D., Edwards, P. M., Shaw, M. D., Evans, M. J., Moller, S. J., ... White, A. (2016, July). Evaluating the performance of low cost chemical sensors for air pollution research. *Faraday Discussions*, *189*(0), 85–103. (Publisher: The Royal Society of Chemistry) doi: 10.1039/C5FD00201J

Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013, July). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, *49*(4), 764–766. doi: 10.1016/j.jesp.2013.03.013

Lorczak, P., Caglayan, A., & Eckhardt, D. (1989, June). A theoretical investigation of generalized voters for redundant systems. In *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers* (pp. 444–451). doi: 10.1109/FTCS.1989.105617

Morawska, L., Thai, P. K., Liu, X., Asumadu-Sakyi, A., Ayoko, G., Bartonova, A., ... Williams, R. (2018, July). Applications of low-cost sensing technologies for air quality monitoring and exposure assessment: How far have they gone? *Environment International*, *116*, 286–299. doi: 10.1016/j.envint.2018.04.018

Muhammed, T., & Shaikh, R. A. (2017, January). An analysis of fault detection strategies in wireless sensor networks. *Journal of Network and Computer Applications*, *78*, 267–287. doi: 10.1016/j.jnca.2016.10.019

Nadaraya, E. A. (1964, January). On Estimating Regression. *Theory of Probability & Its Applications*, *9*(1), 141–142. (Publisher: Society for Industrial and Applied Mathematics) doi: 10.1137/1109020

Panda, M., & Khilar, P. M. (2015, February). Distributed self fault diagnosis algorithm for large scale wireless sensor networks using modified three sigma edit test. *Ad Hoc Networks*, *25*, 170–184. doi: 10.1016/j.adhoc.2014.10.006

BIOGRAPHIES

Sylvain Poupry was born in France in 1986. He received a Generalist Engineering Degree with Integrated Systems Design Option at Tarbes National School of Engineering (ENIT), in July 2019. From 2007 to 2013, he was a marine officer in energy and propulsion systems in "Marine Nationale", at Toulon, France. From 2013 to 2017 he was Automation Technician and Team Leader at "Societe Financiere Altela", in Semeac, France. In 2019, he made a reconversion to be an engineer, and currently, he is a second-year doctoral student within the Production Engineering Laboratory (LGP). The topic of his Ph.D. research is "Contribution to the design and implementation of a reflexive Cyber-Physical System: application to air quality prediction in the "vallées des gaves"". His current research interests are low-cost sensors, air pollution sensing, the internet of things, data science, and Prognostics and Health Management.

Cédrick Béler is assistant professor at ENIT (Ecole Nationale d'ingénieurs de Tarbes). His researches are conducted in the field of Social-Cyber-Physical System and Digital Twin and related to data science, knowledge management. He is especially interested in the way information is organized in distributed networks of information systems with human in the loop. Application are developed in the context of industry 4.0 but also in the public space (local, regional and national authorities).

Kamal Medjaher received the Engineering degree in electronics from Mouloud Mammeri University, Tizi Ouzou, Algeria, the M.S. degree in control and industrial computing from Ecole Centrale de Lille, Villeneuve-d'Ascq, France, in 2002, and the Ph.D. degree in control and industrial computing from University of Lille 1, Villeneuve-d'Ascq, France, in 2005. He was Associate Professor at the National Institute of Mechanics and Microtechnologies, Besançon, France, and FEMTO-ST Institute, from 2006 to 2016. He is currently Full Professor at Tarbes National School of Engineering (ENIT), France. He conducts his research activities within the Production Engineering Laboratory. His current research interests include prognostics and health management of industrial systems and predictive maintenance.

APPENDIX

Import librairy

```
import pandas as pd
from pathlib import Path
import os.path
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.nonparametric.kernel_regression import KernelReg
# For parsing dates
from datetime import datetime, timedelta
from dateutil.parser import parse
```

Import data

```
# Import data from GitHub
url = 'https://raw.githubusercontent.com/spoupry/spoupry/main/pm10_daily.csv'
# Read and use csv file
# Daily PM10
df = pd.read_csv(url, delimiter="\t")

# Import data from GitHub
# url = 'https://raw.githubusercontent.com/spoupry/spoupry/main/humidite.csv'
# Read and use csv file
# For french csv from online website
# df = pd.read_csv(url, delimiter=";",decimal=",") # replace , by .

print(df.head(10))
df.dtypes
```

Use database and parse data as columns index

```
# Parse date
df["DateTime"] = pd.to_datetime(df["DateTime"])
# df.head(2)
# Initialization dataset
# Setting the timestamp as dataframe's index.
df.set_index('DateTime', inplace=True) # DateTime is now the index.
print(df.head(10))
df.dtypes
```

Initialisation

```
# Initialization of the variables
StartTime_df = df.index[0]  # Starting timestamp dataset
EndTime_df = df.index[-1]   # End timestamp dataset
tau = timedelta(minutes=10) # step size for rolling window
StepTime = StartTime_df + tau  # Time slice
Threshold = 35 #  35% bonds for outliers
# Time-series for raw datas for graphics
X1raw = pd.Series( [0] , index = [StartTime_df], dtype=float)
X2raw = pd.Series( [0] , index = [StartTime_df], dtype=float)
X3raw = pd.Series( [0] , index = [StartTime_df], dtype=float)
# print("Start df",StartTime_df,"End df",EndTime_df,"step",StepTime)
# Initialise global time-series (4)
# Median for row datas
Median=pd.DataFrame(index=[StartTime_df],columns=['Median','MedMax','MedMin'])
# Output time-series after median voter
Outagg = pd.Series([0], index = [StartTime_df], dtype=float)
# Alert time-series from no data detection
Alert = pd.Series( ['Begin'] , index = [StartTime_df])
# Outliers time-series of corresponding SmS
X1out = pd.Series( [0] , index = [StartTime_df], dtype=float)
X2out = pd.Series( [0] , index = [StartTime_df], dtype=float)
X3out = pd.Series( [0] , index = [StartTime_df], dtype=float)
# Reliable time-series extracted from Median
X1r = pd.Series( [0] , index = [StartTime_df], dtype=float)
X2r = pd.Series( [0] , index = [StartTime_df], dtype=float)
X3r = pd.Series( [0] , index = [StartTime_df], dtype=float)
```

Functions

```
# Calculate the median of a time-series
# Input : time-series
# Output the median as a float or None if list =[]
def calculate_median(l):
    l = sorted(l)
    l_len = len(l)
    if l_len < 1:
        return None
    if l_len % 2 == 0 :
        return ( l[(l_len)//2] + l[(l_len//2)-1] ) / 2.0
    else:
        return l[(l_len-1)//2]
```

```
# Dectect Hardware fault
# Input: 3 time-series of a slice
# No Output
# Store data in Alert global time-series
def detect_no_data(X1,X2,X3):
    # Detection of no data
    # print("Start detection hardware fault")
    HardwareFault=[]
    # print("size of time-series X1,X2,X3 =",len(X1),len(X2),len(X3))
    if (len(X1) == 0):
      # print("Hardware fault SmS1")
      HardwareFault.append("Hardware Fault SmS1")
    if (len(X2) == 0):
      # print("Hardware fault SmS2")
      HardwareFault.append("Hardware Fault SmS2")
    if (len(X3) == 0):
      # print("Hardware fault SmS3")
      HardwareFault.append("Hardware Fault SmS3")
    if (len(HardwareFault) == 0):
      print("All sensors is ok")
    else:
      print(HardwareFault," Ic= ", 3 - len(Alert.loc[StepTime]),"/3")
    # Store Hardware fault in Alert time-series
    Alert.loc[StepTime] = HardwareFault
    # print("End detection hardware fault")
```

```
[ ]  # Dectect outlier from median and threshold
     # Input: 3 time-series of a slice and a concatenated list of 3
     # Output 3 time-series Xrl of reliable values for filtering
     #  Store in Xr global time-series for later processing
     def detect_outliers(X1,X2,X3,Concat):
         # print("Start ouliers detection &  Store")
         # Evaluate median
         Median_slice = calculate_median(Concat)
         # Calculate MedMax and MedMin for outlier detection
         # print(calculate_median(Concat))
         MedMin = Median_slice - (Median_slice*Threshold)/100
         MedMax = Median_slice + (Median_slice*Threshold)/100
         #  Store median for plotting in Median time-series
         Median.loc[StepTime,'Median'] = Median_slice
         Median.loc[StepTime,'MedMin'] = MedMin
         Median.loc[StepTime,'MedMax'] = MedMax
         # print("Median",Median_slice,"MedMin",MedMin,"MedMax",MedMax)
         # Outputs function: X reliable data time-series
         X1rl = []
         X2rl = []
         X3rl = []
         idx_1rl= []
         idx_2rl= []
         idx_3rl= []
         # Outliers and reliable data of X1
         if(len(X1) != 0):
           for time, row in X1.iteritems():
             if((row > MedMax) or (row < MedMin)):
               X1out.loc[time] = row
             else:
               # Store in global X1r time-series for plotting
               X1r.loc[time] = row
               X1rl.append(row)
               idx_1rl.append(time)
         # print(X1)
         # print(X1out)
         # print(X1r)
         # Outliers and reliable data of X2
         if(len(X2) != 0):
           for time, row in X2.iteritems():
               if((row > MedMax) or (row < MedMin)):
                 X2out.loc[time] = row
               else:
                 # Store in global X1r time-series for plotting
                 X2r.loc[time] = row
                 X2rl.append(row)
                 idx_2rl.append(time)
         # print(X2)
         # print(X2out)
         # print(X2r)
         # Outliers and reliable data of X3
         if(len(X3) != 0):
           for time, row in X3.iteritems():
             if((row > MedMax) or (row < MedMin)):
               X3out.loc[time] = row
             else:
               # Store in global X1r time-series for plotting
               X3r.loc[time] = row
               X3rl.append(row)
               idx_3rl.append(time)
         # print(X3)
         # print(X3out)
         # print(X3r)
         # Construct time-series from lists
         X1rl = pd.Series(X1rl, index = idx_1rl, dtype=float)
         X2rl = pd.Series(X2rl, index = idx_2rl, dtype=float)
         X3rl = pd.Series(X3rl, index = idx_3rl, dtype=float)
         # print("End outliers detection & store")
         return X1rl,X2rl,X3rl
```

```
[ ]  # Smoothing curve by applying kerbel regression
     # Input: 3 time-series Xrl of reliable values
     # Output 3 time-series Xf filtered for median voter
     def filtering(X1,X2,X3):
       from numpy import linspace
       # print("Start filtering")
       # X1
       # Must have two points minimum
       if(len(X1) == 0 or len(X1) == 1):
         X1f = X1
         print("not enough data")
       else:
         X1_ = X1.copy()
         X1_.index = linspace(1., len(X1_),len(X1_))
         kr = KernelReg([X1_.values],[X1_.index.values], var_type='c')
         f1 = kr.fit([X1_.index.values])
         X1f = pd.Series(data=f1[0], index = X1.index, dtype=float)
       # X2
       # Must have two points minimum
       if(len(X2) == 0 or len(X2) == 1):
         X2f = X2
         print("not enough data")
       else:
         X2_ = X2.copy()
         X2_.index = linspace(1., len(X2_),len(X2_))
         kr = KernelReg([X2_.values],[X2_.index.values], var_type='c')
         f2 = kr.fit([X2_.index.values])
         X2f = pd.Series(data=f2[0], index = X2.index, dtype=float)
       # X3
       # Must have two points minimum
       if(len(X3) == 0 or len(X3) ==1):
         X3f = X3
         print("not enough data")
       else:
         X3_ = X3.copy()
         X3_.index = linspace(1., len(X3_),len(X3_))
         kr = KernelReg([X3_.values],[X3_.index.values], var_type='c')
         f3 = kr.fit([X3_.index.values])
         X3f = pd.Series(data=f3[0], index = X3.index, dtype=float)
       # print("End filtering")
       # Return filtered data time-series
       return X1f,X2f,X3f
```

```
[ ]  # Algorithm of the median voter
     # Input : 3 time-series reliable and filtered
     # Output the synthesis as a float or None if list =[]
     def Median_voter(X1,X2,X3):
       # print("Start Median Voter")
       concatenated_series=pd.concat([X1,X2,X3])
       # print(concatenated_series)
       Output = calculate_median(concatenated_series)
       # print("Output=",Output)
       # print("End Median Voter")
       # Return synthesis of median voter
       return Output
```

```
[ ] # Create X time-series in the rolling window in a tau size from dataset
    # Input: a slice of dataset
    # Store in global series Outagg, Xraw
    def processing(df):
        print("Start processing slice")
        #create X1,X2,X3 series
        X1 = []
        X2 = []
        X3 = []
        idx_1= []       # lists for Xi time-series creation
        idx_2= []
        idx_3= []
        for time, row in df.iterrows():
          if(pd.isna(row[df.columns[0]]) == False ): # NaN detection
            X1.append(row[df.columns[0]])            # stock in x1 list
            idx_1.append(time)
            X1raw.loc[time] = row[df.columns[0]]     # input in X1raw time-series
          if(pd.isna(row[df.columns[1]]) == False ):
            X2.append(row[df.columns[1]])
            idx_2.append(time)
            X2raw.loc[time] = row[df.columns[1]]     # input in X2raw time-series
          if(pd.isna(row[df.columns[2]]) == False ):
            X3.append(row[df.columns[2]])
            idx_3.append(time)
            X3raw.loc[time] = row[df.columns[2]]     # input in X3raw time-series
        # print(X1,X2,X3)
        # Concatenation of all elements of the rolling windows
        Concat = X1 + X2 + X3
        # Create time series
        X1 = pd.Series(X1, index = idx_1, dtype=float)
        X2 = pd.Series(X2, index = idx_2, dtype=float)
        X3 = pd.Series(X3, index = idx_3, dtype=float)
        # print("X1,X2 and X3 created")
        # Function detection hardware fault
        detect_no_data(X1,X2,X3)
        # Function detection outliers + stock values in Xout time-series
        Xreliable = detect_outliers(X1,X2,X3,Concat) # return reliable data (Xr)
        # print(Xreliable)
        # Function kernel regression return filtered data (Xrf)
        Xfiltered = filtering(Xreliable[0],Xreliable[1],Xreliable[2])
        # Function Median voter
        Output = Median_voter(Xfiltered[0],Xfiltered[1],Xfiltered[2])
        # Store ouput in the Outagg time-series
        if(Output == None):
          print("oh la la")
        else:
          Outagg.loc[StepTime] = Output
        #return X1,X2,X3
```

Main algorithm

First slice of database

```
[ ] # Initialise the first slice by reading first Timestamps from dataset
    # Create the firts slice of dataset for processing
    # First slice of time from df dataframe
    StepTime = StartTime_df + tau
    Time_slice_list=[StartTime_df ]
    # print("Start: ", StartTime_df, " , End: ", StepTime)
    # Identificate the first slice
    df_slice = df.loc[StartTime_df:StepTime].copy()
    # Processing slice
    processing(df_slice)
    Median.loc[StartTime_df] = Median.loc[StepTime]
    # print(Median)
```

Others slices until the end of database

```
[ ] # Rolling window throught the dataset
    # Other slices
    for time, row in df.iterrows():
      if(time <= StepTime):
        This_time = time
        #print(time)
      if(time > StepTime):
        Start_step = time
        StepTime = time + tau
        Time_slice_list.append(Start_step)
        # Time_slice_list.append(StepTime)
        # print("Start: ", Start_step, " , End: ", StepTime)
        df_slice = df.loc[Start_step:StepTime].copy()
        processing(df_slice)
    #Time_slice_list
```

Outputs and graphics

```
[ ] # Sizes of series for confidence index
    print('SmS1:',len(X1out),'errors /',len(X1raw),'points')
    print('SmS2:',len(X2out),'errors /',len(X2raw),'points')
    print('SmS3:',len(X3out),'errors /',len(X3raw),'points')
    print('Steptime : ', tau , 'minutes')
    print('Threshold', Threshold , '%')
    # Plotting time-series

    plt.figure(figsize=(24,16))
    plt.scatter(X1out.index, X1out.values,label='Outliers 1', s=100, c='red' ,marker = 'x')
    plt.scatter(X2out.index, X2out.values,label='Outliers 2', s=100, c='red' ,marker = '*')
    plt.scatter(X3out.index, X3out.values,label='Outliers 3', s=100, c='red' ,marker = '+')
    X1raw.plot(label='raw SmS1')
    X2raw.plot(label='raw SmS2')
    X3raw.plot(label='raw SmS3')
    plt.fill_between(Median.index,Median['MedMax'],Median['MedMin'],alpha = 0.2, label='Median bonds')
    Outagg.plot(label='Output', linewidth = 3, c='black' )
    # t=10
    # plt.title('Th=50 and tau=%i' %t)
    plt.legend(fontsize=20)
    plt.show()
```