UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE
INFORMÁTICA

DATA QUALITY AND DEPENDABILITY OF IOT PLATFORM FOR BUILDINGS ENERGY ASSESSMENT

João Miguel Mota Cláudio Valente

Mestrado em Informática

Dissertação orientada por:
Prof. Doutor Pedro Miguel Frazão Fernandes Ferreira
e co-orientado por Prof. Doutor António Casimíro Ferreira da
Costa

2021

Agradecimentos

Desde o momento que ingressei na faculdade que o trajecto se apresentou atribulado. Houve momentos de incerteza e desespero, e apesar de todos os resultados em contrário, tive sempre pessoas com quem podia contar e que me apoiaram quando mais precisei. Mantiveram a confiança em mim, hoje dou-lhes razão.

Deixo um enorme agradecimento à minha familia e amigos que sempre lutaram por mim, nos bons momentos, e especialmente nos maus momentos. Fizeram me acreditar que ultrapassaria todos os obstáculos que me foram postos, e estarei para sempre mais do que grato.

Pedro Alves, já estavas incluído, mas mereces um destaque, não só pelo apoio, mas também por uma sessão de conversa longa, que acabou por desbloquear o desenvolvimento do projecto, um muito obrigado, e porte-se bem.

Quero deixar um agradecimento especial à professora do departamento de Matemática, Maria Antónia Lopes Duffner Bessa Monteiro, que apesar não ter consciência da situação, em duas frases, e uma ação ao ínicio de uma aula, teve um impacto enorme até ao dia de hoje, e muito provavelmente, em todos os anos que virão. Um muito, muito obrigado...

Um agradecimento a todos os que me acompanharam no desenvolvimento do projecto, aos meus orientadores Professor António Casimiro, e Professor Pedro Ferreira. Ao Professor José Cecílio, e ao Professor Vinicius Cogo. E a todos os meus colegas presentes no projecto SATO.

Resumo

Esta dissertação foca-se no desenvolvimento de uma framework capaz de avaliar a qualidade dos dados externos que são recolhidos, e processados em tempo real. Fazendo parte do projecto SATO (Self Assessment Towards Optimization), um projecto que ambiciona integrar sistemas de gestão de dados na cloud, e computação de recursos obtidos através ferramentas BIM(Building Information Modelling), sensores IoT e dispositivos presentes num determinado edifício, de forma a obter uma plataforma de avaliação e otimização energética.

Um objectivo para a implementação do projecto SATO é integrar e desenvolver uma SEF(Self Assessment Framework) que usa análise de dados e machine learning para reportar performance energética, comportamento e ocupação do edifício, e falhas no equipamento. Como tal, é neste contexto que está inserido o desenvolvimento de uma framework de dependabilidade e qualidade de dados.

No âmbito das novas plataformas de IoT(Internet of Things), existe uma correlação entre a eficácia e eficiência da plataforma, com a qualidade dos dados disponíveis para processamento e tomada de decisões. O processo torna-se de extrema importância, quando nos deparamos com um cenário de recolha de dados, recolhidos de diferentes tipos de sensores, presentes em vários dispositivos que se encontram distribuídos ao longo do edifício, e cujos dados serão essenciais para o controlo das atuações.

Tendo em vista os objectivos da plataforma SATO, este projecto focou-se na construção de uma framework adaptável que seja capaz de assegurar aspectos de dependabilidade e qualidades de dados, como tal, deve ser capaz de conter todas as funcionalidades necessárias.

Para que a framework consiga cumprir estes propósitos, a metodologia ANNODE foi adotada. ANNODE é uma metodologia que visa a deteção de outliers, verificação de qualidade de cada uma das medidas recebidas, e substituição de valores que não obedeçam a limites mínimos de qualidade. Cada uma destas medidas está sujeita a estimativas, com o intuito de verificar a proximidade destas novas entradas, ao que seria esperado de uma progressão de valores num determinado ambiente.

Para sustentar a metodologia ANNODE, foi criada uma framework de estrutura centralizada capaz de gerir várias funcionalidades distintas. Devido a abstrações criadas e à forma como a comunicação intraprocesso se dá, a integração de novas funcionalidades

torna-se fácil e facilmente monitorizável. Sendo que esta entidade gestora tem acesso a cada uma das funcionalidades, e aos produtos de cada uma delas, a monitorização de cada um dos flows inseridos na framework, dá-se também em tempo real, e são transmitidos para a plataforma SATO.

Após analisar a metodologia ANNODE, e os objectivos inerentes à construção da plataforma SATO, definiu-se que a framework de qualidade de dados ficasse dividida nas seguintes atividades: Recepção de dados e análise de factores de qualidade, Deteção de outliers e avaliação da qualidade, Construção de modelos de redes neuronais, e por fim, a Unidade de monitorização.

Após a recepção de um registo na framework, antes deste ser inserido no sistema, primeiro cada secção desse registo será verificado, para averiguar se obedece aos padrões conhecidos daquele tipo de dados, depois de passar a análise representacional, segue para o componente do sistema responsável pela deteção de outliers.

Para que cada registo sujeite a deteção de outliers, são criados modelos de redes neuronais Keras a partir de uma configuração com as especificidades pretendidas para o modelo. Esta configuração é recebida no sistema, e encaminhada para uma Process Pool, que é responsável por gerir a construção dos vários modelos que dão entrada no sistema, mas também por verificar se a configuração obedece aos requisitos necessários para que seja processada. Esta unidade confere a possibilidade de criação de múltiplos modelos em simultâneo, que podem ser integrados no sistema.

Cada registo que entra na unidade de deteção de outliers, é sujeito a várias estimativas, utilizando conjuntos distintos de valores recolhidos previamente, durante a execução do sistema. Através destas estimativas e de quanto distam do valor registado, é possível verificar com um determinado grau de incerteza, se estamos perante um outlier, ou uma ocorrência comum no sistema. Após a identificação de um outlier, dá-se a substituição do valor registado, caso não exista a identificação de um outlier, é atribuído um coeficiente de qualidade à medição em análise. Independentemente do caso, a medição é inserida no sistema para ser utilizada em análises de registos futuros.

Sendo que a framework desenvolvida, faz parte de uma plataforma que está em constante comunicação com os vários componentes, necessita de transmitir informações sobre as circunstâncias em que se está a dar o processamento, e os resultados de cada processamento, como tal, foi implementada uma estrutura que regista o tempo de processamento de cada secção do software, que permite monitorizar a execução no programa em tempo real. Ao mesmo tempo, a presença de objectos que se encontram presentes desde o momento da chegada de um registo, até ao fim do seu processamento, levam ao reporte de condições associadas ao registo e ao seu processamento, e modificações que foram feitas, nas etapas críticas do processo.

Os pontos fulcrais deste projecto detêm-se com a implementação de conceitos e noções de dependabilidade, numa framework já direcionada para a deteção de outliers e mitigação

de falhas, e em que devido às condições impostas pelo ambiente, performance e monitorização das operações se tornam fundamentais, e portanto, extremamente importantes.

Palavras-chave: Internet das Coisas, qualidade dos dados, deteção de falhas, mitigação de falhas, IoT

Abstract

In the domain of IoT(Internet of Things), there is a correlation between the efficiency and effectiveness of a platform, and the quality of the data available for processing and decision making. When a scenario of data collection from different types of sensors, integrated into several devices that are distributed alongside the building, is presented. And, considering that data is essential to monitor and control the level of actuation in the building, the process becomes a matter of extreme importance.

To achieve this, ANNODE's methodology is adopted for quality verification of each measurement that arrives to the framework. The construction and utilization of neural networks are crucial points of this project, as well as, the correct application of dependability concepts, and notions of efficiency in processes of outlier detection and fault mitigation.

Keywords: Internet of Things, data quality, fault detection, fault mitigation, IoT

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's world, there is an increasingly higher need to control resources in a way that allows a thoughtful consumption, and try to reduce as much waste as possible. Alongside with this mindset, there has also been a recurrent trend in trying to adapt traditional common appliances and technology to a large system that comprehends all the information regarding all those appliances, in order to control, manage and apply different types of resources so human, energetic and structural demands are met.

Internet of Things is the term used for the technological branch responsible for the development of techniques that explore data acquisition on a real-world environment, analyzing that same data and inferring the state of the surrounding environment, with the objective of deciding a valid approach to control or affect the system.

In SATO (Self Assessment Towards Optimization) project, the main goal is to assess the energy consumption of each appliance contained in a building, as well as the energy consumption of the overall building, so it can improve the energetic efficiency of a building, by coordinating the resources during execution.

To achieve such coordination, data-based methods are utilized to interpret sensor's readings and infer the overall energetic state of the building, and to proceed to follow the best course of action.

As part of the SATO project, this thesis will address the quality of the data that arrives to a pre-processing unit, and the mechanisms of fault detection, fault classification and fault mitigation.

1.1 Motivation

From the sensors placed on all electrical outputs, by means of communication protocols, data is sent to another system's component that is responsible for cleaning the data, and to process it, to facilitate and upgrade the process of decision making based on the data that it receives. At the same time, the data received doesn't always obey all quality standards, as it may appear with some wrong or nonexistent values, some records missing, or some different aspects, that will affect the performance of all algorithms and system's actions that depend on this data.

Since the algorithms lose valuable properties when they are trained and implemented in a poor quality data environment, the probability of making a good decision, of getting a good reading from that, decreases. As we are to optimize the efficiency of the system, and the data is shared amongst various components, every dataset should be interpreted, and "improved" before being used to create systems dynamics, or any decision regarding the system.

Furthermore, there might emerge a major issue derived from a decision, if it was based on inaccurate data, faults that went unnoticed, faults that had longer term consequences. The need of fault diagnosis and a strategy to handle those faults, is essential to an efficient and reliable system.

In the context of SATO(Self Assessment Towards Optimization) project, where a platform has as one of the main objective of having full knowledge of all the appliances, so it can regulate power supply based on power consumption, data quality is a major concern.

To achieve optimal control in smart systems, good quality data and fault mitigation strategies should be implemented.

1.2 Objectives

This thesis is part of the SATO project which intends to implement a platform, that enables the assessment and optimization of energy consuming equipment, in a given building. It was placed in the work package WP3, with the task T3.2, and is described as development of data quality and sensor/device failure assessments for fault tolerance.

The purpose of this work is to correctly develop a system capable of detect faults and data quality issues from the data received from the sensors placed inside the building. The main objectives of my work is to correctly develop a system capable of detect faults and data quality issues from the data received from the perceptions of the sensors placed in the building. A complementary objective is to mitigate the effects of the faults that naturally occur in the system, and use predictions to make a good assumption of what a real value would look like.

The research will be done according to the following steps:

- Ensure, assess and evaluate data quality standards in a given dataset
- To study, apply and analyze different approaches and techniques in fault detection
- Develop fault mitigation component in case of faults, or missing data
- Implement a User Interface Dashboard, and Data Visualization methods to overview the whole process

1.3 Contributions

The main contribution of this work is the implementation of a system that enables quality evaluation, fault detection and fault mitigation for the SATO project.

1.4 Document Structure

The remaining sections of this document are organised as follows:

- Chapter 2 - ”State of Art”
- Chapter 3 - ”Context and Design Decisions”
- Chapter 4 - ”Implementation”
- Chapter 5 - ”Results”
- Chapter 6 - ”Conclusion and Future Work”

Chapter 2

State of Art

SATO Quality was thought out as continuation of a previous implemented project, where the focus was to ensure good quality results over certain datasets, the project that served as a starting point for SATO Quality is the ANNODE project [16],[24]. The ANNODE project created a methodology that is designed as an ensemble of supervised learning methods, which required an offline initial training phase for each MLP model construction. Furthermore, the methodology contemplates 4 more blocks, all of them capable of performing in runtime, and for each new received measurement from each target sensor. The blocks are the following:

- Prediction (P) – When a new measurement is received, its quality must be assessed. Since, the true environmental value of the measurement is unknown, through prediction methods, one or more estimates of that ground truth are obtained to be used in subsequent processing blocks with the objective of evaluating the quality and determining a replacement value with better quality.

- Failure Detection (FD) – Intends to identify possible failure behaviors in the dataset. By characterizing a measurement as normal or abnormal, can determine the existence of a failure situation. However it should take in account apparent anomalies cause by real environmental events, thus not signaling the measurement as faulty.

- Quality Evaluation (QE) – Using the outcome of the previous blocks, a quality coefficient for the measurement can be determined. If a measurement is considered faulty, this coefficient is set to 0. Otherwise, it will take a value that may be at most 1.

- Measurement Reassessment (MR) – This block aims to mitigate the detected fault by determining an estimate of the expected value of the measurement with sufficiently good quality. If a measurement is faulty, it should not be used further, as it can affect estimates, and degrade the quality evaluation of future measurements.

At the end of this project, some guidelines were given by [16], and [24] in order to improve the framework, for further utilization:

- Evaluation of the proposed dependability-oriented methodology and developed strategies for prediction, failure detection and measurement reassessment in other scenarios featuring different datasets (containing monitored variables with different characteristics), and considering other types of failures.
- Development of a software tool to support an easy instantiation of the methodology to already deployed and working sensor networks.
- Study of the feasibility and performance of strategies based on different machine learning techniques, namely for the Prediction and the Failure Detection blocks.
- Generalization of the Failure Detection strategies to any number of nodes in the sensor network.
- Definition of improved strategies for implementing the Quality Evaluation block.

2.1 Data quality

Data quality is defined as a qualitative assessment of data. In other words, is the degree of satisfaction that a given dataset presents, relative to the purpose to which such data will be used. As it cannot be defined as a simple concept, it has to be decomposed in several categories, which do change within literature.

To achieve the most understanding, of data quality of a system, particularly in this project, data quality is described according to four categories[27]:

- Intrinsic: *Category that characterizes data itself.(The level of completeness of the data set, the degree of which the data is accurate)*
- Contextual: *Category that concerns with the external state surrounding the production of data*
- Representational: *Category responsible for the format and structure in which data is presented*
- Accessibility: *Category that describes the difficulty of accessing the data*

Each category aggregates specific dimensions, which will serve as guidelines for what it's going to be further implemented, and also transformed to quantitative measurement to evaluate each data set.

2.1.1 Intrinsic

Accuracy

Data Accuracy is one of the most important components to understand when talking about quality of data. It refers to the correctness of data values stored in an object. Another explanation is the degree to which values truthfully represent real world values. In order to have a qualitative assessment of a dataset, the following equation [27] is used:

$$q_{ac}(p_i) = \frac{1}{m}\sum_{j=1}^{m}((1 - \frac{n_{o_j}}{n_{p_j}})w_{o_i} + (1 - e_{sp_i})w_{e_i}) \tag{2.1}$$

$$q_{ac}(p_{DB}) = \frac{1}{n}\sum_{j=1}^{m}(q_{ac}(p_i) \tag{2.2}$$

Where n_{p_j} is the number of entries, n_{o_j} is the number of outliers, in the entries of p_i in the data set ds_j and e_{s_p} is the maximum specified error for the same parameter. w_{o_i} and w_{e_i} are weights, that follow the constraint:

$$w_{o_i} + w_{e_i} = 1$$

The task to ensure the accuracy isn't as simple or straightforward as wished. Inconsistencies can occur at any given time, due to direct changes in system's recording process or granularity, or even real world factors, like hardware degradation or malfunctions.

Issues regarding elements of a dataset are common. Each element of a data is composed by information coming from different sources, alongside with the possibility of an element not be recorded, there's also the very real possibility that some values won't be recorded. The absence of a value may induce an incorrect decision, and consequently creating a value when it's not necessary. Other faults that can occur are data outliers. Data outliers are present when there's a value that is illogically distant compared to the average values over periods of time.

Consistency

Consistency is the dimension in which validity and integrity are evaluated. Data is said to be consistent if there are no conflicts between the values, or if there at most, one single object/value for every instance that was measured. ([8] [27])

It can have many implementations as it depends on the rules considered, and they depend on the business, and field in which is going to be implemented, also depends on the features that are retrieved during data collection. However, to classify the dataset we will use a broad term n_{c_j}, referring to the number of conflicting instances, n_{p_j} the number of entries, in the partition p_i of ds_j.

$$q_{cons}(DS) = \frac{1}{m}\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(1 - \frac{n_{c_j}}{n_{p_j}}) \tag{2.3}$$

Completeness

Dimension that measures the plenitude of values within a dataset. A dataset would be considered incomplete if there's null values. To qualify the completeness of a dataset, the following calculation is produced:

$$q_{com}(DS) = \frac{1}{m}\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(1 - \frac{n_{m_j}}{n_{p_j}}) \tag{2.4}$$

Where n_{p_j} is the number of entries, and n_{m_j} is the number of missing entries, in the partition p_i of ds_j.

To mitigate the lack of a single value, there are techniques that use past and future values to come up with a more or less precise value to replace the missing value. Regression strategies are a reliable solution for these situations. Depending on how many values are missing, other strategies should be used. [10] As errors accumulate through time, over some period the error could be of such an high order, that strategies that use historical patterns may be more reliable.

2.1.2 Contextual

Timeliness

Timeliness refers to features that are strongly related to time, other way to interpret it is, the degree to which data are up-to-date.

Timeliness analysis has to be done according two main dimensions [29]:

- Currency - How distant in time is the value to the record's last update.
- Volatility - How fast is the collecting new data, when working. It can be measured as the frequency of updates in the system, or the time period length between two updates.

Some issues that can occur in this context are, skewed timestamps. Those errors might just present themselves because later data attribution between different devices, however when analyzing, can have as consequence wrong computations. Comparing data from different devices can be a strategy to detect and correct timing issues.

Measurement Rate

Measurement Rate can be defined as the rate at which deployed hardware is sampled. Although some authors, interpret this dimension as a qualitative measurement, in which, measurement rate arrives from a formula that uses comparison real-sampling and desired-sampling ([27]). Personally, find it to be more useful, if used as a quantitative value that derives from the formula:

$$q_{Ms}(DS) = \frac{1}{m}\sum_{j=1}^{m}(I(j)) \tag{2.5}$$

Where $I(j)$ is interval of time, j, in seconds, that takes the system to measure two different entities are sampled.

Appropriate Amount of Data

The amount of data refers to the degree to which the quantity and volume of data is enough to deliver a reliable result of a given task. It will be calculated based on:

$$q_{AA} = \frac{1}{n}\frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{m}(1 - \frac{n_{p_j}}{n_{d_j}}) \tag{2.6}$$

Where n_{p_j} refers to the number of fields that have a valid value in entity j, n_{d_j} is the desired number of valid fields in entity j.

2.1.3 Representational

Interpretability

Dimension that considers the appropriateness of the notation used to represent information. Occasionally data values are inserted under a wrong field, or with characters that either are not supported, or do not conform with rules of representation.

Representational Consistency

Degree to which the format and structure conforms to either declared standards, or previously stored values. It concerns with homogeneity throughout the dataset.

2.1.4 Accessibility

Availability

Availability can be addressed as the capability of the data to be available, with a state such, that allows the utilization. Implies the ability to perform within boundaries quantified to meet what's considered an acceptable performance.

To measure accessibility, are used performance measurements such as:

- MTBF - Mean Time between failures

$$MTBF = \sum \frac{\Delta Uptime}{number\ of\ failures}$$

- MDT - Mean downtime

$$MDT = \sum \frac{\Delta Downtime}{number\ of\ failures}$$

- MTTR - Mean time to repair

$$MTTR = \sum \frac{Mean\ Time\ to\ Repair}{number\ of\ failures}$$

Security

The extent to which data is protected against unpermitted accesses, or write operations, and if the data that was received passes the verification tests. Data points in smart infrastructures are prone to attacks, usually exploring authentication issues, but can also be, false data injections attacks that provide data injections as real recorded data.

2.2 Fault Detection

Faults can be described as unintended, unwanted deviations of a system's property. Consequently, data records related to that property will present values that are further from what is considered a standard, or an acceptable condition. Faults may present themselves in two categories:

- Failure - Permanent interruption of a system's ability to perform a required function under some specified operating conditions. [14]
- Malfunction - Irregularity that presents itself with an intermittent behaviour while a system's operating.

When implementing automation based on data, there is the need to achieve high levels of data quality so it can be utilized for precise decisions throughout the system. To ensure data quality, there are some areas where we have to intervene so that information can be processed accordingly, and so that errors can be detected and treated accordingly. Error propagation in decision making process must be avoided.

Fault detection is responsible for determining the occurrence of fault in a monitored system. Takes advantage of inputs from the different processes, actuators and sensors, in a system, and using its dependencies, can infer if there is a fault present, and in some cases, inform the location of such fault, and how to solve it.

To produce that knowledge of the system, the majority of detection systems follows one of three main approach[4]:

- Models that use communication protocols to perceive when some component stops working correctly.
- Models that use data analysis, to infer if there was an fault occurrence in the system.
- Models that study the workflow of the system at each step, by the use of a model that's parallel to the system tries to predict and make some type of estimations.

2.2.1 Network-level Approach

Network approaches are effective when trying to find failures in the system, but lack the granularity to be able to detect malfunctions on the system. Because of it, is efficient to quickly detect serious issues on the system, but fails to prevent or detect small flaws in the data gathering.

Some of the strategies used, are:

- Packet monitoring to detect problematic sensors
- Markov models to estimate anomaly-free probabilities from past observation traces and derives optimal anomaly detection rules for sensor failure detection [4]

2.2.2 Homogeneous Approach

Homogeneous approach consists in utilizing sensor redundancy to identify if a sensor shows some anomalous behavior. The principle is a faulty sensor will display an awkward behavior when compared to functioning ones. And using comparison algorithms such as majority voting between common sensors.

Time-series analysis model such as, Auto Regressive Integrated Moving Average which compares the predicted measurement with the reported measurement. Homogeneous approaches take advantage of the redundancy gain from deploying multiple sensors of the same type spatially close to each other increases the cost of deployment[22]. Finding an optimal threshold isn't actually very easy, as it can decrease the accuracy of the system.

2.2.3 Heterogeneous Approach

Systems that use an heterogeneous approach for fault detection, are systems that use different types of sensor data to detect faults in each components. They can have three types of heterogeneous models[29]:

- Knowledge based Models
- Data based Models
- Process Based Models

2.2.4 Knowledge-based Models

Knowledge based models rely on rule based methods that infer the occurrence of a fault. These rules may be defined by an expert, that describes what should be considered a fault. Obviously, besides making the system vulnerable to new types of faults, and its efficiency

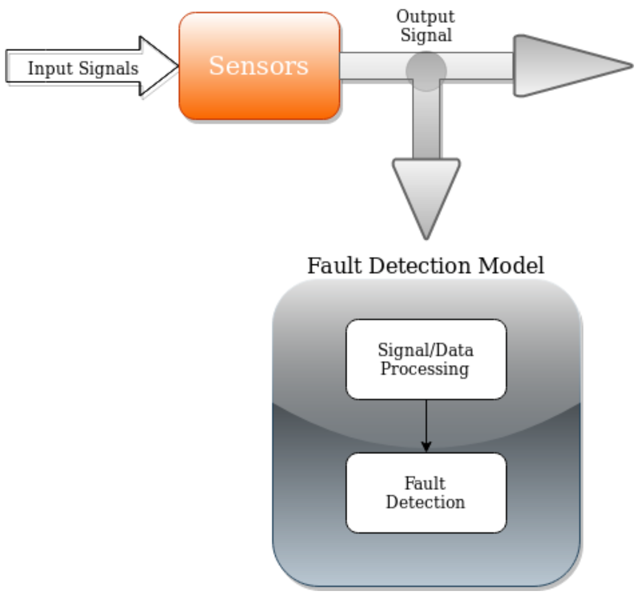


Figure 2.1: Scheme for Models based on data

depends on the expertise of the specialist, it is prone to problematic assessments, and demands periodic management of the rules in a long run.

Fuzzy logic systems are also knowledge based system[14], as it uses rule based inference to make decisions. The difference being, the fuzzification of inputs and, the output of such function instead of being binary, returns the fault severity.

Data based Models

For fault detection in data based models, the main focus is to monitor significant variations in data compared to the common oscillations in the system. This may happen using a simple verification of a given field of a given machine in several records to detect a sensor failure, or full analysis of several sensor's readings.

There are different types of models that use data analysis to detect faults. Some models use signal analysis, others exploit historical data to find thresholds for maximal and minimal values, or to use as an input to parametric equations to understand the fluctuation of signal through time.

Pattern recognition is also a valid option to discover some abnormality in data. For that, neural networks has accomplish significant results. Unsupervised learning has been a proved solution to achieve accurate results.

- Spectrum analysis and parametric models
- Pattern recognition

Usually, signal analysis[22] uses techniques such as bandpass filtering, fourier analysis, or some analytics regarding wavelet, spectrum, or correlation between dependant vari-

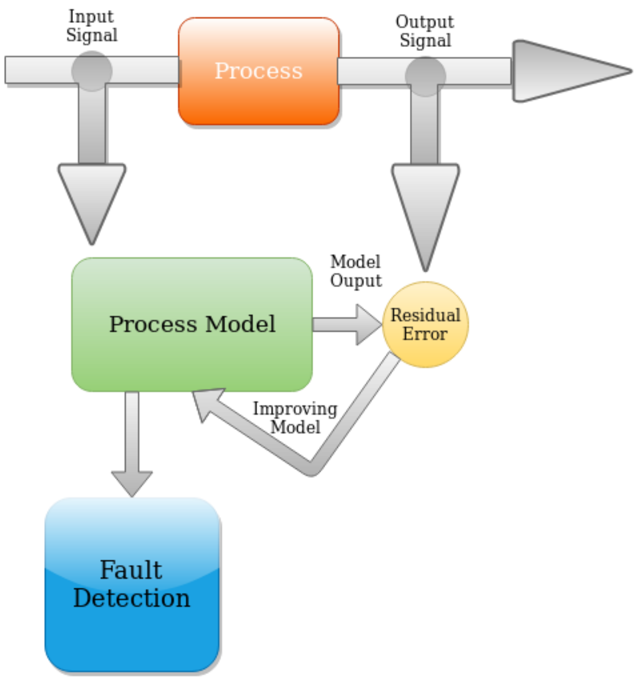


Figure 2.2: Scheme for Process-based Models

ables. There are also techniques that use estimation of parameters (parametric spectral estimation, ARMA parameter estimation).

Process-based Models

The idea behind this type of models is to have a parallel process, usually takes the form of a prediction model that simulates the actual process, receives the same inputs and, when both processes are finished is expected to have similar results.

- Parity Equations - Through the use of mathematical equations that model the components in analysis, assesses possible errors that occurred in inputs, outputs, and deviations from the equation model to the actual process.

- State Observers and State Estimation - Reconstructs the outputs of a system from the inputs, with the integration of observers, by estimating the error involved in the process, or the residual error between actual process, and the simulation component. [13]

- Parameters Estimation - Estimation of parameters based on relationships between faults and changes in physical parameters(such as voltage, resistance, friction, inductance, capacitance, etc.)

2.3 Fault Imputation

In an IoT context, Imputation is the process of replacing missing data or incorrect values, by inserting an estimation value, to be able to process data in a effective manner. The value that comes from an estimation, intends to be as close as possible, to the value that was not registered.

To replace those values, there are different models that try estimate the missing value, and can be categorized as the following [20]:

- Information-based Models
- Similarity-based Models
- Error-based Models
- Probability-based Learning

2.3.1 Information-based Models

Information-based models are models take advantage of data quantification techniques to infer which are the more influential features to determine the value of a target feature, this way producing an estimate. Entropy, information gain or frequency of a single value, are some of the measurements that are common when implementing these types of models.

Models that use information extraction from data to make predictions, have a similar approach to decision trees. Although, these models are commonly applied to categorical data prediction, some adaptations allow to extend its reach, and allow, continuous features.

2.3.2 Similarity-based Models

Similarity-based models are models that utilize feature spaces, and measures of similarity between previous instances, to be able to predict new incoming values in a dataset. By computing distances between instances, it is capable of attributing a value in target classification. An algorithm that uses similarity principles is the Nearest Neighbor Algorithm.

2.3.3 Probability-based Models

To make a prediction, these types of models use fundamentals of probability theory, namely conditional probabilities, the probability chain rule and the Theorem of Total Probability, in order to make a prediction on the most likely value. A model that uses this type of approach is Naive-Bayes Model.

2.3.4 Parametrized Models

Parametrized Models are models for algorithms that performs a search for a set of parameters in order to fulfill the objective of minimizing the sum of errors gathered from the predictions.

Some algorithms that distinguish themselves:

- Support Vector Machine
- Multivariable Linear Regression Models
- Non linear Regressions
- ISTM (Incremental Spatial-Temporal Model) [23]
- Neural Networks

Chapter 3

Context and Design Decisions

3.1 Context

As part of SATO's project, the work that will be developed, focuses on aspects of Data Quality and Dependability. The data comes from a network of sensors, and can't take various formats, types and ranges of values.

This project follows the ANNODE methodology standards:

- The usage of multiple estimates using Multilayer Perceptrons(MLPs) to predict real world environment behaviours.
- The attribution of a quality evaluation of sensor measurements.
- Utilize the MLPs estimates to identify faulty behaviours in sensors measurements.

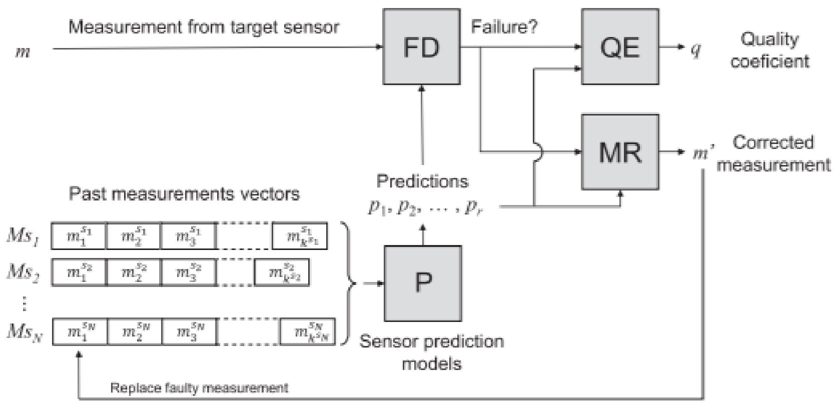


Figure 3.1: ANNODE methodology[17]

When the architecture started to be formed, there was an intention of building a structure that can easily fulfill the objectives of providing a platform able to correctly detect outliers, treat data so it can reach other parts of the project to be processed. It is the first step of SATO's operation, and should secure a quality standard for all the following operations.

Because of the early stage in the project, the platform was designed with notions of modularity and efficiency, facilitating the integration of other incoming features. Because of it, creating an ”open” platform was also a priority in implementation.

Having the goal of becoming a full functioning system as soon as possible. Some adaptations had to be made, taking in account the differences between environments, and differences in constraints of both projects.

Main concerns arose from the changes between the amount of sensors that an instance is going to serve and between types of sensors necessary to monitor aquatic environments such as rivers, and to monitor all energetic appliances, as well as environmental conditions, that are placed inside and outside a building.

Consequently, if ANNODE's goal was to achieve a high level of accuracy, for SATO Quality, the goal is to reach the high values of accuracy achieved by ANNODE, in addition of having a framework capable of dealing with an high volume of data arriving into the system. A Black-box approach to the development was suggested, because of it, every input, output, and variations on the system behaviour should be easily accessible, to be monitored further in SATO operation. This approach should also apply, by requiring minimal interference on the normal execution, which translates to decisions that allow the maintenance of platform, ideally during

The data quality and dependability framework was idealized as a component of a larger project, which as the goals to ensure the correction of faulty measurements, that are retrieved by the sensors. There were some extra concerns during the conceptualization of the structure of the project, namely:

- Since its a project at an early stage, changes, replacements or the inclusion of certain features are expected. Because of this modular structure of easy management is necessary, a simple, flexible structure that allows these type of changes without the need of implement major changes to already implemented flows and objects

- Achieving efficient and fast execution, since the speed and throughput capacity of the framework will dictate the number of sensors that will be attributed to system, and the overall implementation and configuration of the rest of the SATO system.

- There will be an unknown variety of sensor types that will be processed on this system, therefore, every step of the data quality inference should be founded on common characteristics present in sequential data analysis.

- Use a black box approach, where it should be possible to understand how the system is operating by monitoring the inputs injected, and the outputs that are triggered consequently.

3.2 Framework Structure

To be able achieve those targets, the structure is implemented as a decentralized structure in which modules are appended, according to the necessity of having different functionalities.

The modules are called Components, as each of them has different tasks, requirements and interactions with the system and other components, some subclasses were implemented to ensure that particular specifications of a module would not overload the generality of other components.

The data quality framework core structure is then composed by an instance Manager, and several instances of the superclass component. On top of that, classes, files, configurations are stored in order to be utilized by different modules across the project.

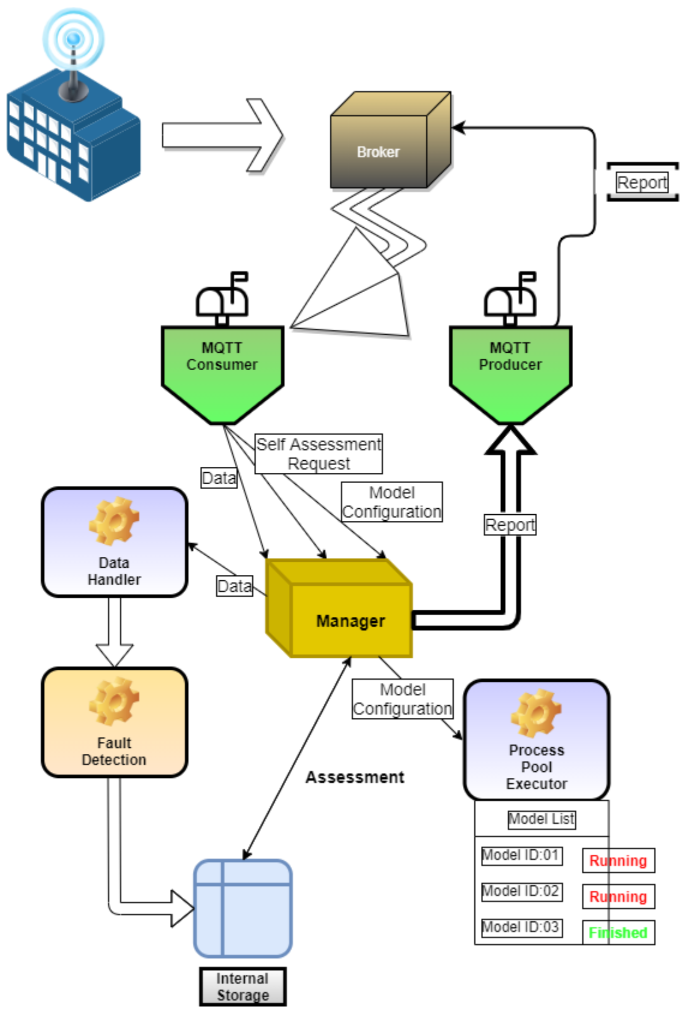


Figure 3.2: Framework Architecture

3.2.1 Manager

The class responsible for managing the different components in the system, giving indirect access to the resources of the system, mediating the message passing between Component instances, and also has direct access to each running instance of Component class.

Decides, in which terms each Component is going to be implemented in the context of the running application. Has access to all the data stored in the system, and Employs the notion of state in all the components, and is able to restore and save system's state. It can be considered the Main of the project, as it is responsible for starting all the Components in the system.

Each Component is stored under a name, that can be used to get the properties that characterize the object. This allows to add components to the system, without having majors changes, in the code.

3.2.2 Component

Component is the class that structures the instances to be added to the Manager instance, that manages the platform . Each instance that derives from Component class, is expected to perform some type of task, that is going to be called when the platform is running. Adding to this run() method, there are methods to save and load files, and attributes that should be in every Component instance.

Since, the Component class is such a simple abstraction, and adding more features would result in instances having a lot of useless methods. There was the need to specify further the types of Components that should be added to the platform.

IOComponent

The communication products arrive to the manager through IOComponents. IOComponents act as interfaces between the SATO's system, and external sources of information. The IOComponent structure should serve as a way to standardize inputs for the manager to redirect or act accordingly, this will occur not only by dictating the type of format that should enter the system, but has the job of associating each incoming message, to a certain flow inside the system.

Since, the SATO Quality framework should be used as an opaque structure, it's structurally important to specify communications interfaces that act on the same level as all the different modules, and not be implemented as the main starting point of all operations.

Communication structures are prone to error. If for any reason, the communications structures are down, other parts of the Quality process cannot be at risk, or dependant.

WorkComponent

WorkComponents are the most common subclass of Components instances. As all modules share common restrains, methods and implementation similarities between them, the creation of an abstract class facilitates major changes that influence multiple models at the same time. This abstraction serves the aspect of maintainability, as it allows complex modifications to be made simple for the programmer to implement.

To perform any given task, all the methods, classes and information should be found in a single place, hence the creation of workcomponents. To be able to communicate between them, have a FIFO queue is needed, and to perform the task given, each workcomponent should also have a thread associated.

3.2.3 Intra-Component Communication

Since the manager has a mapping of all the running components, if a component has to send data to perform a given task, that component will just have to create a new Message object to be passed. This message encapsulates two objects:

- a Task object, which contains the task to perform, and the identity of the component who requested it
- a data object, taking the format of a JSON, or a Dataunit object

This gives all the information needed to perform a task, where to redirect the information after task is finished. As it mediated by the manager, security restrictions can be implemented easily based on the type of Component that initiated the communication

3.3 Data Flow

3.3.1 Data Handling

Each entry enters through the system by means of an IOComponent. In this project scope, the main method of fetching data is by means of a **Kafka consumer**. This **Kafka consumer** is going to consume messages of designated topics. After those messages reach the system, they are task categorized and redirected to the corrected part of the system, where they'll be processed.

Messages that are destined to be subjected to quality evaluation and outlier detection, proceed to be stored, and reach the first stage in quality evaluation. Contextual and Representational standards should be looked at the earliest stage. Does the new entry has the right amount data to be stored in the system? Does it qualify? Representational standards such as format consistency, can be looked by comparing the format of the newly arrived message relative to the standard format of previous messages, and see if they share the same fields, and if they share the same type of value.

In terms of contextual standards, Timeliness is something that must be looked at this stage, how much is the new message distances itself, in the time space, from the last entry? Did it respect the standard distance? Was it early, late? Was there any measurement that should arrive before this?

Before, the entry should be inserted in the outlier detection cycle, it is necessary to check if the entry respects the notions of data quality, and, signalize what problems have been encountered when it arrived.

3.3.2 Outlier Detection

For each entry, two matrix are formed, one with the values, for instance, Temperature values and the second, containing every timestamp, with the same time period constraint. Both matrix are then subjected to some operations, in order to gather the exact amount of data to serve as input to a multilayer perceptron, which will output an estimate/prediction for that particular time instance.

To detect a outlier, the system requires three ANN estimates: ”self” that uses only input values from the main sensor, ”neighbours” that uses input values that come from all the neighbours sensors, and ”all” that uses values from all the sensors(main and neighbour sensors).

For all estimates, an error is obtained relative to the value registered by the sensor. This error is used to obtain a probability of being a outlier.

If two or more estimates have considerable errors, this meaning have a probability that exceeds the confidence margin, then the measurement is marked as an outlier, and the raw value should be replaced by an weighted average of those estimates.

If the entry in the system is not considered an outlier, a quality index is obtained through a calculation based on the errors.

After the processing is complete a new entry is integrated in the system as processed data, and a Dataunit object is sent by a kafka producer to predesignated topic, and beyond the scope of the project.

3.4 Model Flow

A model configuration is received in json format, with details regarding the whole process of building a keras model. That should contain:

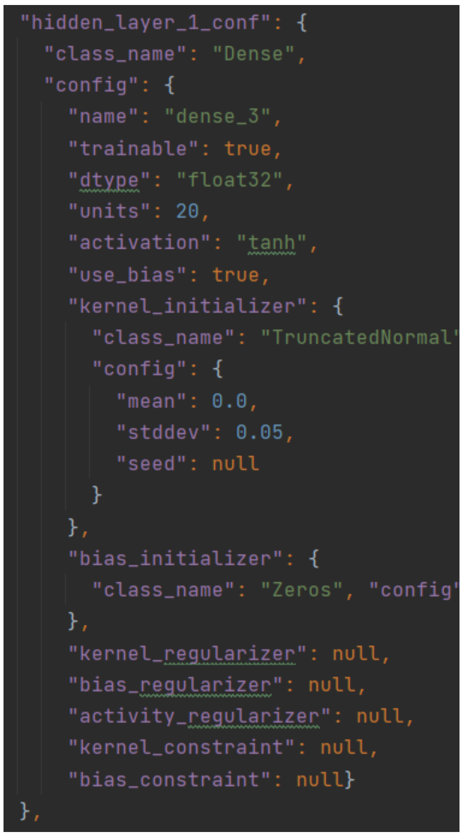
- Layer configuration
- Optimizer, loss function, metrics

- Model construction specifics, such as number of epochs, loss function or when to store checkpoints
- Data specifics such as period length, skip period, save path
- Options related to model training

3.4.1 Layer Configuration

Each layer of the model must be described, it should follow the configuration's guidelines of the Keras Layers API, however it is necessary to add layer type information.

Initializers, regularizers, shape of inputs, activation function, and constraints are discriminated here. Each layer has a configuration associated, and it will be inserted in the model by order of declaration.

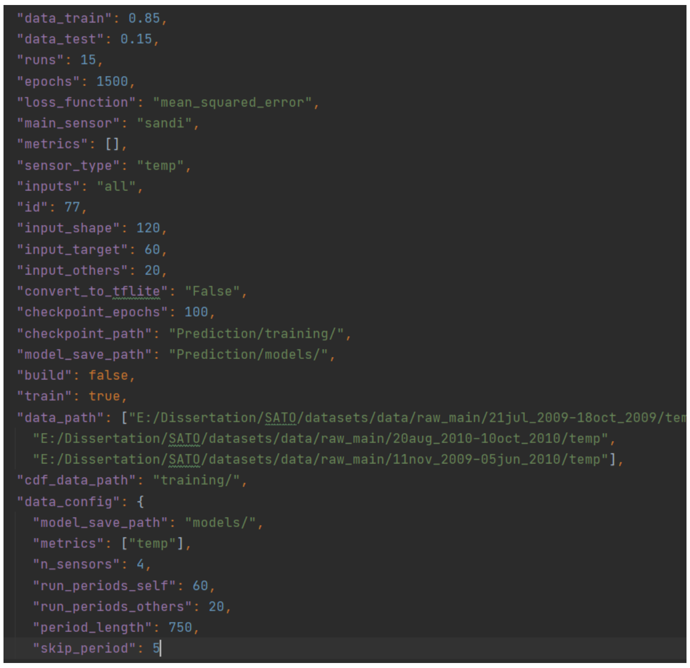


```
"hidden_layer_1_conf": {
  "class_name": "Dense",
  "config": {
    "name": "dense_3",
    "trainable": true,
    "dtype": "float32",
    "units": 20,
    "activation": "tanh",
    "use_bias": true,
    "kernel_initializer": {
      "class_name": "TruncatedNormal'
      "config": {
        "mean": 0.0,
        "stddev": 0.05,
        "seed": null
      }
    },
    "bias_initializer": {
      "class_name": "Zeros", "config'
    },
    "kernel_regularizer": null,
    "bias_regularizer": null,
    "activity_regularizer": null,
    "kernel_constraint": null,
    "bias_constraint": null}
},
```

Figure 3.3: Model Configuration:Layer Configuration

3.4.2 Data Configuration and Model Specifics

To be able to setup data, ready to be inserted in a keras model, there's the need to specify some construction parameters that can differ, depending on what physical property is being measured and the way it changes across time. The number of sensors that have some correlation and will be utilized to predict one's future value, must be placed in this configuration as well.



```
"data_train": 0.85,
"data_test": 0.15,
"runs": 15,
"epochs": 1500,
"loss_function": "mean_squared_error",
"main_sensor": "sandi",
"metrics": [],
"sensor_type": "temp",
"inputs": "all",
"id": 77,
"input_shape": 120,
"input_target": 60,
"input_others": 20,
"convert_to_tflite": "False",
"checkpoint_epochs": 100,
"checkpoint_path": "Prediction/training/",
"model_save_path": "Prediction/models/",
"build": false,
"train": true,
"data_path": ["E:/Dissertation/SATO/datasets/data/raw_main/21jul_2009-18oct_2009/tem
  "E:/Dissertation/SATO/datasets/data/raw_main/20aug_2010-10oct_2010/temp",
  "E:/Dissertation/SATO/datasets/data/raw_main/11nov_2009-05jun_2010/temp"],
"cdf_data_path": "training/",
"data_config": {
  "model_save_path": "models/",
  "metrics": ["temp"],
  "n_sensors": 4,
  "run_periods_self": 60,
  "run_periods_others": 20,
  "period_length": 750,
  "skip_period": 5
```

Figure 3.4: Configuration:Model Specifics

- What percentage of data should be used in training, and for testing
- Number of epochs
- Keras optimizer
- Keras metrics
- Loss function
- Main sensor identification
- Takes account values from the main sensor, from the neighbours, or all of them

3.4.3 Communications

The main protocol of communication utilized is MQTT communication protocol. It's not part of Data Quality project, but it is part of the SATO project.

The MQTT communication in this project is done by means of a cluster of Kafka server, more specifically a cluster of kafka brokers.

Distributed, highly scalable, fault tolerant and secure, each Kafka broker stores event streams under different topics, each topic can have multiple partitions, and replicate data across a set of brokers. The topics can afford to have multiple clients writing to them, and reading from them, The events are sent to the broker by what's designated as Kafka Producer, whereas the client reading from the broker, it's called a Consumer.

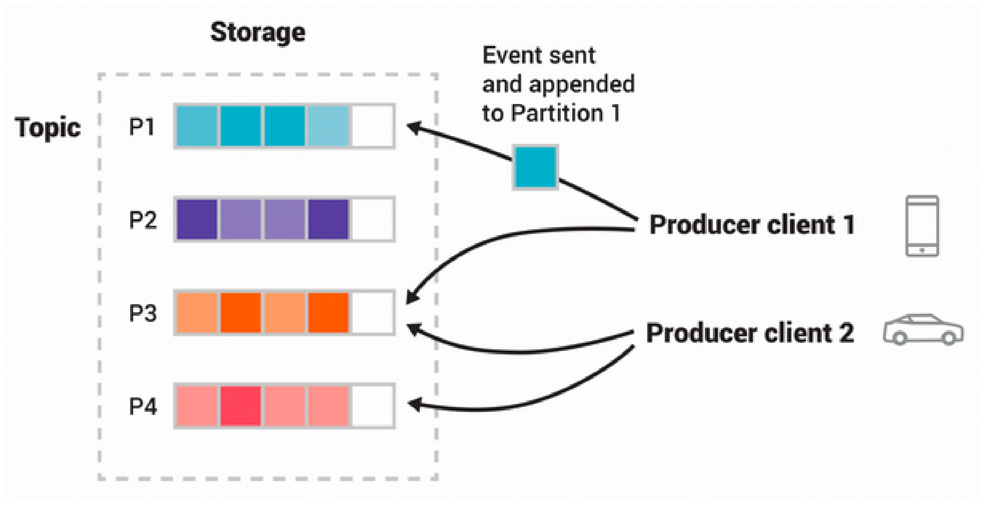


Figure 3.5: Kafka:Event based Communication[6]

Since, SATO Quality requires both receiving and sending events, both entities (Consumer and Producer) have to be implemented.

3.5 Monitoring

In order to oversee the normal execution of the program, some structure is needed to make periodic assessments. This assessments should describe as accurately as possible, how is program working during runtime.

Since, each execution implies multiple activities running at the same time, a way that this can be seen, is by record the time that takes each part of the flow to complete a single passage, identify each record accordingly, and either store them in a easily accessible place, or redirect those records to a specialized unit to process them and send an external entity to handle the arriving information.

Chapter 4

Implementation

4.1 Datasets

The source of the data used comes from SATURN Observation Network, in which data was gathered from a set of sensors placed along an estuary-river system from the river Columbia, located in the United States of America, between the states of Washington and Oregon, and is connected to the Pacific Ocean. This part of the river is monitored by CMOP Science and Technology Center monitoring network. The dataset consists of 6 sets, from different periods in time. Each set is composed of temperature and salinity measurements, with associated timestamps from 4 different sensors. For the study, only readings from the temperature sensors, were used.

The different time periods present in the dataset are:

A 1/July/2009 - 16/July/2009

B 21/July/2009 - 18/October/2009

C 11/November/2009 - 5/June/2010

D 20/August/2010 - 10/October/2010

E 2/October/2010 - 5/October/2010

F 1/October 2013 - 31/December/2013

Small sets of data, or sets that are included within other sets, such as A), E) were not used at all. F) was used to evaluate the development, while B), C) and D) were used strictly to train models.

Each dataset contains information about 4 different stations:

- Jetty A (jettya)
- Lower Sand Island Light (sandi)

- Desdemona Sands Light (dsdma)
- Tansy Point (tansy)

4.2 Modules

4.2.1 MQTT

The MQTT module allows the framework to exchange events with a kafka broker, it hold an instance of a kafka Consumer and an instance of kafka Producer. It's the main IOComponent for the project. The library used to implement this exchange it's the Kafka-Python library[19].

Consumer

By reading a configuration, the Consumer connects himself to a list servers, and subscribes to defined topics. For this operation to continue, is necessary to set ”group.id” and ”client.id” that were set as ”SATO”, and ”SATO_Quality”, respectively. The most important configurations that were set, were ”auto_offset_reset”, that defines which resetting policy to use in case of OffsetOutOfRange errors, was set to ”latest”, in order to move to the most recent offset, so that the same message won't enter the system multiple times.

It permits the deserialization of both key, and values from the new unit arriving the system in JSON format.

The Consumer is the ”start” of each flow, consumes a ”max_records” number of records in each iteration, in this case set with the value of 100 messages, classifies each message based on the TopicPartition, and sends them either to a DataHandler instance, to be processed as a measurement received from the sensors, or to the Prediction module, to be treated as a configuration.

Producer

The Producer instance is also defined at the MQTT module, also receives a similar configuration that sets itself to given servers, and, which serialization functions to use, for both keys and values.

Contrary to the Consumer instance, the Producer is only called at the end of each flow, and is responsible for reporting each operation of the framework back.

4.2.2 DataHandler

DataHandler is a Component that holds objects of type ”DeviceData”, that receives, handles and submits entries to outlier detection and quality assessment. The receiving data operation, consists in comparing the new entry to the known sensors, if it's the first entry of a sensor it creates a new DeviceData for that sensor, if not, forwards the arriving entry to be compared with the data stored.

DeviceData

DeviceData is the object that has all the data of a given sensor. Data can be separated in three parts:

- **Buffer** - First 60 entries of that device, is used to state sensor standards(data format, measurement rate) .
- **Raw Data** - Data that arrived to the system, without any type of processing.
- **Data** - Data that has already been processed.

Pandas was the python library chose to store all the values, being a well constructed, flexible library, easily integrated, as it is the case, of libraries such as Dask[7], Vaex[30] or PySpark[2], and being to store large quantities of data, while achieving high performances in operations, such as filtering, or calculating statistical properties of a dataset. It's a starting point for any efficient data handling python program.

Pandas Dataframes were used in Buffer, Data and Raw Data. There was however a slight change added to store Raw Data, since this type of data is rarely used during the execution, and each append operation in pandas Dataframe always involves the creation of a new object with a new entry appended, a object named **Frame** was created. It is mainly a pandas Dataframe, and list object, where incoming data is first stored in the list. When that list reaches a certain length, or is called with getData() method is concatenated along with the Dataframe, and the execution proceeds. This allows a much faster appending speed, for attributes that are mainly used to store values, and are subjected to very few calls.

Quality Operations

After appending the new entry to **Raw Data**, each entry is subjected to representational consistency and timeliness verification. After checking if the data format is consistent with the standard data format, it will check if the time difference between the arriving entry and the last entry stored in the system.

If the time difference exceeds, or is below, a tolerance factor from the standard time difference, it is reported. If the time difference is considerably different from what was expected, it calculates how many entries didn't reach the system. Those created entries are labeled as missing values, and the quality evaluation is stated with 0. Both, new entry and created entries are then passed to the outlier detection module. The created entries are treated as an outlier, and it's value is calculated from the estimates of the keras models.

4.3 Outlier Detection and Quality Evaluation

When initiating an instance of this module, all the keras models available in a folder are loaded. Based on the sensor identification, dimension (”self”,”neighbours” or ”all”), models are chosen for each existing identification, and using the testing evaluation score that was produced during training, the best ones are chosen to be used during the execution.

Relative to ANNODE’s algorithm to create inputs for the ANN model predictions, some changes had to be made. Many features were kept, the size of the inputs for each dimension and, principles behind both algorithms are still the same.

The number of inputs per dimension, ”self”= 60 values, ”neighbours”= 60 values and ”all” = 120 values. Each set of data used to find these inputs, is a subset of all the timestamps stored present in each correlated set of sensors. This subset of data is a result of

In AQUAMON’s each ANN input, was a result of applying a function that selected certain indices, from a preprocessed set of data, and limited to a range of 750min from the newly arrived measurement. The ”neighbours” inputs are obtained from the concatenation of 20 values from each sensor, ”self” utilizes 60 values from the main sensor, and ”all” is composed by the other two dimensions.(60 values of the main sensor + (20 + 20 + 20) from each neighbour sensor.

The algorithm to select data suffered a change in terms of implementation, the old implementation, was simply not efficient in a building environment, where it is expected a greater amount of sensors per division. Even if multiple instances of project were placed inside a building, it would be nearly impossible to maintain the speed of response necessary for the user to be satisfied.

Where Times_MS is the time column for the main sensor. As you can see, the algorithm complexity in Big O notation, can be classified as, in best case scenario, O($D_{s_0} * N_s * log_2(D_{N_s})$), where D_{s_i} refers to the dimension of time table for the sensor S_i, with $i = 0, 1, .., n$, and N_s the total number of sensors correlated.

The solution was to implement an algorithm that would be capable of detecting the nearest timestamp(per sensor) relative to the main sensor. The fastest way and most secure way to perform this task, is by creating a matrix formed by the timestamps of all the sensors, computing the module of the differences from a main sensor timestamp and retrieving the indices where the distance is minimum. By computing this matrix for a range of number, and selecting the first solution where the sum of minimum distances is minimal, the result will be the set of indices closer to each other, under a range of number.

Algorithm 1 AQUAMON's Implementation

```
1: Get fst (first common time in dataset)
2: With $time\_t \in Times\_MS$
3: for $time\_t = fst, \ldots, last\ index$ do
4:     for $sensor = 0, \ldots, number\ of\ sensors$ do
5:         s1) Find closest time relative to time_t (Bisection Search)
6:         if Conditions are met then
7:             s2) Gets Index, time difference and timestamp
8:         else if Conditions not are met then
9:             s3) Do not continue process
10:        end if
11:    end for
12: end for
13: s4) Determine which indices will be used
14: for $sensor = 0, \ldots, number\ of\ sensors$ do
15:     s5) Find Initial time (Bisection Search)
16:     for $number\ of\ inputs = 0, \ldots$ do
17:         s6) Append wanted value
18:     end for
19: end for
20: Return inputs for each dimension
```

Algorithm 2 Optimal Distance Algorithm

```
1: s1) Filtering data based on 750min from arriving timestamp and skip period
2: if $length\ of\ Data\ <\ number\ of\ inputs$ then
3:     Store value. Not subjected to processing. Signaled as unprocessed
4: end if
5: s2) Establish indices to use as inputs for the main sensor.
6: for $I_s = 0, \ldots, n\_inputs$ do
7:     for $Precision = 0, \ldots, min(2, f(n\_inputs, data\ length))$ do
8:         Compute Differences Matrix $DM_s$
9:         Get indices of minimum distance (per sensor)
10:        Convert indices to values
11:        Store values, and sum of distances
12:    end for
13:    Get minimum cost index $ind_s$
14:    Increment by $ind_s$ all further $I_s$ except $I_(n\_inputs)$
15:    Remove appended indices from further iterations
16: end for
17: for $sensor = 0, \ldots, number\ of\ sensors$ do
18:     if sensor= main sensor then
19:         s3) Picks number of inputs per neighbour sensor
20:     end if
21: end for
22: Return inputs for each dimension
```

4.4 Model Construction

The model configuration is received as a Dictionary object, in the Prediction system. The Prediction system consists mainly on a Process Pool Executor that can launch multiple processes in parallel, during runtime, having a defined maximum of 3 processes that can run simultaneously.

To keep the Process Pool available (a single exception on a process will result in a crash in the Process Pool), all the verification and validations concerning the model configuration, must be done before the submission of such configuration in the process pool.

Subsquently, a set of configuration is submitted, having each configuration converted into an object that has all the methods necessary to run an ANN building process. Such object is then converted to a Future, and the process is launched.

Depending on a configuration option **build** (a boolean property), the construction can either build a new set of inputs and targets to be used in running, or this step can be skipped, in such case the model will either retrieve inputs and targets with the given information, or if there is no inputs sets that fulfill the characteristics of the configuration, it will ignore the option, and force the building of those inputs by fetching all datasets that are placed under **data_path**.

In case of multiple datasets, after all data gathered from the different datasets, the inputs and targets are concatenated, and following ANNODE's model training system, are subjected to a permutation and split according to a percentage for training and testing, and the model training starts.

When the training finishes the result of the evaluation using the testing set is stored in a file.

In the final stage of the process, a Cumulative Distribution Function(cdf) is described from the errors of the all the predictions. This distribution is what is enables to situate an error, result of an outlier detection estimate, giving him a probability of being an error.

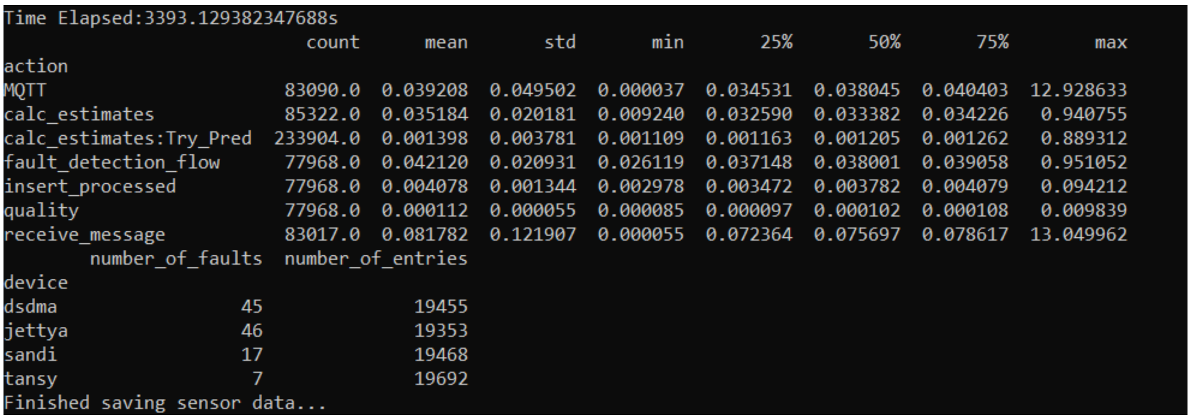
4.5 Monitoring

To be able to understand the framework is behaving, three types of information should be present:

- A timestamp to situate the report across time
- Information related with the final outputs of the operation
- Reports of intermediate operations that occur in the system, and that are not represented in the outcome

The next figure is an example of a report done during runtime.

Has a basic timestamp, in this case, time measured from the starting of the program.



```
Time Elapsed:3393.129382347688s
                           count      mean       std       min       25%       50%       75%        max
action
MQTT                     83090.0  0.039208  0.049502  0.000037  0.034531  0.038045  0.040403  12.928633
calc_estimates           85322.0  0.035184  0.020181  0.009240  0.032590  0.033382  0.034226   0.940755
calc_estimates:Try_Pred 233904.0  0.001398  0.003781  0.001109  0.001163  0.001205  0.001262   0.889312
fault_detection_flow     77968.0  0.042120  0.020931  0.026119  0.037148  0.038001  0.039058   0.951052
insert_processed         77968.0  0.004078  0.001344  0.002978  0.003472  0.003782  0.004079   0.094212
quality                  77968.0  0.000112  0.000055  0.000085  0.000097  0.000102  0.000108   0.009839
receive_message          83017.0  0.081782  0.121907  0.000055  0.072364  0.075697  0.078617  13.049962
        number_of_faults  number_of_entries
device
dsdma                 45              19455
jettya                46              19353
sandi                 17              19468
tansy                  7              19692
Finished saving sensor data...
```

Figure 4.1: Monitoring Example

By measuring the time that took to complete a single passage, for each significant intermediate operation, we can have a deep knowledge of what's happening in the system. If some part of the process is down, slow, or if a new implementation is not doing what is supposed to, it is reflected on the analysis.

To analyze the validity of the operation, data is gathered from the results of process, in this case, only the number of outliers detected per sensor, however, since each dataunit contains the all results from intermediate steps of the operation, it could be interesting to obtain the average quality evaluation, or the number of missing entities that have been processed. That information, although not used, is all available for future analytical inference.

Additionally, the system works with two loggers, one for interprocess communications, such as Kafka communications, where all the entries that enter and exit the system are signaled, and another that reports every operation done, during the processing of a measurement. These loggers are setup automatically, when associating a given Component to the Manager, and they're differentiated based on being a IOComponent, or a WorkComponent.

Chapter 5

Results

To open the possibility of comparing the differences between the previous approach of the algorithm, and SATO's approaches, the dataset utilized was the same. The temperature dataset from the time period of "1/October 2013 - 31/December/2013".

Two approaches were considered, given the intention to make an analysis on the conveniences and concerns, in utilizing the ANN models to estimate the value of missing entry. Because of it, a set of tests was created for SATO without creation of entries, and with creation of entries.

Also, as a start for a quality comparison, there are represented the plots for raw values of each sensor, and the outlier detection results previous to SATO.

To each of the approaches, the main goal was to analyze the capability of each detect outliers and how were they affected with different confidence intervals. On top of that, two statistical concepts are used to make a qualitative assessment of the outliers detected: **Detection Rate(DR)** and **False Positive Rate(FPR)**.

DR = $\frac{\#TruePositives}{\#TruePositives\ +\ \#FalseNegatives}$

FPR = $\frac{\#FalsePositives}{\#FalsePositives\ +\ \#TrueNegatives}$

Given the changes in the objectives of the work, it was also included a performance analysis. Comparing both SATO to AQUAMON was essential, since the quality analysis is not sufficient, to consider dependability factors.

To determine the performance of the AQUAMON project accurately, the section of the RPC communication was removed, so there is not the necessity of delaying messages and to eliminate the influence of this section in the rest of this project. To highlight the actual performance of that project, the dataset was fully loaded before the initiation of time measuring, and all the performance data was saved when all the queues of process were empty. At worst case scenario, the last entry of the dataset might not be covered at the last section(Quality Assessment), in both performance and quality analysis.

The average speed of each part of the Data flow process was reported, under three phases:

- Receiving Data

- Outlier Detection
- Quality Assessment

Following that, it is also reported how much time each framework takes at processing a dataset with 85753 entries, how many of them pass through each phase of the process, and long it takes for each of phase to complete.

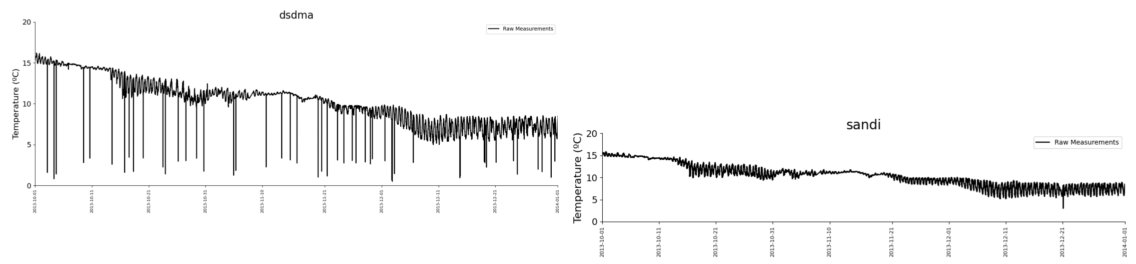


Figure 5.1: Plots for Unprocessed Data:Sandi and Desdemona.

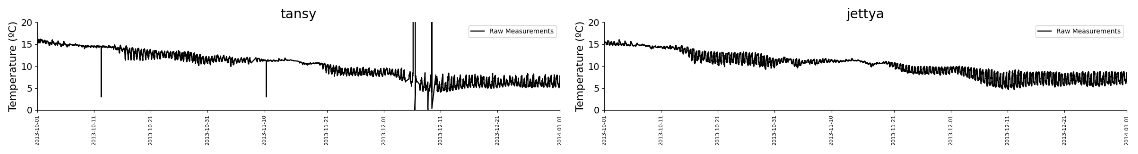


Figure 5.2: Plots for Unprocessed Data:Tansy and Jettya

Sensor	Jetty A			Desdemona			Tansy			Lower Sd		
Real Outliers	0			44			11			1		
	Detected	DR	FPR	Detected	DR	FPR	Detected	DR	FPR	Detected	DR	FPR
0.98	4	0%	0.129%	52	100%	0.0422%	9	81.82%	0%	2	100%	0.053%
0.99	-	-	-	-	-	-	-	-	-	-	-	-
0.995	1	0%	0%	48	100%	0.0211%	9	81.82%	0%	2	100%	0.053%
0.997	0	100%	0%	47	100%	0.0158%	9	81.82%	0%	1	100%	0%
0.998	0	100%	0%	45	100%	0.0158%	9	81.82%	0%	1	100%	0%
0.999	0	100%	0%	45	100%	0.0158%	9	81.82%	0%	1	100%	0%

Table 5.1: AQUAMON's Results

5.1 Quality

For the lower intervals of confidence, 0.98, 0.99, we can see a disparity between the results obtained previously.

SATO performs worse under this conditions, much likely due to the less accurate models created. It is possible to verify this, by looking to the Tansy sensor plot, and see that the predictions for missing values are too distant from the processed values.

However, as the intervals of confidence become increasingly larger, it is possible to see that gap in results vanishes. For the sensors Jetty A, and Lower Sd, all the outliers are correctly detected for bigger intervals of confidence larger than 0.99.

Sensor	Lower Sd						Tansy					
Real Outliers	1						11					
	SATO without Creation			SATO with Creation			SATO without Creation			SATO with Creation		
	Detected	DR	FPR	Detected	DR	FPR	Detected	DR	FPR	Detected	DR	FPR
0.98	1	100%	0%	1	100%	0%	8	72.73%	0%	27	100%	0.05%
0.99	1	100%	0%	1	100%	0%	7	63.64	0%	17	100%	0.032%
0.995	1	100%	0%	1	100%	0%	6	54.54	0%	11	100%	0%
0.997	1	100%	0%	1	100%	0%	6	54.54	0%	9	81.82%	0%
0.998	1	100%	0%	1	100%	0%	6	54.54	0%	9	81.82%	0%
0.999	1	100%	0%	1	100%	0%	6	54.54	0%	9	81.82%	0%

Table 5.2: Outlier Detection:Lower Sd and Tansy

Sensor	Jetty A						Desdemona					
Real Outliers	0						44					
	SATO without creation			SATO with creation			SATO without creation			SATO with creation		
	Detected	DR	FPR	Detected	DR	FPR	Detected	DR	FPR	Detected	DR	FPR
0.98	26	0%	0.129%	199	0%	0.922%	56	73.21%	0.075%	61	100%	0.079%
0.99	0	100%	0%	66	0%	0.306%	47	87.23%	0.03%	51	100%	0.028%
0.995	0	100%	0%	0	100%	0%	47	87.23%	0.03%	51	100%	0.028%
0.997	0	100%	0%	0	100%	0%	47	87.23%	0.03%	51	100%	0.028%
0.998	0	100%	0%	0	100%	0%	47	87.23%	0.03%	51	100%	0.028%
0.999	0	100%	0%	0	100%	0%	47	87.23%	0.03%	51	100%	0.028%

Table 5.3: Outlier Detection:Jetty A and Desdemona

For the sensors Desdemona and Tansy, if we compare with AQUAMON's approach, the quality has slightly decreased. However, when looking at FPR values for both of SATO's approach, the creation of entries influences positively this parameter by increasing the number of true negatives. And, by being able to detect more correct outliers, it

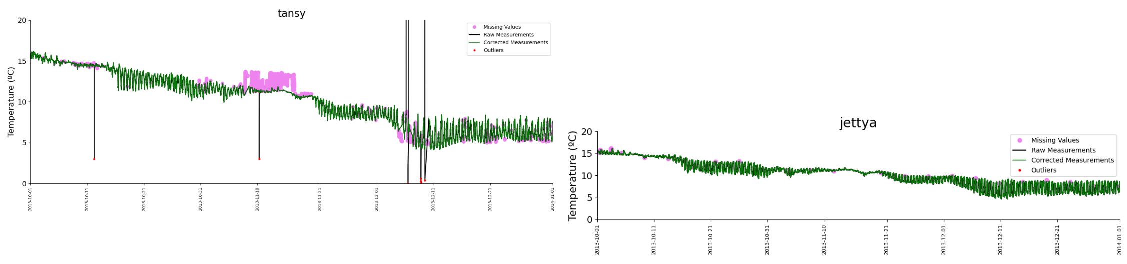


Figure 5.4: Plots for SATO(ME) at 0.999: sensors Tansy and Jettya.

does indicate that there are fewer time periods where the quality evaluation cannot be implemented.

Missing Data restricts the outlier detection process as some entries are not being processed, consequently, many outliers are not detected.

By comparison, executing the framework with creation of entries, increases DR, and consequently FPR as well, however, by creating accurate enough models of prediction, the increase of the FPR can be mitigated.

Meanwhile, when estimating entries for long periods of missing values, small errors are being added cumulatively over time. These types of events result in weaker estimations, but it also may trigger an erroneous detection of some outliers. Because of it, creating entries without any time limit is not beneficial, as it may cause a quality decay for several estimations.

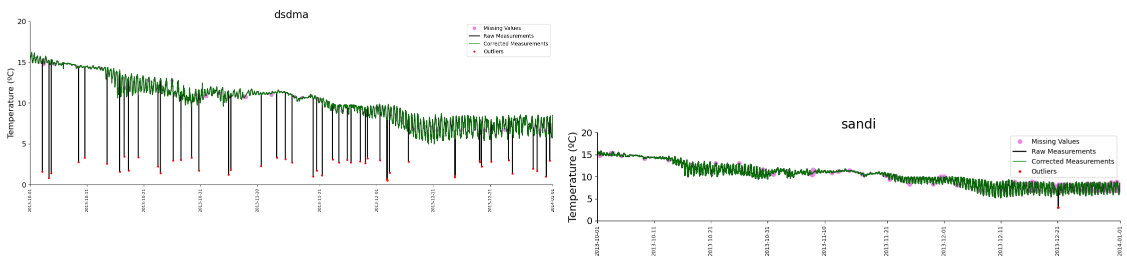


Figure 5.3: Plots for SATO(ME) at 0.999: sensors Desdemona and Sandi.

The comparison between SATO without the creation of entries, and the previous project is conclusive in regards to which one performs better at quality evaluation. Preparing inputs to an ANN, with an algorithm dependant of a limited search range, will never be better than another that uses brute force search. This can be overcome by creating entries, when data is missing. Having a dataset where the time distance is always below a fixed limit, will get the same results, using the same ANN models.

Utilizing SATO's alternative with creation of entries enables better results in quality evaluation and outlier detection, as it is able to process more entries overall, and as a consequence it provides a tighter control of the system. On top of that, only the sensor Desdemona presented a slight decrease of quality, when compared to the AQUAMON's solution. SATO has been able to preserve most of the quality achieved by AQUAMON, while being able to introduce new principles of data quality and dependability to the system.

5.2 Performance

To obtain performance evaluations from both projects, data was gathered from several executions in a machine provided by LASIGE. That machine has 2 processors with 12 core each (with hyperthreading), with 128GB of RAM memory and operates with an Ubuntu software.

For each section of each project, data was constructed measuring partial time, referring to the each part of the system that composes the data flow, parallel to this approach the global time of a single execution is also measured. Partial time measuring focuses on the time it takes for a single passage in a given section of the system, this section can be a function, or an entire flow of the framework. This type of analysis allows to understand how much time each passage takes, how many times each function was called, and the overall time spent in the program.

Also, was measured the time that each framework took at processing a full dataset.

Action	Count	Mean	Std	Min	25%	50%	75%	Max
Receiving Data (in secs)	39991.0	0.000080	0.004119	0.000013	0.000035	0.000038	0.000043	0.596375
Outlier Detection (in secs)	39601.0	0.624052	0.254832	0.143781	0.397501	0.602951	0.821559	1.611712
Quality (in secs)	39600.0	0.415414	0.273044	0.009051	0.174863	0.396609	0.617513	2.844010

Table 5.4: Performance:Previous Implementation

In this implementation, for it to be able to process a full dataset with 85753 measurements, it would took some time in between of 10 hours and, 10 hours and 30 minutes. For the SATO implementation to process the same dataset with 85753 measurements, the implementation processed the dataset in less than 1 hour and 30 minutes, in all tests that were done.

The next table shows the **worst** performance of all the tests. This particular run took 1 hour, 28 minutes and 20 seconds.

Also, from a dataset with 85753 measurements, was able to insert 1286(1046 + 240 from the initial buffer) more entries than what as arrived to the system, and 8028 more entries from what would be expected from SATO without the creation of data (78971 entries that finished the flow), as a result from filling missing data.

Action	Count	Mean	Std	Min	25%	50%	75%	Max
Receiving Data (in secs)	85753.0	0.119351	0.200350	0.000056	0.086107	0.117431	0.135010	21.620503
Outlier Detection (in secs)	86799.0	0.060601	0.031380	0.026029	0.042936	0.059010	0.069046	2.823217
Quality (in secs)	86799.0	0.000168	0.000151	0.000086	0.000115	0.000161	0.000197	0.026281

Table 5.5: Performance:SATO's implementation

In terms of performance, from the results it is possible to notice a big gap between implementations. SATO's implementation offers much better results in terms of speed, thus a greater capacity in handling multiple sensors data inputs.

Both partial and full time measurements have presented improvements, on Outlier Detection and Quality Assessment there was a decrease on time average by a factor of 10 and 2500. At the same time on a global time analysis, a single execution on SATO spends less than 7x the time that an AQUAMON execution.

However, if we consider the speed for receiving data for SATO, even ignoring the outlier of 21.63 seconds, has considerably a lower speed, when compared with other sections, this arises from the cost of utilizing Pandas Dataframes, and the operations to determine missing values, and creating new entries in the system. It is the most costly flow of the sytem, although the necessity of having organized data, and ensuring that it can detect lost measurements, this section of program should be worked on, as it is (now) the critical point, performance speaking, of all data quality operation.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This platform fulfills the standards defined in the ANNODE's methodology. The separation between, phases of Data Quality Assessment are defined, and most of the principles are either included, or easily employable.

The organization of the project allows a consistent and reliable monitoring of internal state of the framework.

Even though there wasn't a significant change in terms of accuracy in detecting outliers, it does allow the flexibility needed to progress the development of the framework, in a transparent and organized manner.

The main improvement of this implementation can be found in dependability aspects, such as Currency, and Volatility, the last one being the most improved. The framework has principles and creative space to be improved much further. Personally, I think it can be considered as an great starting point, of what's going to be a fully optimized, accurate quality framework.

6.2 Future Work

Although, the quite satisfactory results, in future versions of the framework, there is still a lot room for improvements.

The most important section that needs a well thought approach, is the capability of integrating multiple types of sensors from multiple locations. The data system despite being correctly implemented, is still does not take in account the integration of different sources of data, in order to be a full functioning system. How to relate different sources of data with different timestamps, even before it reaches any step of data quality, is something yet to be implemented.

At this moment, the system creates missing entries by estimating how many entries should have received since by calculating the time difference from the last timestamp present in the system. This is too late of a solution, as many measurements still can't be processed. There is still the need for an approach, that can act at smaller time interval than the approach that's currently being used.

In terms of quality, getting the exact entries to serve the ANN predictions is a subject that needs study, mainly what type of inputs should be delivered in the ”all” dimension. Is it better to use a combination of the two other dimensions, or should be use a similar approach to the calculation of the ”neighbour” dimension.

The Prediction Component, the component responsible for the triggering parallel training of models, still needs to be improved, studied and correctly tested. Despite performing well for several incoming configurations, it does lack a proper understanding of it's limitations, and how should be used. Attached to the Manager instance, or implemented separatel and connected to the Manager instance.

One other aspect that wasn't addressed during my implementation was security, the framework is still quite prone to unsafe interactions with the system, that can jeopardize the veracity and integrity of the values that are send this framework.

Bibliography

[1] Python Software Foundation: Python 3.7. https://docs.python.org/3.7/. November 2020.

[2] Apache. http://spark.apache.org/docs/latest/api/python/. 2021.

[3] Barun Basnet, Hyunjun Chun, and Junho Bang. An intelligent fault detection model for fault detection in photovoltaic systems. Republic of Korea, June 2020.

[4] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, and Jong Kim. Detecting and identifying faulty iot devices in smart home in context extraction. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Network*, Pohang, Korea, 2018.

[5] Ronald Christensen. Advanced linear modeling statistical learning and dependent data. In *ACM SIGMOD Record*, Edinburgh, Scotland, August 2019.

[6] Confluent. In *Kafka Streams Documentation*, August 2021.

[7] Dask. https://dask.org/. 2021.

[8] Wenfei Fan. Data quality: Theory and practice. In *ACM SIGMOD Record*, Edinburgh, Scotland, August 2012.

[9] Jerry Gao, Pengcheng Zhang, and Fang Xiong. Data quality in big data processing: Issues, solutions and open problems. August 2017.

[10] Mouzhi Ge, Stanislav Chren, Bruno Rossi, and Tomas Pitner. Data quality management framework for smart grid systems. Brno, Czech Republic, August 2012.

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren

Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. volume 585, pages 357–362. Springer Science and Business Media LLC, 2020.

[12] Benjamin T. Hazen, Christopher A. Boone, Jeremy D.Ezell, and L. Allison Jones-Farmer. Data quality for data science, predictive analytics, and big data in supply chain management: An introduction to the problem and suggestions for research and applications. April 2014.

[13] Mehrdad Heydarzadeh. A two-stage fault detection and isolation platform for industrial systems using residual evaluation. In *IEEE Transactions on Instrumentation and Measurement*, Texas, Dallas, June 2016.

[14] Rolf Isermann. Fault diagnosis. In *Fault-Diagnosis Applications:Model-Based Condition Monitoring Actuators, Drives, Machinery, Plants, Sensors, and Fault-tolerant Systems*, Berlin, Germany, 2011.

[15] Ivan Izonin, Natalia Kryvinska, Roman Tkachenko, and Krystyna Zub. An approach towards missing data recovery within iot smart system. In *The 16th International Conference on Mobile Systems and Pervasisve Computing*, Halifax, Canada, August 2019.

[16] Gonçalo Jesus. A dependability framework for wsn-based aquatic monitoring systems. Lisboa, Portugal, 2019.

[17] Gonçalo Jesus, António Casimiro, and Anabela Oliveira. Using machine learning for dependable outlier detection in environmental monitoring systems. In *ACM Transactions on Cyber-Physical Systems Volume 5 Issue 3 Article 29*, Lisboa, Portugal, July 2021.

[18] Joe-Air Jiang, Cheng-Long Chuang, Yung-Chung Wang, Chih-Hung Hung, Jiimg-Yi Wang, Chien-Hsing Lee, and Ying-Tung Hsiao. A hybrid framework for fault detection, classification, and location: Concept, structure, and methodology. In *IEEE Transactions on Power Delivery*, Texas, Dallas, August 2011.

[19] Kafka-Python. https://kafka-python.readthedocs.io/en/master/index.html.

[20] John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy. Fundamentals of machine learning for predictive data analytics. Massachussets, United States of America, 2015.

[21] kiba, Takuya, Sano, Shotaro, Yanase, Toshihiko, Ohta, Takeru, Koyama, and Masanori. Optuna: A Next-generation Hyperparameter Optimization Framework,

booktitle = Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, year = 2019, note = , month = , address = .

[22] Dubravko Miljkovic. Fault detection methods: A literature survey. In *MIPRO, 2011 Proceedings of the 34th International Convention*, Opatija, Croatia, May 2011.

[23] Tao Peng, Sana Sellami, and Omar Boucelma. Iot data imputation with incremental multiple linear regression. In *Open Journal of Internet of Things*, Marseille, France, February 2019.

[24] João Penim. Implementação de solucões para confiabilidade de dados em redes de sensores sem fios. Lisboa, Portugal, 2020.

[25] Memory Profiler Project. https://github.com/pythonprofilers/memory_profiler. December 2020.

[26] Pytest-Benchmark Project. https://github.com/ionelmc/pytest-benchmark. November 2020.

[27] Asha Radhakrishnan and Sarasij Das. Quality assessment of smart grid data. In *Proceedings of the National Power Systems Conference (NPSC)*, Tiruchirappalli, India, 2018.

[28] Fathi I. Salih, Saiful A.Ismail, Mosaab M. Hamed, Othman M. Yusop, Azri Azmi, and Nurulhuda F.M. Azmi. Data quality issues in big data: A review. In *Advances in Intelligent Systems and Computing*, Kuala Lumpur, Malaysia, September 2018.

[29] Fatimah Sidi, Payam Hassany Shariat Panahy, Lilly Suriani Affendey, Marzanah A.Jabar, Hamidah Ibrahim, and Aida Mustapha. Data quality:a survey of data quality. Malaysia, August 2013.

[30] Vaex. https://vaex.io/docs/index.html. 2021.

[31] Wes McKinney. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. editors: Stéfan van der Walt and Jarrod Millman.