POLITECNICO DI TORINO

College of Computer Engineering, Cinema and Mechatronics

Master's Degree Thesis

3D visualization and analysis of a large amount of real-time and non-real-time data

Supervisors
prof. Bartolomeo Montrucchio
dr. Antonio Costantino Marceddu

Candidate
Antonio Scaldaferri

April 2021

Summary

The scope of the thesis is to improve the PhotoNext project, whose purpose is to process the displacement and temperature information coming from the sensors of an airplane in order to display its status on a heat mapped 3D model, which can be shown through a desktop/AR application. The project is part of a collaboration with the DAUIN (Department of Control and Computer Engineering) and DIMEAS (Department of Mechanical and Aerospace Engineering) of Politecnico di Torino, as part of the POLITO Inter-Departmental Center for Photonic technologies (PhotoNext). The actual system is composed of different parts, which collaborates for reaching the scope:

- an interrogator, which is specialized hardware that generates UV lasers and reads the displacement/temperature information coming from the sensors;
- a middleware, which manages the incoming data from the sensors connected to the interrogator;
- a Desktop/AR application called Viewer, which displays those data simply and intuitively.

Before this thesis, the only way to connect the middleware with the Viewer was to use a Wi-Fi connection, which is extremely fast but has also a short-range. Therefore, it was decided to prefer a 4G/5G connection to send sensor data to a database in real-time and then communicate their addition to the Viewer. The advantage of this approach is that, since this data is now persistent, it can also be viewed later, therefore not in real-time.

The sensors of the airplane are of a specific type, called Fiber Bragg Grating (FBG) sensors. These can determine the variation of the refractive index of the optical fiber core caused by temperature, strain, pressure, tilt, displacement, acceleration, load, and more. FBG technology is already present in a wide range of fields, such as aeronautic [4], automotive, civil engineering [5], structure monitoring, undersea oil exploration, biomechanics, and rehabilitation applications [6] [7], and, because of their multiple properties like small size, lightweight, multiplexing capability, immunity to electromagnetic interference, the number of applications and fields where it will be used will increase in the next years.

The FBGs are enabled by an interrogator, which is specialized hardware that generates UV lasers useful to read the information coming from the sensors. Unfortunately, there is no standard protocol for communication with the interrogator, so each manufacturer equips their product with a proprietary solution. For the thesis, the SmartScan© from SmartFibres©, which is able to get information from 4 different fiber channels, each having 16 different gratings, has been used.

A C++ application, which acts as middleware, was created to retrieve the incoming data from the sensors connected to the interrogator. This multithread application can establish an Ethernet connection with the interrogator, and, once connected, can send the incoming data to a MongoDB database or directly to the Desktop/AR application via TCP-IP. It runs on a Raspberry Pi 4 Model B, which is a tiny and powerful computer board with Linux Operating System and, because of it, it has an easy configuration for multiple connections.

For data storage, it was needed a database with fast writes and a built-in notification framework. After careful investigation, the choice made was MongoDB, which is a NoSQL database known for its high read/write performance, its high data availability, and its data stream update

provided by the Change Stream technology. Thanks to this feature, multiple applications can subscribe to all data changes on a single collection, on a database, or an entire deployment, and immediately react to them. In the scope of the project, the middleware writes data coming from the sensors to the MongoDB instance and the Viewer subscribes to the real-time feed provided by Change Stream, reacting in real-time to the newest values.

The Viewer is a 3D application developed in Unity. This application gets real-time and non-real-time data coming from the sensors and it displays them as a heat map on a 3D model or a line graph. For real-time data, it is possible to receive data directly from the middleware through a TCP-IP connection or it can subscribe to a MongoDB instance, receiving the real-time sensors data using the Change Stream technology. Instead, for non-real-time data, it can read the information of the sensors from a MongoDB collection.

Due to the current pandemic, it was not possible to perform outdoor tests with a full flight of the airplane. Instead, there were conducted laboratory tests as proof of concept with multiple FBGs, analyzing the strength and the bottleneck of the proposed framework. The test scenarios are represented by an aluminum flexible bar and a thermometric probe, both equipped with FBG sensors. The focus is to test both the reliability and availability of the platform while maintaining stable real-time monitoring. Test results showed that the platform is compliant with the high data transmission speed. The application provides uninterrupted sensor information maintaining an acceptable near real-time experience.

The entire framework is designed to analyze and monitor an aircraft engineered by the ICARUS team in Politecnico di Torino, for the scope of their project called "ANUBIS". The sensors can detect the bending of structural elements of the airplane and the temperature in specific spots. The deployment performs as expected, however, it still requires fixes and improvements. Network connectivity between the components was designed to support 4G/5G but, due to COVID pandemic, has been set aside in favor of the Wi-Fi network. The interrogator library could be improved in order to enhance data throughput and system stability. In particular, an abstraction layer could be implemented to allow the use of different interrogators.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, a brief overview of the thesis will be provided, followed by some notions on the involved technologies: they will be useful to better understand the following chapters and the tests performed.

1.1 Overview

The scope of the thesis is to improve the PhotoNext project, whose purpose is to process the displacement and temperature information coming from the sensors of an airplane to display its status on a heatmapped 3D model, which can be shown through a desktop/AR application. It is an enhancement of a previously developed solution that involves IoT elements [1] and VR/AR monitoring system [2] and will present a solution that integrates the two approaches in a scalable and reusable way. The project is part of a collaboration with the DAUIN (Department of Control and Computer Engineering) and DIMEAS (Department of Mechanical and Aerospace Engineering) of Politecnico di Torino, as part of the POLITO Inter-Departmental Center for Photonic technologies (PhotoNext). The entire framework is designed to analyze and monitor an aircraft engineered by the Innovation Center for Amateur Rocketry and Unmanned Ships (ICARUS) Team of the Politecnico di Torino, for the scope of their project called "ANUBIS". The sensors can detect the bending of structural elements of the airplane and the temperature in specific spots. The actual system is composed of different parts, which collaborates for reaching the scope:

- an interrogator, which is specialized hardware that generates UV lasers and reads the displacement/temperature information coming from the sensors;
- a middleware, which manages the incoming data from the sensors connected to the interrogator;
- a Desktop/AR application called Viewer, which displays those data simply and intuitively.

The sensors of the airplane are of a specific type, called Fiber Bragg Grating (FBG) sensors. These sensors can determine the variation of the refractive index of the optical fiber core caused by temperature, strain, pressure, tilt, displacement, acceleration, load, and more. FBG technology is present in a wide range of application, such as aeronautic [4], automotive, civil engineering [5] structure monitoring, undersea oil exploration, biomechanics and rehabilitation applications [6] [7]. The number of applications will increase in the next years thanks to their properties, like small size, lightweight, multiplexing capability, and immunity to electromagnetic interference.

FBG sensors are powered by specific hardware, called interrogator, that generates UV lasers useful to read the information coming from the sensors in form of wavelength, which can then be sent to the next layer of the architecture via an Ethernet connection. The interrogator used during the thesis is SmartScan© from SmartFibres©, which can get information from 4 different fiber

channels, each having 16 different gratings. It has been connected via Ethernet to the Middleware, which is C++ application hosted in a Raspberry Pi 4 Model B.

The Middleware elaborates the data coming from the interrogator and stores them in a MongoDB instance hosted in a Ubuntu Server. MongoDB has been chosen as data storage because of its fast write time and its notification feed "Change Stream". Change Stream is a functionality that provides a real-time feed of all the changes happening in a database instance. This feed is useful in the scope of the thesis because the VR application can be notified of the new data coming from the sensor without the risk of tailing the operation log of the database.

The Viewer is a 3D real-time application developed in Unity and C#. It can subscribe to real-time or non-real-time data after connecting to the remote database. It enables the visualization of a 3D model, where the user can virtually place the different sensors. Moreover, it shows in the 3D model itself the strain applied to the sensors with different gradients, and, in a graph, the real-time wavelength value of the sensors. After the simulation, it creates via Python a new graph that shows the data collected during the test. In figure 1.1, an overview of the architecture of the project is displayed.

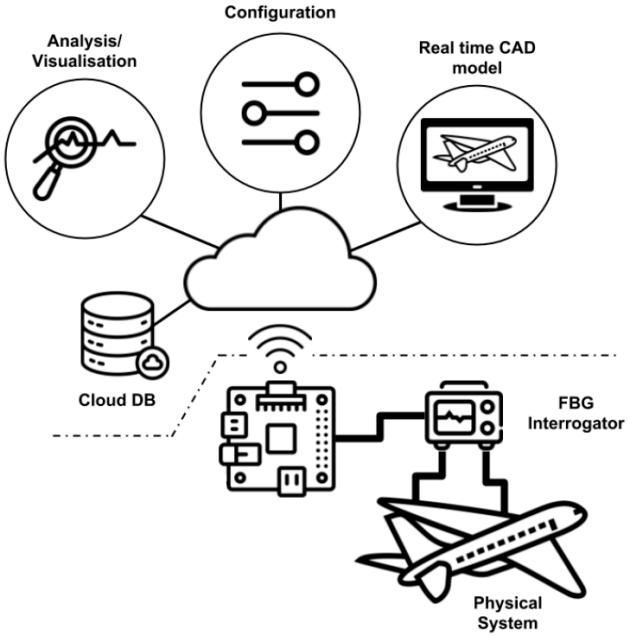

Figure 1.1. Architecture of the system [1] [2].

1.2 FBG technology

Fibre Bragg Grating (FBG) sensor is defined as a "very high sensitive and versatile optical device for measuring several physical parameters including for example strain, temperature, pressure, vibration, and displacement" [8]. The fiber is enabled by a UV laser beam and the sensor works as a narrow band filter. The FBG has a symmetric structure that allows reflecting the light at the Bragg wavelength no matter the direction of the light itself. One of its big advantages is its intrinsic multiplexing capability. This means that with only one single optical source it can address multiple sensors in the same fiber with only minor loss and no crosstalk, as long as enough spectral band of the spectrum of light is reserved for each sensor.

In the scope of this project, FBGs sensors are used to monitor strain and temperature. In fact, when a fiber is stretched or compressed, it is possible to detect the deformation as a change of the microstructure and of the Bragg wavelength (figure 1.3). In the case that the temperature

change, also the Bragg wavelength change thanks to the variation of the silica refraction index induced by the thermo-optic effect, as can be seen in figure 1.4.

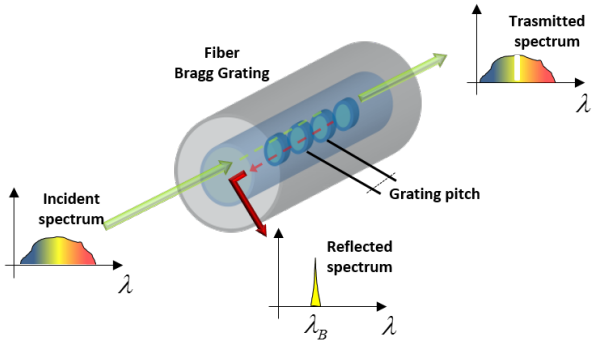

Figure 1.2. FBG structure [8].

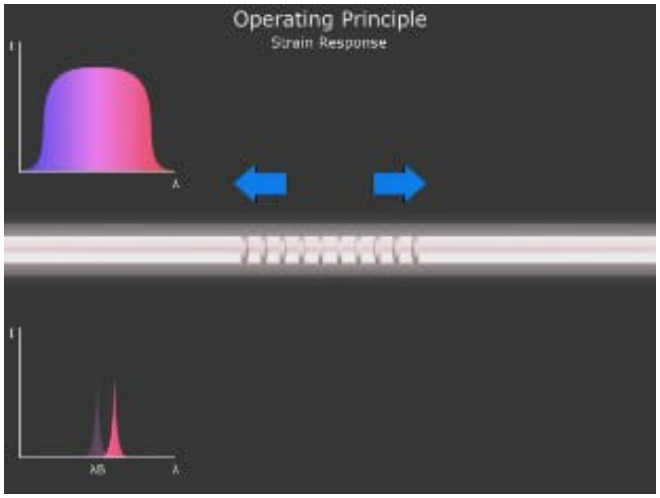

Figure 1.3. FBG strain graph [9].

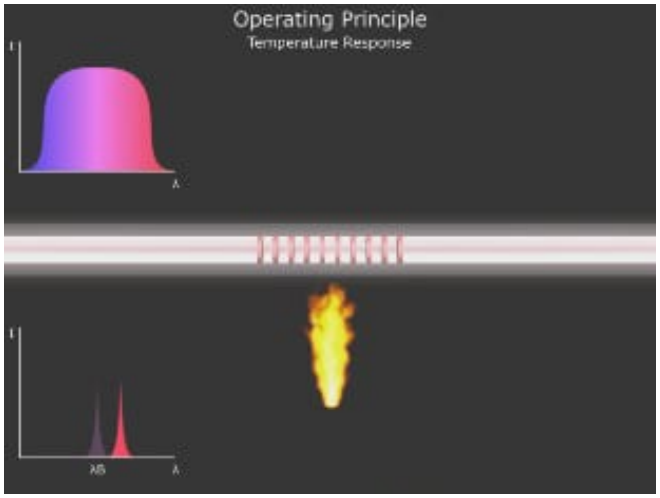

Figure 1.4. FBG temperature graph [9].

Bragg wavelengths are defined as:

$$\lambda_B = 2n_{ef}\Lambda[9]$$

where $2n_ef$ is the effective core refractive index of the fundamental mode and Λ is the grating pitch. Any change in refractive index of the sensor can be calculated with the following formula:

$$\frac{\Delta\lambda}{\lambda_0} = \frac{\Delta(2n_{ef}\Lambda)}{2n_{ef}\Lambda}[9]$$

If a strain is applied to the fiber, the change in refractive index becomes as follows:

$$\frac{\Delta\lambda}{\lambda_0} = \frac{\Delta(2n_{ef}\Lambda)}{2n_{ef}\Lambda} = (1 + p_e)\Delta\varepsilon = k\Delta\varepsilon[9]$$

where k is the k factor of the Bragg grating and p_e is the photoelastic constant (variation of the refraction index with axial tension).

If the temperature changes, the change in refractive index can be calculated as follows:

$$\frac{\Delta\lambda}{\lambda_0} = \frac{\Delta(2n_{ef}\Lambda)}{2n_{ef}\Lambda} = (\alpha + \xi)\Delta T[9]$$

where α is the coefficient of thermal expansion of the fiber and ξ is the thermo-optic coefficient (dependence of the index of refraction on temperature).

1.3 Thesis structure

This thesis is divided in six chapters:

- the chapter 2 summarizes the system architecture;
- the chapter 3 describes the solution proposed in this thesis;
- the chapter 4 can be seen as a user and developer guide to help to perform the deployment of the system;
- the chapter 5 presents the tests performed and their outcome;
- the chapter 6 contains some ideas for future development and the conclusions about the work made.

Chapter 2

System Architecture

The following chapter describes the system architecture, focusing on the current design and the interconnections between the modules.

2.1 System overview

The thesis purpose is to enhance the existing PhotoNext project. The framework is composed by an interrogator, specialized hardware that reads the displacement/temperature coming from FBG sensors. The system was integrated into a non-real-time monitoring system using an IoT Cloud platform (KaaIoT) [1] and in a real-time application using Hololens [2]. The focus of the thesis is to create a software layer that combines the real-time monitoring of the desktop application while collecting sensors data remotely in a Server (figure 1.1). The system is composed of the following elements:

- Physical System;
- Interrogator;
- Middleware;
- Cloud network/Server;
- Desktop/AR application.

In the next sections, each component will be analyzed, with the focus on what has been enhanced during the thesis.

2.2 Physical system

The physical system consists of an aircraft monitored by a set of FBG sensors. The aircraft was built by the Innovation Center for Amateur Rocketry and Unmanned Ships (ICARUS) Team (figure 2.1), while study, validation and dislocation of the FBG sensors on the aircraft is out of the scope of this thesis, as they will be coordinated by Department of Mechanical and Aerospace Engineering (DIMEAS).

2.3 Interrogator

The interrogator is a device capable to read the information from FBG sensors. There are multiple solutions available in the market: for the thesis, the SmartScan© from SmartFibres© has been

Figure 2.1. Aircraft built by ICARUS team for the Air Cargo Challenge [10].

chosen. It provides both raw and peak data and it is capable of reading 64 differents FBGs (4 channels, 16 gratings per channel). This interrogator have a RJ45 Ethernet Connector with built-in network status lights, a Serial connector for diagnostics/servicing and 2 USB ports. SmartFibres provides also a Microsoft Windows application called SmartSoft Application Software©, which communicates with the interrogator via UDP.

Figure 2.2. SmartScan© from SmartFibres© [11].

2.4 Middleware

The middleware is the core of the project and consists of a C++ application hosted in a Raspberry Pi 4 Model B (figure 2.3). It is connected via Ethernet to the interrogator, from which retrieves the sensors data and elaborates them in different ways. In particular, the first iteration [1] was able to reinterpret data and inject them in KaaIoT, an open source IoT platform. The second iteration [2], instead, managed a TCP connection with the Unity application in order to visualize

those data in a simple manner. Both solutions are designed to handle raw and wavelength data at high speed, thanks to an optimized multi-threaded application. Moreover, at each measurement there are associated metadata, like time-stamp and position of the sensors. Those information are crucial for all types of monitoring, both real-time and non-real-time.

In the thesis, the Middleware is in charge to store the wavelengths and the metadata. Moreover, the application has been modified in order to be more modular for future iterations, which could add the support to other interrogators or different monitoring systems. The design of the information to store is very important as it optimizes bandwidth and increases throughput.

Figure 2.3. Raspberry Pi 4 Model B [13].

2.5 Cloud network - Server

The cloud network was implemented on KaaIoT platform, an open-source framework that supports various relational and non-relational databases. The strength of this cloud service is to create multiple micro-services that are customizable and easy to use [14]. After the setup of an ID and the creation of a model for the message, it is possible to store all the information in the cloud at high speed as it allows multiple insert operations on different databases at the same time. Moreover, it provides tools to analyze data thanks to the integration of Apache Zeppelin, a web-based notebook which allows to perform offline analysis of the received data by providing different type of graphs. Since the open-source version was deprecated years ago, in this thesis it has replaced by a data storage system powered by MongoDB, which offers high-speed insertion and easy management.

2.6 Viewer

The Viewer is a Desktop/AR Application developed in Unity, useful for displaying real-time and non-real-time data. It can work with multiple configurations, as it can read data from a MongoDB instance or from a direct TCP-IP connection with the Middleware. After importing a custom 3D model, it will show in the model itself the information about the temperature and wavelength with a certain level of gradient and, moreover, it will plot those information on a graph. In the thesis, the functionalities were enhanced in order to create a new real-time functionality based on the MongoDB Change Stream feed (figure 2.4).

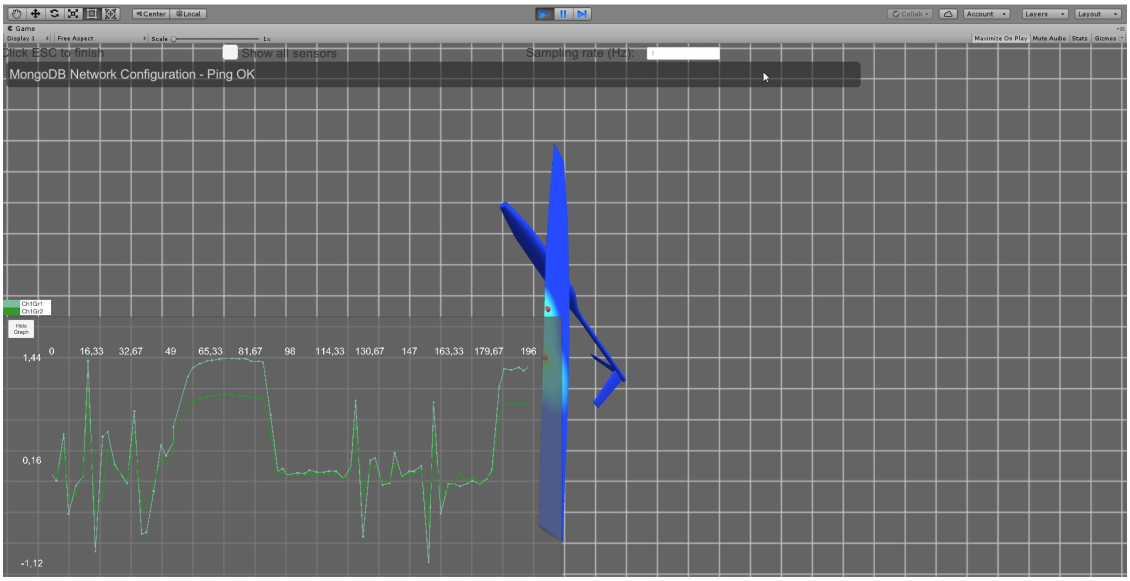

Figure 2.4. The Viewer.

Chapter 3

Proposed Solution

In this chapter, the solutions developed for the thesis will be presented, with the focus on technologies and methodologies put in place.

3.1 Overview

For the sake of simple scalability, The layered structure shown in the previous chapter has been kept. Because of technological constraints, the cloud solution for IoT based on KaaIoT has been discarded. Instead, a simple and reliable solution, which uses an in-house Server with a MongoDB instance, will be proposed. This solution is easy to setup and it will allow the storage of the data in any server solution which recurs to any Linux distribution. Then, using MongoDB as data storage will give the benefit of using the Change Stream, which is a feature that allows applications to access real-time data changes without the complexity and risk of tailing the operation log (also called oplog).

The other updated layer is the Middleware, which has the functionality of writing the data stream into a MongoDB instance. To achieve that, the code has been refactored and it was added an activation parameter, "-noDB", for coming back to the previous solution, which recurs to the TCP-IP connection. For what regards the Desktop/AR application, the focus was to improve the functionality of the Unity framework with the possibility of subscribing to a Change Stream, which allows showing the information coming from the sensors in real-time. In the following sections of this chapter, those technologies will be presented in detail.

3.2 Physical System

The physical system is the unity that will be monitored by a set of FBG sensors. The need for monitoring can be various, from real-time structural health monitoring of the data coming from the sensors to an off-line prevention strategy for identifying structural issues in a system achieved by storing sensor data. The system can be any system that needs a constant and reliable measure of strain or temperature in key points of its structure.

The scenario of the thesis is an aircraft. The sensors are placed at strategic points to monitor the bending and the temperature of the aircraft itself during a flight. FBG sensors will be placed in an Unmanned Aerial Vehicle (UAV) developed by the ICARUS Team of the Politecnico di Torino by the DIMEAS Department. In this scenario is required to retrieve data from the sensors at a high-speed rate in order to detect anomalies. The real-time case requires monitoring the status during the flight itself, while the non-real-time case requires storing the data of the flight and providing a framework for data analysis.

Figure 3.1. Aircraft simulation by Icarus team [10].

3.3 Interrogator

As described in section 2.3, the interrogator is an FBG sensing instrument able to detect and measure any changes in the refractive index of the wavelength signal sent through the optical fiber link. Different interrogators are available in the market, which can differ in scan speed, supported range of spectrum wavelengths and physical parameters, such as robustness in harsh environments or EMI isolation. Interrogators can provide different types of sensor data, like raw data regarding, for example, the intensity of the reflected wavelength in the spectrum interval or the peak data derived from the wavelength raw data.

The selected device for the thesis is SmartScan© from SmartFibres©. It is capable of retrieving raw and peak data for a maximum of 64 FBG sensors, distributed in 4 channels with 64 gratings per channel. SmartFibres equips his interrogator with a dedicated monitor and analysis software called SmartSoft© (figure 3.2).

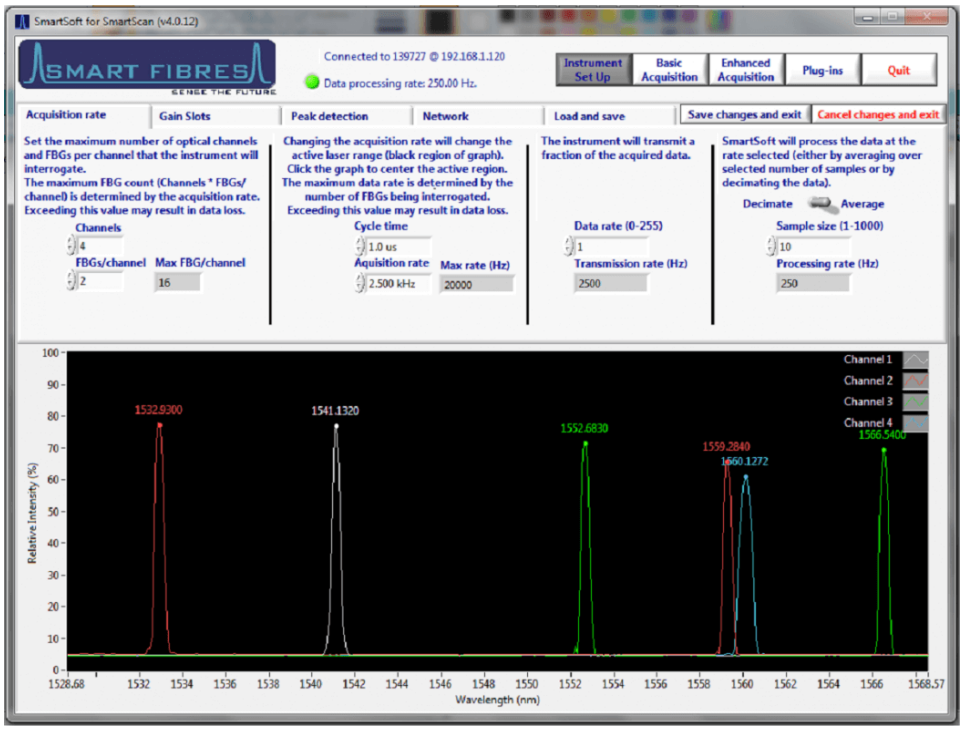

Figure 3.2. SmartSoft© by SmartFibres© [12].

To extract information from the interrogator, it is not possible to rely on proprietary software. In previous works [1] two libraries were developed:

- the first is in charge of the communication with the interrogator for retrieving sensors data and recurs to the UDP protocol provided by Smartfibres© other than the default SBI reader module.
- the second is in charge of reading data directly from the Ethernet connection. Thanks to the library, it is possible to extract 2 types of data from the interrogator, **peakData** or **rawData**, whose data structure will be described in the following sections.

3.4 Middleware

The core of the project is the middleware. It is "responsible for most of the intelligence in IoT, integrating data from devices, allowing them to communicate, and make decisions based on collected data" [16]. It is a multi-thread C++ application hosted on a Raspberry Pi 4 Model B, that communicates on one side with the *interrogator* via Ethernet, and on the other with the *Viewer* via Wi-Fi. The scope of the thesis is to enhance its functionalities, integrating the possibility to store all the data retrieved from the sensors on a MongoDB instance. Inserting sensors data in this database, it is possible to notify the Viewer of each new data registered in real-time. This feature is provided by MongoDB with its Change Stream. Moreover, the structure of the application has been refined to split the communication with the Viewer into two functionalities.

3.4.1 Technologies and Dependencies

As MongoDB has been chosen as a storage database, the application needs to be able to connect to it in order to perform CRUD operations. To achieve that, the MongoDB community has released a library called mongocxx, which provides C++ applications the basic functionalities for managing the database. The iteration "mongocxx 3.6.x", which has a dependency on a C library called "libmongoc 1.17.0", has been used for this thesis. Both of them are not provided with a standard Linux Distribution because of licensing issue (see 3.5), so a simple installation guide is presented in Chapter 4.

3.4.2 Modules and Refactoring

To make an easy-to-read and maintainable code, the functionalities of the middleware have been refactored into 3 main modules:

- Client;
- TCP Manager;
- MongoDB Manager.

The first one is in charge of the main thread. It reads the given inputs and instantiates the appropriate threads. The only input that can be received is "-noDB", which enables the TCP-IP workflow. Otherwise, if no inputs are given, it will initialize the MongoDB workflow. The second module is the TCP Manager: it reads the peak data from the *SmartScanInterrogator*, elaborates them and creates a new TCP connection with the *Viewer* for sending, every 8 ms, the timestamp and the wavelength of each sensor. Moreover, it will store the sensor configurators, ready to be sent to the *Viewer* when needed. The last module is the MongoDB Manager, which reads the information of the peak data from the SmartScanInterrogator and elaborates them, as the TCP Manager. Moreover, it establishes a client instance with MongoDB on a given URL and starts to store sensors data in a new collection with the name format YYYYMMDDHHMMSS. In the next sections, the different flows are explained in detail.

TCP Manager

This module is inherited from previous works [2]. In the scope of the thesis, this functionality has been refactored to make the code more scalable. It is capable to read peak information data from the interrogator and store them in a vector of wavelength and timestamp for each channel and grating. This mode is enabled running the middleware with the option "-noDB". TCP Manager instantiates 2 different threads to create a new TCP Client and Server connection.

The Server creates a TCP connection with the Unity application and waits for messages. There are 4 types of message that the middleware can receive from the TCP Manager:

- REQUEST_CONFIG: after receiving this message, the middleware sends the configuration of the sensors;
- REQUESTDATA_START: received when the simulation starts on the Viewer, it enables the forwarding of the peak data;
- REQUESTDATA_END: received when the simulation is terminated o the Viewer, it disables the forwarding of the data without closing the server;
- ENDTHREAD: it closes the connection and shuts down the middleware.

The Client, instead, stores the data coming from the sensors and, when the application unlocks the communication, it starts to send the wavelengths with their timestamps at intervals of 8000 microseconds (125 Hz) via TCP. The full flow is described in figure 3.3.

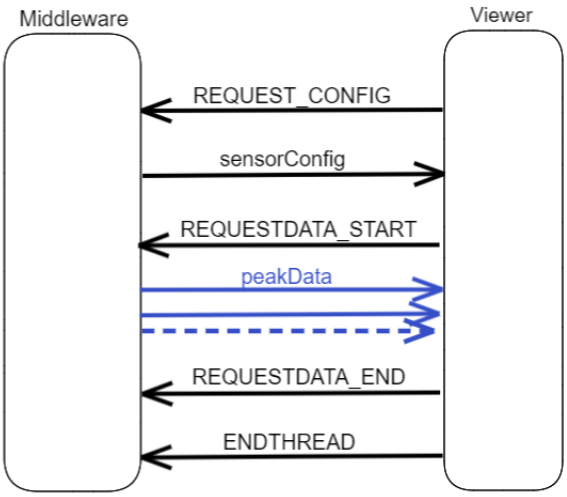

Figure 3.3. Communication flow for TCP-IP mode. Server messages are highlighted in black, while the Client messages are highlighted in blue.

MongoDB Manager

The MongoDB Manager is implemented with a class that is able to connect with an online instance of MongoDB. If not already present, it creates a database instance called "photonext". When the simulation starts, the software stores the sensors data into the database: in particular, they are stored in a new collection of data called "YYYYMMDDHHMMSS", which represents the timestamp in the local time zone of the experiment. There are two types of documents that can be stored in the database: configuration data and peak data. In this flow, there is no connection between the middleware and the viewer.

```
1  class mongodbManager
2  {
3  public:
4      mongodbManager(std::string collname);
5      void insertPeakData(peakData pd);
6      void insertUnityData(int index, uint64_t timestamp, float wav, uint64_t
           curr_time);
7      void insertConfig(sensorConfig config);
8      void listDB();
9      void showDB();
10 };
```

Figure 3.4. mongodbManager class definition.

This solution is more scalable than the previous one because of the new independent layer, but it adds a delay to the whole system. The new delay is due to the insert time into the database and the notification delay of the Change Stream. To improve the response time of the system, it is important to reduce the amount of information stored in the database. Knowing the configuration of the sensors, it is possible to occupy less bandwidth with only the essential information, without throttle the connection. Unfortunately, the bottlenecks of the solution remain the database insertion time and the connection ping. All the tests and analysis of the delay are described in Chapter 5.

To manage all the CRUD operations into the database, a new class in C++ with dependencies on mongocxx 3.6.x has been implemented using libmongoc 1.17.0 libraries. The definition of the class is represented in figure 3.4.

When the class is instantiated through the constructor, it establishes the connection with the database using the given IP and database name "photonext". Moreover, it creates a new collection with the name passed as a parameter. The methods of the class are:

mongodbManager(std::string collname): constructor of the Class, it creates a client connection with the MongoDB instance using the collection with the name passed as a parameter.

void insertPeakData(peakData pd): this function inserts raw peak data [see 3.4.3 for the definition] into the collection "collname".

void insertUnityData(int index, uint64_t timestamp, float wav, uint64_t curr_time): this function inserts peak data into the database in a format easy to decode for the viewer. It takes as input the index of the sensor, the timestamp of the wavelength coming from the interrogator, the value of the wavelength and the time of the insertion [see 3.4.3 for the complete definition].

void insertConfig(sensorConfig config): this function inserts into the database the configuration data of the sensors. See section 3.4.3 for the definition of the structure "sensorConfig". This function is called only one time at the beginning of the simulation.

void listDB(): support method, it is used for debugging purposes. It lists the databases available in the MongoDB instance through the standard output.

void showDB(): support method, it is used for debugging purpose. It prints all the documents in the selected collection in the standard output.

After the creation of the client connection and the collection, the application starts to get sensors data from the interrogator. The data are provided with a thread-safe function that implements a Producer-Consumer model. The interrogator libraries act as a Producer in a buffer, inserting the data coming from the sensors in the format peakData (details in 3.4.3), while the

main thread runs as a Consumer, reading the peakData from the buffer, manipulating them and inserting them into the database (details in 3.4.3). This flow is represented in figure 3.5. As the figure shows, the Viewer is not represented in this flow. This is achieved by design because it is possible to collect data autonomously and visualize the data offline on the Viewer itself.

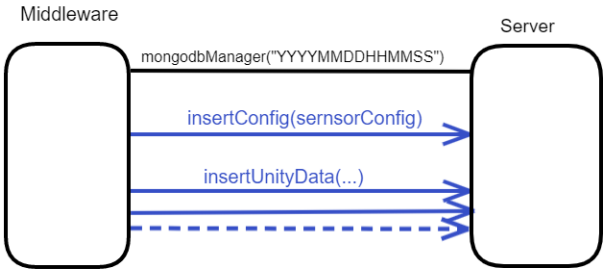

Figure 3.5. MongoDB Manager flow. The creation of the new collection is highlighted in black, while the insertion operation is highlighted in blue.

3.4.3 Data Models

In this subsection, the data models used on the Middleware application and the Interrogator library are described in detail. The library provides two types of data models coming from the sensors: rawData and peakData. In the middleware, those data are elaborated into Unity Peak Data and stored in the database. The last type of data model is the "Configuration Data", useful for the setup of the sensors into the viewer.

rawData

Raw data are the raw information coming from the interrogator. The definition of the structure is represented in figure 3.6 and in details:

- uint32_t rd_timestamp_sc: represents the seconds in UNIX epoch time of the current raw data;
- uint32_t rd_timestamp_fr: represents the fractions of second in UNIX epoch time of the current raw data;
- uint16_t rd_data: represents the value of the raw data coming from the sensors;
- uint16_t rd_slot: represents the metadata of the current raw data.

```
1 typedef struct rawData
2 {
3     uint32_t rd_timestamp_sc ; // seconds
4     uint32_t rd_timestamp_fr ; // fraction
5
6     uint16_t rd_data ; // scan data
7     uint16_t rd_slot ; // scan data slot
8
9 } rawData ;
```

Figure 3.6. rawData struct definition.

```
1  typedef struct peakData
2  {
3      uint32_t pd_timestamp_sc; // seconds
4      uint32_t pd_timestamp_fr; // fraction
5
6      uint16_t pd_data; // peak data
7      double pd_wavelength; // wavelength
8      uint8_t pd_channel; // channel number
9      uint8_t pd_grating; // grating number
10
11 } peakData;
```

Figure 3.7. peakData struct definition.

peakData

Peak Data is a data model inherited from the interrogator library. Its definition is represented in the code on figure 3.7 and in details:

- uint32_t pd_timestamp_sc: represents the seconds in UNIX epoch time of the current peak data;
- uint32_t pd_timestamp_fr: represents the fraction of seconds in UNIX epoch time of the current peak data;
- uint16_t pd_data: represents the value of the peak data of the simulation;
- double pd_wavelength: represents the value of the wavelength of the current peak data;
- uint8_t pd_channel: is the channel of the current peak data. The system can have 4 channels maximum;
- uint8_t pd_grating: is the grating of the current channel of the FBG. The system can have 16 gratings per channel maximum.

This data represents the raw information coming from the sensors. That information is elaborated by the Middleware into "Unity Peak Data" and inserted into the database.

Unity Peak Data

Unity Peak Data is a representation of the information of the sensors. The design takes into account the experimental value of the sensors data with a look at the communication throughput with the database. The model is represented in figure 3.8 and in detail:

- string type: indicates the type of data present in the document. The type can be "peakData" for Unity Peak Data or "config" for Configuration Data;
- unit64_t curr_time: represents the UNIX epoch timestamp (seconds and fractions) at the time of the insertion of the data. This information is useful to calculate the elaboration delay of the application;
- int index: represents the index of the current sensor. It has been calculated as $(channel * 16 + grating)$ because the system is capable of carrying 64 FBG sensors (4 channels, 16 gratings per channel);

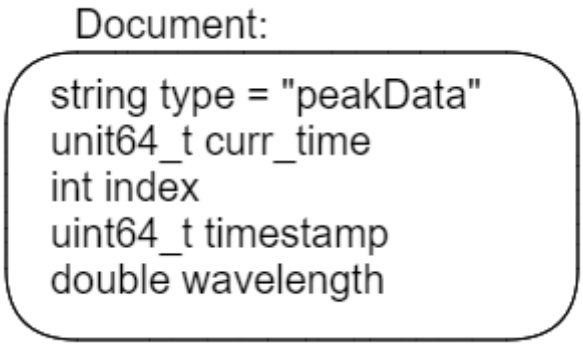

Figure 3.8. Unity Peak Data document model.

- uint64_t timestamp: is the timestamp coming from the detection of the peak data. This is elaborated as follows: $(pdTimestampSc) * 1000000 + (pdTimestampFr)$, where pdTimestampSc corresponds to the raw peak data "pd_timestamp_sc" and pdTimestampFr corresponds to "pd_timestamp_fr";
- double wavelength: is the value of the wavelength of the sensor coming from the peakData.

This data model represents the data injected into the database as a BSON document.

Configuration Data

This data represents the configuration of the FBGs in the physical system. The interrogator is capable of managing 64 FBGs, distributed in 4 channels with 16 gratings each. The data structure, expressed in figure 3.9, is designed for interrogators that can give more information about the sensors. Unfortunately, the only information retrieved from the library is the peakData of each sensor.

```
1  struct vec3 {
2      float x, y, z;
3
4      vec3(float _x, float _y, float _z) : x(_x), y(_y), z(_z) {};
5  };
6
7  struct sensorConfig {
8      uint8_t channel, grating;
9      bool is_active;
10     vec3 position;
11     float wavelength_idle;
12     float wavelength_var;
13
14     sensorConfig(uint8_t c, uint8_t g, bool ia, vec3 pos, float wi, float wv)
           :
15         channel(c), grating(g), is_active(ia), position(pos),
16         wavelength_idle(wi), wavelength_var(wv) {};
17 };
```

Figure 3.9. Config data struct definition.

The client fills this data structure with the first peak data received from the interrogator. The parameters are as follows:

- uint8_t channel: is the channel of the current peak data the system can have a maximum of 4 channels;
- uint8_t grating: is the grating of the current channel of the FBG. The system can have a maximum of 16 gratings per channel;
- bool is_active: boolean that indicates if the current sensor (channel and grating) is active or not. If the value is "true", the FBG is active, "false" otherwise;
- vec3 position: it contains the tridimensional position of the sensor on the airplane 3D model. This data is filled with (0,0,0) at the moment of the acquisition;
- float wavelength_idle: it contains the idle value of the sensor wavelength. Unfortunately, it is not possible to retrieve this value from the interrogator, but it needs to be calculated in the Viewer.
- float wavelength_var: it contains the variance of the sensor wavelength. Unfortunately, we cannot retrieve this value from the interrogator, but it will be calculated in the Viewer.

This information will be stored in the database as a BSON document, with the model expressed in figure 3.10. The structure is similar to the one seen before, with the only addition of the attribute "type" that is set with the value "config". This data structure is filled at the beginning of the simulation and needs to be retrieved by the Viewer before reading the peak data.

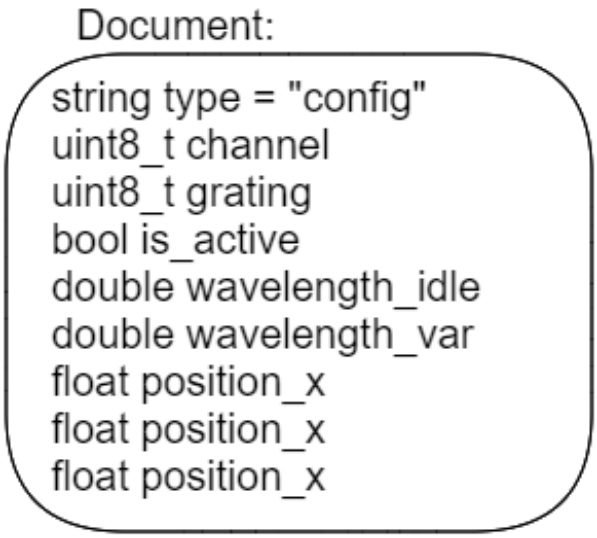

Figure 3.10. Config database Data model.

3.5 Server

This section describes the technologies implemented into the storage Server. The Server deployed in the thesis is an instance of Ubuntu Server 20.04, while database technology implemented in the server is MongoDB. The choice of Ubuntu Server OS is due to licensing issues for Mongo distribution: the entire MongoDB suite is licensed under Server Side Public License (SSPL), and for that reason has been removed from the Debian, Fedora, and Red Hat Enterprise Linux distributions. The next sections will describe the technologies put in place for the thesis, with a focus on the features provided by MongoDB.

3.5.1 MongoDB

MongoDB is an industry-standard NoSQL database, source available, and document-based software, developed by MongoDB Inc. It uses for the storage of proprietary JSON-like documents, called BSON. MongoDB provides various features, such as:

- High Performance, thanks to the support for embedded data models and indexes;
- Rich Query Language, providing CRUD operation as well as Aggregation, Text Search and Geospatial; Queries;
- High Availability, with replica set that provides automatic failover and data redundancy;
- Horizontal Scalability, thanks to Sharded Clusters;
- Support for multiple storage engines and different file systems.

The main features used in the thesis, like indexes, replica set, and Change Stream will be described in detail in the rest of the chapter.

3.5.2 Indexes

MongoDB indexes are a B-tree data structure that allows an efficient query execution. Without them, MongoDB has to perform a full collection scan for each query executed. MongoDB provides various type of indexes:

- Single Field Indexes: provide a sorted structure in ascending or descending order. Those are useful to achieve fast search for any field in the database;
- Compound Indexes: provide a structure to combine multiple single field indexes;
- Multikey Indexes: allow queries to select documents that contain arrays by matching on element or elements of the arrays;
- Geospatial Indexes: are useful in case of geospatial coordinate data;
- Text Indexes: provide support for string base fields;
- Hashed Indexes: provide the hash of the value of a field. These indexes have a random distribution of values along with their range, but they only support equality matches and cannot support range-based queries.

3.5.3 Replica Set

In MongoDB, a replica set is a group of MongoDB processes that maintain the same data set [15]. This feature provides redundancy and increases data availability. In particular, there is a primary node that is the only one capable of performing write operations on the database and recording them on the *oplog* after the commit. The oplog, or operations log, is a capped collection that keeps a rolling record of all operations that modify the data stored in the databases. The secondary processes will asynchronously read the operations from the oplog updated by the primary and will replicate the same operations in their own data sets. Thanks to this feature, MongoDB can create data redundancy and provides automatic failover. If the primary is unavailable, a secondary instance can call an election to elect itself or another secondary as a new primary. There are various ways to setup a replica set: in Chapter 4, a full guide on how to setup a MongoDB instance in the server will be presented.

3.5.4 Change Stream

The Change Stream is a MongoDB feature that allows different client applications to access real-time data changes without the complexity and risk of tailing the oplog. This feature is available for replica sets and sharded clusters since MongoDB 3.6. Thanks to that, it is possible, for an application, to subscribe to all data changes on a collection. To subscribe to this feed, the desktop application needs to create a connection via the C# MongoDB driver and open a change-stream pipeline for the collection indicated. It is possible to interact with the collection using an iterable cursor that reacts as soon as something changes in the pipeline. It is possible to control change stream output by providing an array of one or more of the following pipeline stages in the configuration of the change stream:

- $addFields;
- $match;
- $project;
- $replaceRoot;
- $replaceWith;
- $redact;
- $set;
- $unset.

Thanks to that, in the thesis it was possible to optimize data transmission and bandwidth of the stream, minimizing the information with only the one needed by the Desktop/AR application. The events implemented in the feature are all CRUD operations, such as insert, update, replace, delete, drop, rename, dropDatabase, invalidate. The project was optimized by using the only insert Event of the Change Stream in a given collection created by the middleware, to give real-time feedback from the database.

3.6 Viewer

The Viewer is a Desktop/AR application developed in Unity. It has been developed as a Windows application, but it can be ported in multiple OS thanks to the multi-platform deployment provided by the tool. The software provides user-friendly support for motoring and analyzing the data coming from the sensors. The monitoring is achieved through a dynamic graph and a heatmap which shows the actual sensors data in an imported 3D model of the aircraft. Moreover, it offers the possibility to personalize the 3D model of the simulation, change the position of the sensors on it, and customize the color gradient of the heatmap.

The application can retrieve sensors data using 3 different flow types:

- TCP-IP: establishing a TCP-IP connection with the middleware;
- Real-Time: using Change Stream on a MongoDB instance;
- Non Real-Time: reading previously recorded data from a MongoDB instance.

In the next subsections, a walk-through of the application, explaining the functionality developed in the solution, will be presented.

3.6.1 Overview

The home view of the Desktop/AR application, which can be seen in figure 3.11, is divided into 3 main zones:

- the *Import Model* menu, marked in red, can be found on the left. It offers the functionality for importing a model of the device to monitor in .obj format and, for the sake of simplicity, it also proposes the possibility to import a default model of an airplane;
- the center of the home view is devoted to the visualization of the 3D model. This part is only interactable during the simulation or the configuration of the sensors and the model;
- on the right side the *Configuration Menu*, which is divided into 3 subareas, can be found:
  - on the top-right, marked in green, there is the *Change HeatMap Color* menu, where a collection of preset gradients and a plus sign button for adding more can be found;
  - the *Server/Network Configuration* menu is marked in blue. It can be used to configure the connection necessary to retrieve sensors information via a drop-down menu containing three different entries: TCP-IP, Real-Time, and Non-Real-Time. The input form will give the possibility to configure the connection according to the choice made in the drop-down;
  - the *Sensor Configuration* menu, marked in orange, can be found on the right-bottom. It can be used to retrieve the configuration of the sensors from the database or the Middleware and modify their properties, such as the position on the model, the idle, and the variation of the wavelength.

During each simulation, a Python tool creates 2 types of .csv logs: one with the peak data retrieved from each sensor and one with the time delay of the data retrieved. Moreover, from the first file, the software builds a graph with all the wavelength received in the simulation.

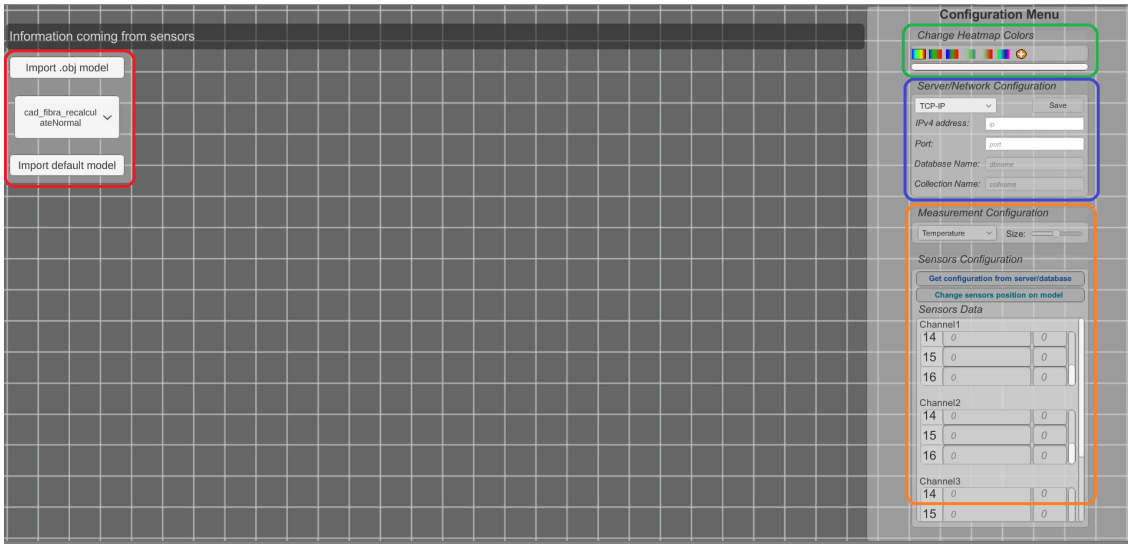

Figure 3.11. Home view of the Desktop application. The *Import Model* menu is marked in red, the *Change HeatMap Color* is marked in green, the *Server/Network Configuration* menu is marked in blue and, finally, the *Sensor Configuration* menu is marked in orange.

3.6.2 Import Model

The Import model feature allows the user to upload any .obj model through the file system. It is possible to see a zoom of this section in figure 3.12.

The development of this feature is achieved using two Unity assets:

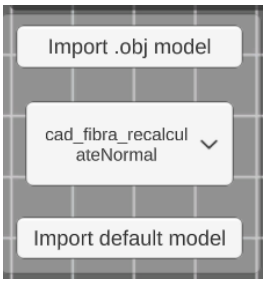

Figure 3.12. Import Model section.

- *Simple file browser*, developed by Graces Games, which allows browsing the file system in order to choose the right model;
- *Runtime obj importer*, developed by Dummiesman, allows importing the .obj model into a GameObject in Unity.

The functionalities for navigating the file system and import .obj files are implemented in the GuiManager class; its definition can be seen in figure 3.13. In detail:

- *public string [] fileExtentions = new string [] {"obj"}*: indicates the extensions allowed to show while navigating the file system through the file browsing window;
- *public void CallerFileBrowser ()*: instantiates the *FileBrowser* prefab, which allows the navigation of the file system;
- *private void LoadFileUsingPath (string path)*: loads the .obj file chosen in the *path* using *OBJLoader* prefab;
- *private void CloseBrowser ()*: closes the file browsing window, deactivating its panel;
- *public void ResetImportMenu ()*: resets the import menu window;
- *public void LoadDefaultPrefab (TMP_Dropdown value)*: loads the default prefabs, which is already imported into the application.

```
1
2  public class GuiManager : MonoBehaviour
3  {
4      ...
5      public string [] fileExtentions = new string [] {"obj"};
6
7      public void CallerFileBrowser ();
8      private void LoadFileUsingPath ( string path );
9      private void CloseBrowser ();
10     public void ResetImportMenu ();
11     public void LoadDefaultPrefab ( TMP_Dropdown value );
12     ...
13 }
```

Figure 3.13. GuiManager class definition.

Before confirming the import, the window provides various functions to change visual aspects of the model. This is provided for the user because it could be possible that the size or the rotation of the model cannot fit the window. To achieve that, the class *MonitoredObject* has been implemented with the following methods:

- *public void OpenNewMonitoredObject (GameObject sourceObject)*: imports a new MonitoredObject model;
- *public void UpdateRotation (TMP_InputField field)*: updates the rotation of the object using the window field as input;
- *public void UpdateScale (Slider slider)*: updates the local scale of the object using the window slider element as input;
- *private void CombineMeshes (List <CombineInstance> list, GameObject objectC)*: combines the meshes with the imported object;
- *private float ? NormalizeMesh (GameObject objToNorm)*: normalizes the mesh of the new imported object;
- *private void ReduceSizeMesh (GameObject objToNorm , float size)*: reduces the size of the meshes on the normalized object;
- *private Vector3 ? CenterMesh (GameObject obToCenter)*: centers the meshes on the normalized object;
- *public void ConfermModel ()*: confirms the import of the model by pressing the **Confirm Import** button;
- *public void CancelImport ()*: aborts the import procedure.

Those parameters and buttons are available in the *Import window*, shown in figure 3.6.2.

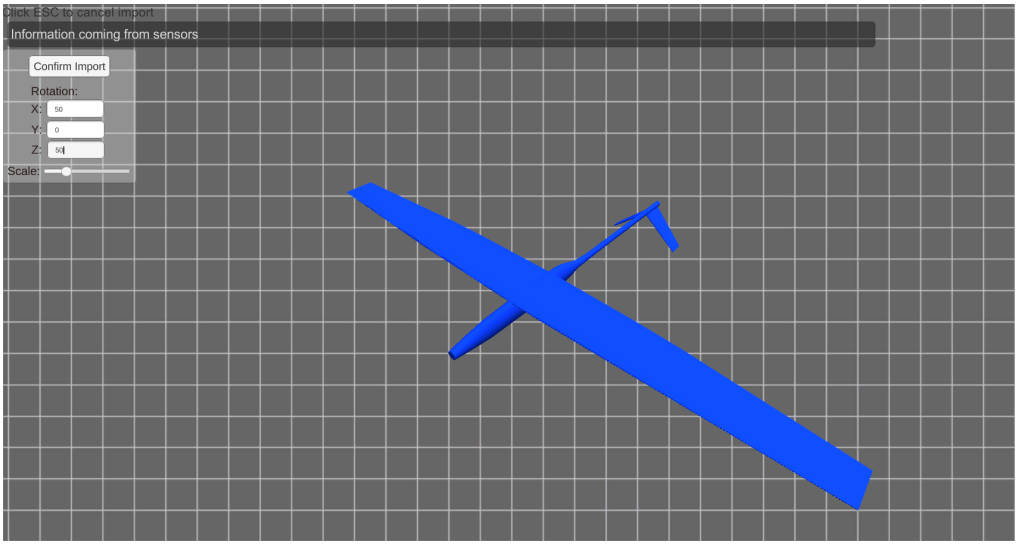

Figure 3.14. Import window.

3.6.3 HeatMap Customization

For effective monitoring of the health of the airplane, it is important to select, for the heatmap, an effective color palette. There are 6 ready-to-use presets, but the user can personalize his own using the *Change HeatMap Color* menu, which can be opened by pressing the plus button (figure 3.15). In this window, a slider and a box allow the user to select a color hue and a saturation for a single gradient. Once that a gradient is selected, it will be displayed in one of the 9 boxes on the left. Pressing the Confirm button saves the newly created heatmap on the top default presets, otherwise, by pressing the Cancel button, the newly created heatmap will be discarded.

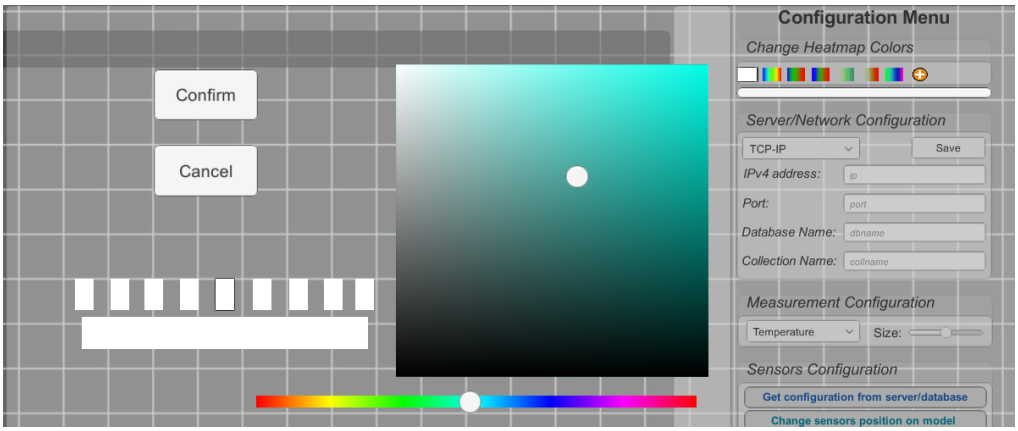

Figure 3.15. *Change HeatMap Color* menu.

3.6.4 Sensor Configuration

The Sensor Configuration menu allows the user to import the sensor configuration from the source of the data, modifying their position into the model and updating the information about the idle and the maximum variance of the wavelength (figure 3.16).

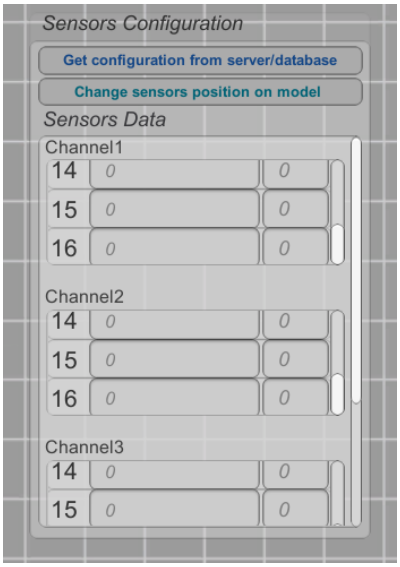

Figure 3.16. *Sensor Configuration* menu.

By pressing the *Get configuration from server/database* button, the application will retrieve the stored configuration from the Middleware; this operation, in case of TCP-IP mode, will be performed by sending a REQUEST_CONFIG message, while, in the case of Real-Time or Non-Real-Time mode, the application will retrieve the configuration data by querying the database.

The changing sensors position feature is activated by pressing the *Change sensors position on model* button. It will deactivate the menu window and the status **ChangePos** will be activated. In the **Change Position** window, which can be seen in figure 3.17, the user can interact with the sensors, indicated by the label with channel and grating, and relocate them in different parts of the model. The impact of the sensors on the model is displayed using a Raycast object, which casts a ray from the camera position in direction of a pointed coordinate, stopping when it hits the MonitoredObject GameObject **Collider**.

The Sensor Data configuration can be found at the bottom of the menu and it can be used

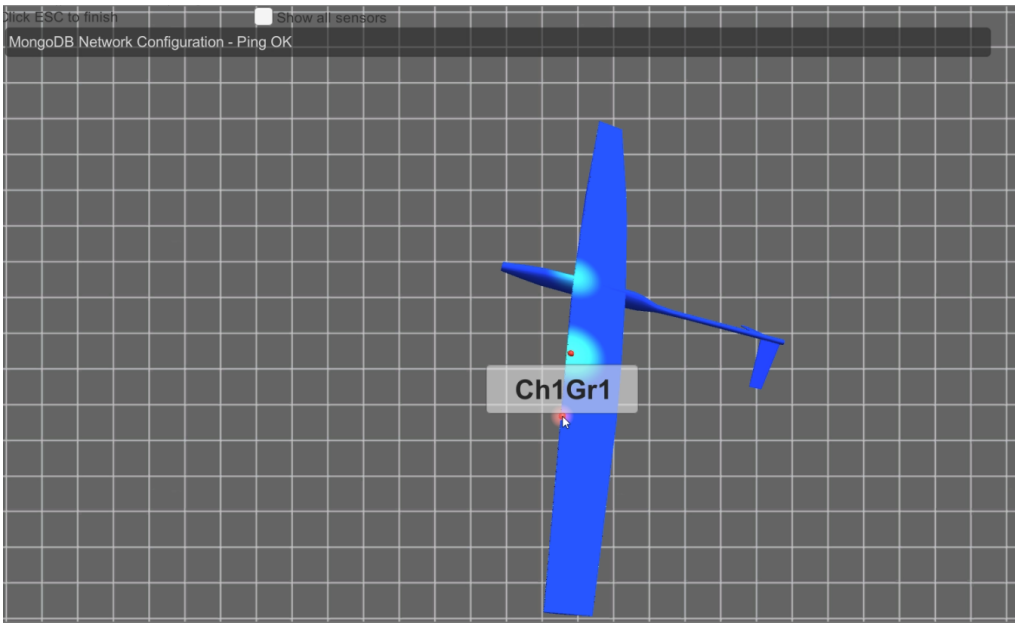

Figure 3.17. Unity *Change position* view.

to update the information regarding the idle and the max variance wavelength of each sensor. This menu shows all the possible sensors that the interrogator can provide. In the case study the SmartScan, which supports a maximum of 64 sensors distributed by 16 gratings in 4 channels, is used.

3.6.5 Simulation view

The simulation view can be enabled when the model and sensors are setup by pressing the **Start Monitoring** button. It can be enabled using 3 modes: TCP-IP, Real-Time, and Non-Real-Time. This window displays the model of the monitored object, comprehending the sensors placed on it, and a real-time graph, which is updated with the wavelength of the sensors (figure 3.18).

Every time a new wavelength data arrives, the GameManager class will update the MonitoredObject with the new properties and adding the newly read wavelength into the graph. The graph is an implementation of the *Graph and Chart* plug-in, which allows adding different lines, called Categories, and updating their information during the simulation. The methods implemented for managing the graph are:

- *public void InsertCategory ()*: inserts a new category on the Line graph;
- *public void UpdateCategory (List <KeyValuePair <uint64, float >> wav)*: updates the line graph, a point for each active sensor considering the max number of points that the user can update in the window;
- *public void ShowGraph ()*: sets the line graphs as visible;
- *public void HideGraph ()*: hides the line graphs in the window.

The heatmap on the model is updated each time that new data is received. The data contains the ray cast shader of the closest sensor and it is updated considering the displacement of the measure. Moreover, the GameManager class saves the latency of the received data considering the timestamp of the registration of the data coming from the Middleware and the timestamp of the actual display of the sensor data on the application.

The user can terminate the simulation by pressing the Escape button. The application will update the window with all the menus of the home view and will create, in the background, a post-process graph containing all the wavelength data of the simulation (figure 3.11).

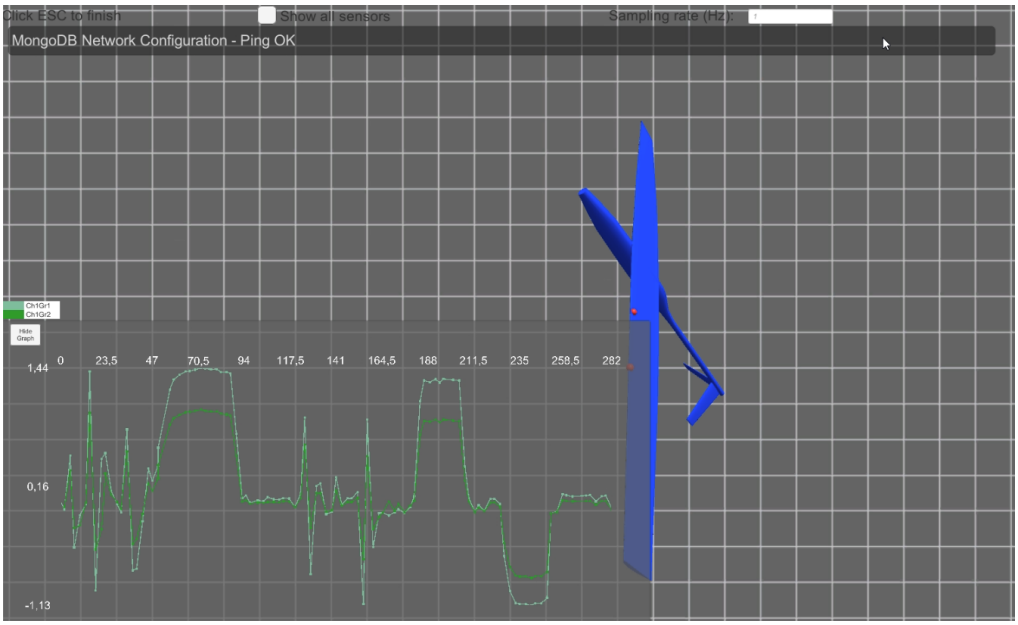

Figure 3.18. Simulation view.

3.6.6 TCP-IP mode

The TCP-IP mode has been implemented in the previous iteration of the project [2]. It can be enabled in the *Server/Network Configuration* menu by selecting the TCP-IP option in the drop-down element (figure 3.19).

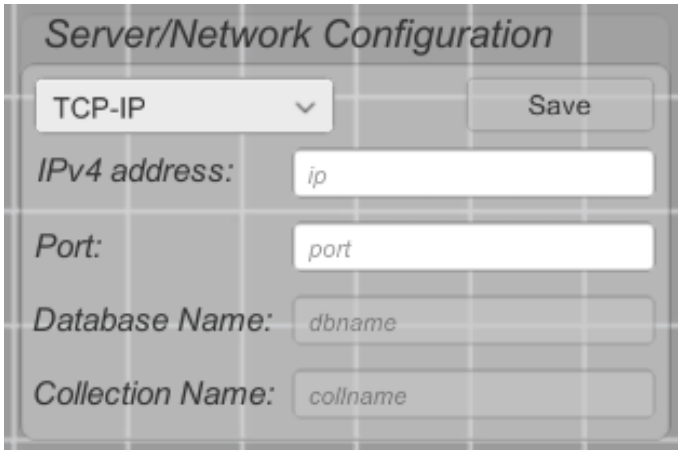

Figure 3.19. Server/Network Configuration menu with TCP-IP.

By providing the IPV4 of the Middleware and its port and by pressing the Save button, the application is able to establish a TCP connection with the Middleware itself. If the connection is successful, the configuration of the sensors can be retrieved by pressing the *Get configuration from server/database* button, which can be found on the Sensor Configuration menu. Through the TCP protocol, the application will send a REQUEST_CONFIG message to the Middleware, which will replay with the Configuration Data (check subsection 3.4.3 for further details).

Once the simulation has been setup, the user can start the monitoring by clicking the **Start Monitoring** button. This will open the Simulation View and send a REQUESTDATA_START message to the Middleware, enabling the client communication on its side. Then, always using the TCP protocol, the application will start receiving the peakData from the Middleware and

showing them on screen in the Simulation View.

At the end of the Simulation, the application will send a REQUESTDATA_END message to the Middleware, releasing the respective thread in charge of the client connection. This flow is managed by the TCPManager class, whose definition can be seen in figure 3.20.

```
1
2  public class TCPManager : MonoBehaviour
3  {
4      ...
5      // SERVER_METHODS
6
7      //It start the TCPListener based on the information set by the user
8      public bool InitTcpListener(){..}
9      //TCP Server for the application, it run on a different thread
10     public void ListenerData (Int32 port){...}
11     //TCP Server for the network configuration
12     public void ListenerConfig (Int32 port){..}
13     // CLIENT_METHODS
14     //Send message to the middleware client
15     public void Send ( TypeMessage type){...}
16     //Serve client connected to the server
17     public void ServeClientData ( TcpClient result ){...}
18     //Serve configuration packet
19     void ServeConfigPacket ( NetworkStream stream ){...}
20     ...
21 }
```

Figure 3.20. TCPManager class definition.

3.6.7 Real-Time mode

The Real-Time mode is the feature that uses the ChangeStream feature of MongoDB. This option, as it is depicted in figure 3.21, can be enabled by clicking the drop-down element of *Server/Network Configuration* menu. Then, the user can configure the IPV4 address, port, and database name of the MongoDB instance.

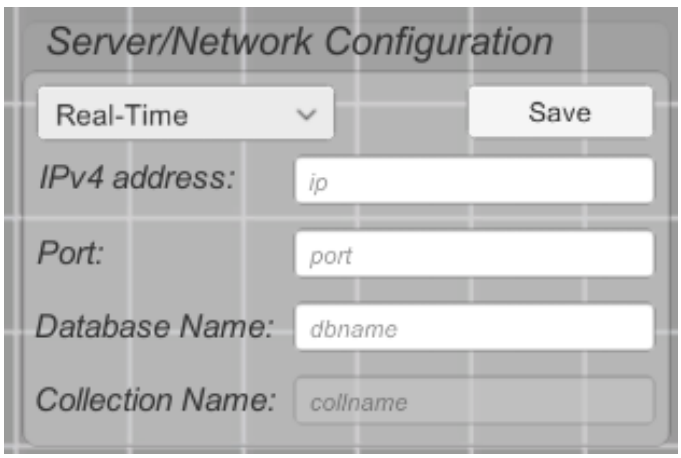

Figure 3.21. Server/Network Configuration menu with Real-Time.

```
1  public class ChangeStreamManager : MonoBehaviour
2  {
3      public string IPAddress { get ; set ; }
4      public int Port { get; set; }
5      public string DbName { get; set; }
6
7          public bool SetupChangeStream() {...}
8
9          public void StartWatch() {...}
10
11         public void GetConfiguration(){...}
12
13 }
```

Figure 3.22. ChangeStreamManager class definition.

This flow is managed by ChangeStreamManager class, whose definition is depicted in figure 3.22. When the user presses the **Save** button, the application will try to connect with the MongoDB database. The GuiManager class will call the `ChangeStreamManager.SetupChangeStream()` function, which tries to reach the database and to update the object information. If the connection is successful, the function will return **true**.

The user can get the configuration of the sensors by pressing the *Get configuration from server/database* button, which can be found on the Sensor Configuration menu. This action will call the `GetConfiguration()` function, which will query the sensors configuration from the database. The last collection will be the one which will be read because, in the optic of one single-use at-a-time, it will be surely the one where the Middleware is storing the real-time data.

By pressing the **Start Monitoring** button, the simulation view will be enabled and the menu will be disabled. Moreover, it will start the reading from the ChangeStream feed. The setup of the feature can be resumed with 2 lines of code:

```
var options = new ChangeStreamOptions { FullDocument =
    ChangeStreamFullDocumentOption.UpdateLookup };
var pipeline = new
    EmptyPipelineDefinition<ChangeStreamDocument<BsonDocument>>().Match("{
    operationType: { $in: ['insert'] } }");-}
```

This will create a new ChangeStream pipeline from the collection. The pipeline options will specify the watch of the full document at each insert operation. In order to enable the pipeline in the collection, the function **Watch(...)** will create a new cursor for the results, which will be looped until a new insert event will happen in the collection. The driver will store the ChangeStream feed into the cursor, which can be iterated because there could be more than one result. The aforementioned process is expressed by the C# code illustrated in figure 3.23. The insert pipeline watching operation is interrupted by a TermianteThread event, which is activated by the user by pressing the Escape button.

3.6.8 Non-Real-Time mode

The Non-Real-Time mode is an offline visualization of the sensors data of a previous simulation test. As can be seen in figure 3.24, it can be selected in the *Server/Network Configuration* dropdown menu. To retrieve the data of the simulation, it is necessary to indicate IPv4 address, port, database name, and collection name by typing those data in their dedicated section. Pressing the Save button, the application will perform a check on the connection to the Server by calling the **InitMongoDB()** function, which, as can be seen in figure 3.25, is defined in the MongoDBManager class. If the connection is successful, the user can retrieve the configuration of the sensors

```
1  while (!GameManager.instance.TermianteThread){
2      using (var cursor = collection.Watch(pipeline,options)){
3          while (cursor.MoveNext() && cursor.Current.Count() == 0 &&
               !GameManager.instance.TermianteThread)
4          {
5              //Waiting for data
6          } // keep calling MoveNext until we've read the first batch
7
8          var result = cursor.Current;
9          foreach (var elem in result){
10             //reading line in the cursor
11             ....
12         }
13     ...
14     }
15     ...
16 }
```

Figure 3.23. Detail on Change Stream code setup.

by clicking the *Get configuration from server-database* button, which will query the selected collection using the **RequestSensorsConfiguration** function of the MongoDBManager class and will update the application sensor data.

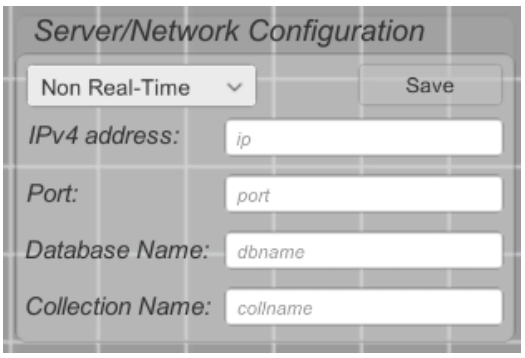

Figure 3.24. Server/Network Configuration menu in Non-Real-Time mode.

At the start of the simulation, the **UpdateSensorInformation()** function, which asynchronously reads peakData from the database, is called by the GameManager class. Asynchronous programming is made necessary because of the poor performance of the selection query with a huge amount of data. Thanks to the MongoDB indexes, the data are consistent with the natural order of the simulation. At the creation of the collection, it is possible to specify the indexes: in this case, it has been specified a compound index with the field "type" and "timestamp" of the Unity Peak Data (defined in the subsection 3.4.3). In the graph of the simulation view, the user can select the window of the data to be displayed: this option facilitates the readability of the simulation data. By pressing the Escape key, the user will be redirected to the main menu.

3.6.9 End of the simulation

At the end of each simulation, the application will build a log file and a line graph as a recap of the simulation. The log file, an example of which can be seen in figure 3.26, is a .csv where it is

```
1  public class MongoDBManager : MonoBehaviour
2  {
3      IMongoClient client = null;
4      IMongoDatabase db;
5      IMongoCollection<BsonDocument> collection;
6
7      public MongoInformation mongoInfo { get; set; }
8
9      public bool InitMongoDB() {...}
10
11     public void RequestSensorsConfiguration () {...}
12
13     public void UpdateSensorInformation (){...}
14 }
```

Figure 3.25. MongoDBManager class definition.

possible to find all the wavelengths of the simulation at each timestamp. The file is filled in the same queue as the simulation by the **LogManager** class.

```
1  Timestamp       ,Ch1Gr1   ,Ch1Gr2   ,Ch2Gr1
2  1613579105122918,1546.7190,1530.6010,1540.02
3  1613579106221705,1546.8470,1530.7100,1540.02
4  1613579107320490,1546.8490,1530.7100,1540.02
5  1613579108308992,1546.8500,1530.7100,1540.02
6  1613579109415673,1546.8520,1530.7100,1540.02
7  1613579110513960,1547.7320,1531.4490,1540.02
8  1613579111502955,1547.0360,1530.8280,1540.02
9  1613579112601740,1546.7640,1530.6180,1540.02
10 1613579113700121,1546.8750,1530.7080,1540.02
11 1613579114689108,1546.9040,1530.7180,1540.02
12 1613579115787395,1546.8890,1530.7070,1540.02
13 1613579116886164,1547.6050,1531.2710,1540.02
14 1613579117882964,1546.2450,1530.3170,1540.02
15 1613579118981345,1546.9080,1530.7100,1540.02
16 1613579119970254,1546.7320,1530.5870,1540.02
17 1613579121068634,1546.6300,1530.5440,1540.02
18 1613579122167418,1546.8800,1530.6920,1540.02
19 1613579123266286,1547.0170,1530.7830,1540.02
20 1613579124254782,1546.7660,1530.6110,1540.02
```

Figure 3.26. Example of a Log file with three active sensors.

After that all wavelength data are written in the .csv file, the file is closed and a Python script asynchronously runs using the *Process* class, creating a line graph from it: it is possible to see an example in figure 3.27. It is based on the Numpy and Pandas libraries and generates a line graph as .png image by reading each line of the previously created .csv file.

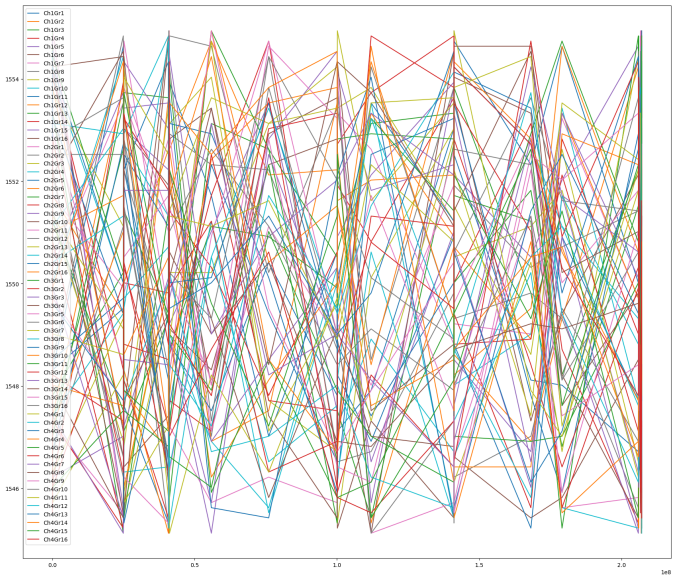

Figure 3.27. Example of a line graph image with 64 active sensors.

Chapter 4

User and Developer Guide

In this chapter, it will be explained how to install and operate with the software of the thesis. The focus will be the network configuration, and on how to setup the various components.

4.1 Network Configuration

Depending on the flow, the components of the projects need to be configured accordingly. The interrogator and the Middleware are connected via Ethernet. In case that a Smartscan act as interrogator and a Raspberry Pi 4 Model B acts as Middleware, the user needs to setup on the Middleware a static address for the Ethernet port (eth0) with IPv4 address 10.0.0.2 and gateway 10.0.0.150, as in figure 4.1.

```
ethernets:
    eth0:
        addresses: [10.0.0.2/24]
        gateway4: 10.0.0.150
        match:
            driver: bcmgenet smsc95xx lan78xx
        optional: true
        set-name: eth0
version: 2
```

Figure 4.1. Middleware configuration for Ethernet.

On the other side, the middleware needs to be configured to connect to the Desktop/AR application, in case of TCP-IP mode (4.2), or to connect to the Server where MongoDB is hosted in case of Real-Time mode (4.3). This configuration is embedded in the code of the Middleware, and can be found in the file *client.hpp*.

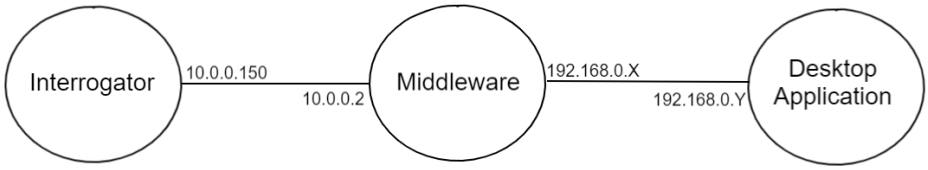

Figure 4.2. Network configuration for TCP-IP mode.

The variables to setup are:

- HOLOCLIENT_SRC_ADD : IPv4 address of the Middleware for the wireless LAN (wlan) interface;
- HOLOCLIENT_DST_ADD : IPv4 address of the Desktop/AR Application, needs to be setup in case of TCP-IP mode;
- MONGO_URI : IPv4 address of the Server in MongoDB connection string format like "**mongodb://192.168.0.103:27017**".

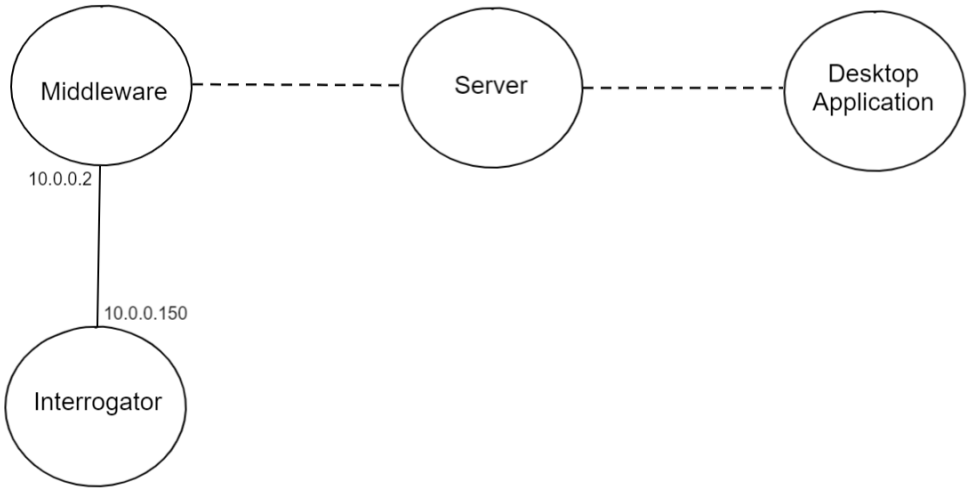

Figure 4.3. Network configuration for Real-Time mode.

4.2 Middleware setup

The source code of all the components of the thesis can be found on git, hosted in Bitbucket by Atlassian. The middleware source code is under the repository *photonext-middleware* and in order to be compiled and run, it needs various dependencies. There are in-house dependency derived by the interrogator *libutils* and *libsmartscan*, and MongoDB dependencies as *mongocxx* and *libmongoc*. Both dependencies are not available through Linux or Debian Package Manager.

The MongoDB driver version is the 3.6.2 for *mongocxx*, which requires a *libmongoc* version of 1.17.0 or later. In order to install *libmongoc* and *mongocxx* it is needed to follow those passages:

1. clone the repository or Download the release tarball from the MongoDB repository on GitHub;
2. build and install the source code into the machine.

Using the default configuration, the process will install the libraries in */usr/local/lib*. Figure 4.4 shows the Linux commands to install the dependencies needed.

4.3 Server setup

The Server can host any OS in which can be deployed a MongoDB instance having a version equal or superior to the 3.6, in where the Change Stream was released. In the thesis, it has been chosen to use *Ubuntu Server 20.04 64 bit* as OS with the MongoDB version 4.4.3. Similar to the driver, MongoDB is not updated in the Package Manager of Ubuntu or Debian.

```
wget https://github.com/mongodb/mongo-c-driver/ \
    releases/download/1.17.4/mongo-c-driver-1.17.3.tar.gz
tar xzf mongo-c-driver-1.17.3.tar.gz
cd mongo-c-driver-1.17.3
mkdir cmake-build
cd cmake-build
cmake -DENABLE_AUTOMATIC_INIT_AND_CLEANUP=OFF ..
sudo cmake --build . --target install

...

curl -OL https://github.com/mongodb/mongo-cxx-driver/releases/ \
   download/r3.6.2/mongo-cxx-driver-r3.6.2.tar.gz
tar -xzf mongo-cxx-driver-r3.6.2.tar.gz
cd mongo-cxx-driver-r3.6.2/build
sudo cmake --build . --target EP_mnmlstc_core
cmake --build .
sudo cmake --build . --target install
```

Figure 4.4. Linux commands for install MongoDB C Driver (libmongoc), BSON library (libbson) and C++ Driver (mongocxx) [19] [20].

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc \
    | sudo apt-key add - \
echo "deb [ arch=amd64,arm64 ] \
    https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" \
    | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

sudo apt-get update
sudo apt-get install -y mongodb-org
sudo systemctl enable mongod
sudo systemctl start mongod
```

Figure 4.5. Bash commands for install MongoDB on Ubuntu Server [21].

To install MongoDB, follow the steps detailed with code in figure 4.5:

1. install the MongoDB 4.3 GPG key;
2. add the source location for the MongoDB packages;
3. download the package details for the MongoDB package;
4. install MongoDB;
5. enable MongoDB service to load at startup.

In order to deploy a replica set which will enable Change Stream, it is needed to modify the configuration file of the database */etc/mongod.conf* through the addition of the following lines:

```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0
#replication:
replication:
   replSetName: "rs0"
```

After the changes, the Server needs to be restarted to enable all the functionalities.

4.4 Desktop application - external dependencies

To end the simulation activities in the Desktop/AR application and to build the line-graph, it is required to install a Python interpreter. For the thesis, it has been used Python 3.9.1 version with *NumPy* and *Pandas* as additional libraries.

Chapter 5

Tests and Results

In this chapter, it will be shown the methodologies and tests conducted during the thesis. Due to the COVID pandemic, it was not possible to perform tests with a fully built aircraft. Some tests, representing real case problems, were performed in DIMEAS laboratories recreating almost entirely the real environment.

5.1 Test scenarios

Tests have been setup with a focus on the responsiveness of the software. There are 2 major scenarios for analysis: Synthetic test and Laboratory test. In the case of synthetic test, the system has been setup with a software emulator of the interrogator running on a Ubuntu virtual machine. The Middleware and the Server were hosted on a Raspberry Pi 4 Model B, that is connected via Ethernet with the interrogator and via Wi-Fi 5GHz with the viewer. Wi-Fi was chosen over 4G/5G because of the limitation imposed during the COVID pandemic. The viewer was installed on a Windows laptop, equipped with an Intel(R) Core(TM) i7-9750HF CPU @2.60GHz as CPU, 16 GB of DDR4 RAM, and an NVIDIA® GeForce® RTX 2060 as GPU. The laptop is powerful enough to not create a bottleneck on a full system. In the laboratory test, SmartScan© from SmartFibres© was used as interrogator for the FBG. The equipment used in the laboratory has been documented with two photos, depicted in figure 5.1 and figure 5.2.

Figure 5.1. Test equipment used in the laboratory. On the left there is the SmartScan© from SmartFibres©, while on the right the Raspberry Pi 4 Model B can be found.

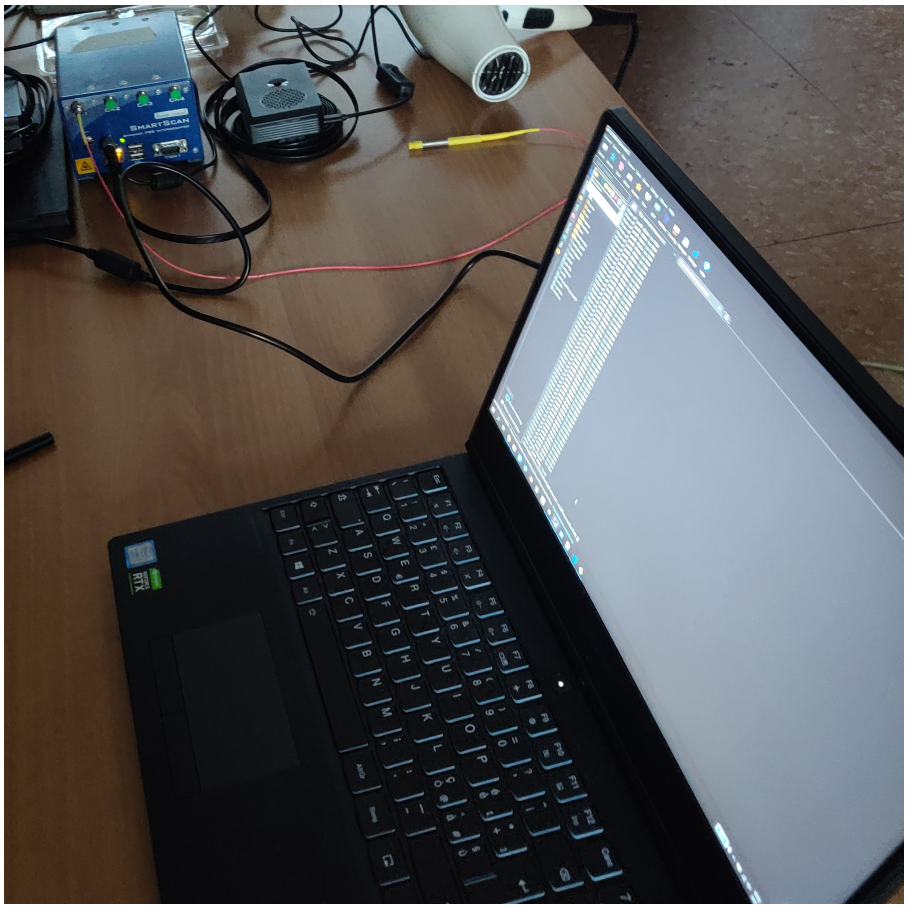

Figure 5.2. Test equipment used in the laboratory.

The delay of the system is calculated as the overall delay from the middleware to receive the peak data to the visualization in the Desktop/AR application. During the flow, the application collects timestamps at each phase of the simulation, in particular in the Interrogator, the Middleware, and the Viewer, as expressed in figure 5.3.

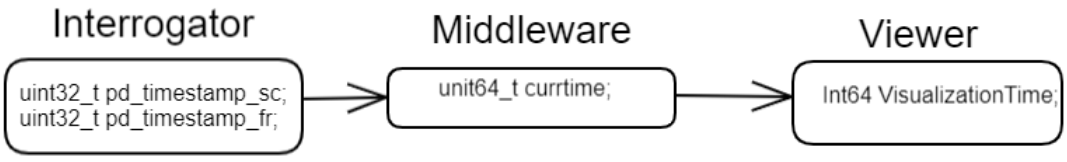

Figure 5.3. Timestamps stored during the simulation.

5.2 Synthetic test

The synthetic test was performed using the SmartScan interrogator emulator. It is a C-based application for Linux that generates random UDP traffic, acting similarly to the way that the SmartScan interrogator acts. The SmartScan interrogator can send 4 types of messages:

- diagnostic messages: they are used to set the operational state of the interrogator and to check its current status;

- maintenance messages: using these messages it is possible to set the configuration of the interrogator, like the transmission speed, the scan speed, etc;
- continuous data: this is the peak data stream detected by the interrogator;
- scan data: raw data collected by the interrogator.

The messages are built configuring the payload of the UDP message expressed in figure 5.4. It is possible to configure the number of channels and gratings and their transmission rate. Thanks to that it is possible to test the system with the interrogator loaded at full capacity.

```
1  typedef struct S_DATA_DEF
2  {
3          uint_16 usFrameSize; // Total Nr of bytes - 2
4          uint_8 ucHdrSizex4; // Nr of bytes in header/4 (9)
5          uint_8 ucFrameFormat; // eg 0x84, 8gratings, 4channels
6          uint_32 ulFrameCount; // Incremented number
7          uint_32 ulTimeStampH; // UTC seconds H (scan time)
8          uint_32 ulTimeStampL; // UTC second L (scan time)
9          uint_32 ulTimeCodeH; // UTC seconds (tx time)
10         uint_16 usTimeInterval; // sample interval in uS, ie 400
11         uint_16 usSpare; //
12         uint_16 usMinChannel; // eg 1
13         uint_16 usMaxChannel; // eg 400
14         uint_32 ulMinWaveFreq; // freq @ min channel
15         uint_32 ulSpare; //
16 } S_DATA, *PS_DATA;
```

Figure 5.4. Payload data definition of the interrogator.

Used Instruments

To recreate the real scenario as best as possible, the emulator was installed in different hardware than the Middleware. In particular, it runs on a Ubuntu virtual machine hosted on the same laptop as the Viewer. The laptop was configured to interact via Ethernet with the middleware and to receive the processed sensors data from the Server via Wi-Fi.

Results

The emulator generated random signals in all 4 channels and 16 gratings. Thanks to that, it is possible to stress the Middleware and the Viewer in order to retrieve the overall delay of the project. From the processed data there is not a lot to read, as the signal does not represent a real case scenario but some randomly generated wavelengths (figure 5.5).

What is interesting about the scope of the test is the overall delay from the registration of the data to their actual visualization in the Desktop/AR application. The mean delay is 1100 ms ca. with a min value of 908ms and some spikes of 10903 ms. Considering this result, the framework is not suitable for real-time visualization of the data but, as the tests in the laboratory show, the delay decreases drastically in a real case scenario. This increase in delay is due to 2 main factors:

Lack of resources for emulator and viewer: The emulator was hosted in a Virtual Machine on the laptop where the desktop application is hosted. This implies that there is another abstraction layer and the machine has to share resources with two computation heavy applications;

Number of sensors: This test was operated by mocking all 64 FBGs sensors, that increase the bandwidth consumed by the application.

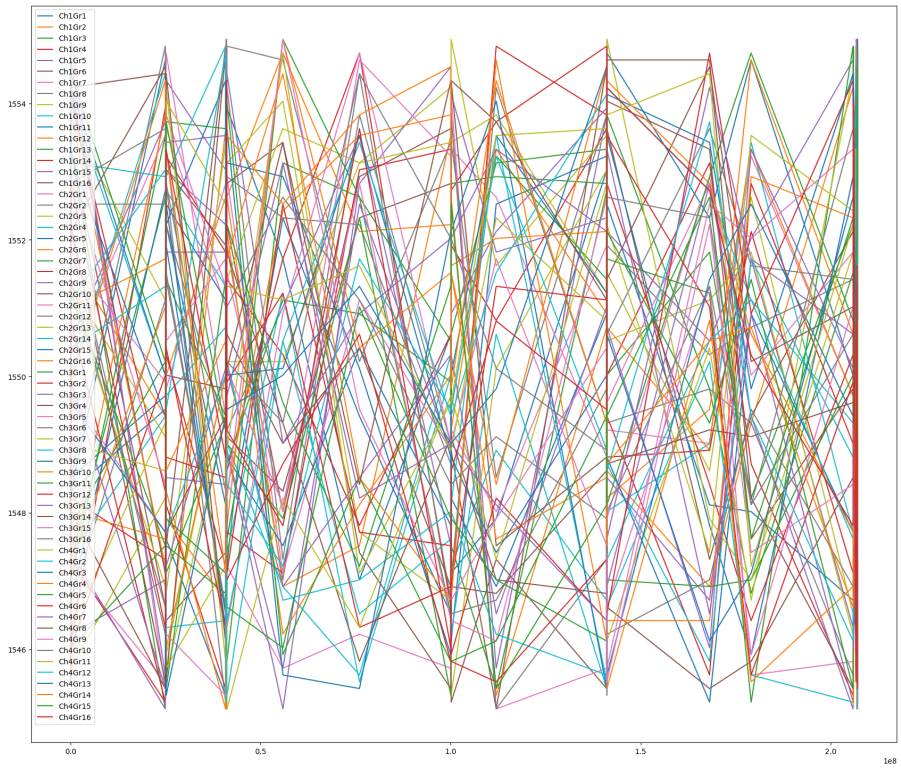

Figure 5.5. Post processed graph of peak data from the synthetic test.

5.3 Laboratory tests

Tests were performed using laboratory equipment provided by DIMEAS. The idea was to use actual FBGs sensors and apply the same strains and temperatures that the aircraft could be subjected to. There were performed 3 types of tests:

- Thermometric Probe: 1 FBG grating in 1 channel;
- Tension Bar: 2 FBGs gratings in 1 channel;
- Thermometric Probe + Tension Bar: 2 channels with 1 and 2 FBGs gratings respectively.

These sensors are similar to the ones that will be used in the aircraft. All the tests were performed in case of idle and in case of stress, that, in the case of the Thermometric probe, implies heating the probe with a hairdryer and, in the case of the Tension Bar, it implies applying a strain in different zones of the bar.

The FBGs sensors are connected with the SmartScan© from the SmartFibres© interrogator and, at the same time, it is connected with the Middleware, hosted on a Raspberry Pi 4 Model B, via Ethernet. The Server with the MongoDB instance is hosted also in the same hardware. The visualization framework runs on a Windows Laptop connected with the Server via a WiFi 5Ghz hotspot.

5.3.1 Thermometric Probe

This test is designed to analyze how the system behaves with thermometric data. The λB wavelength calculated of an FBG sensor, subjected to physical or thermal disturbance, can be expressed as:

$$\frac{\Delta\lambda B}{\lambda B} = (1 - p_e)\varepsilon + (\alpha + \xi)\Delta T[17]$$

in which $p_e, \varepsilon, \alpha, \xi$, and T are the effective photo-elastic constant, axial strain, thermal expansion coefficient, thermal optic coefficient, and temperature shift, respectively [18].

Used Instruments

DIMEAS engineers crafted the Thermometric Probe used in the experiment (figure 5.6) relying on the building principle depicted in figure 5.7.

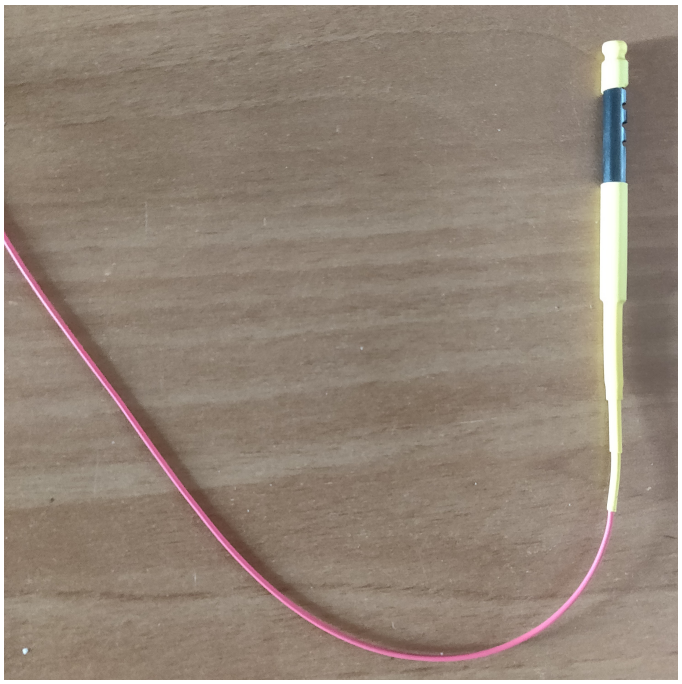

Figure 5.6. Thermometric Probe used on the test.

This sensor is connected to the interrogator using channel 1. To heat the sensor, a hairdryer that shoots hot hair directly into the sensor was used.

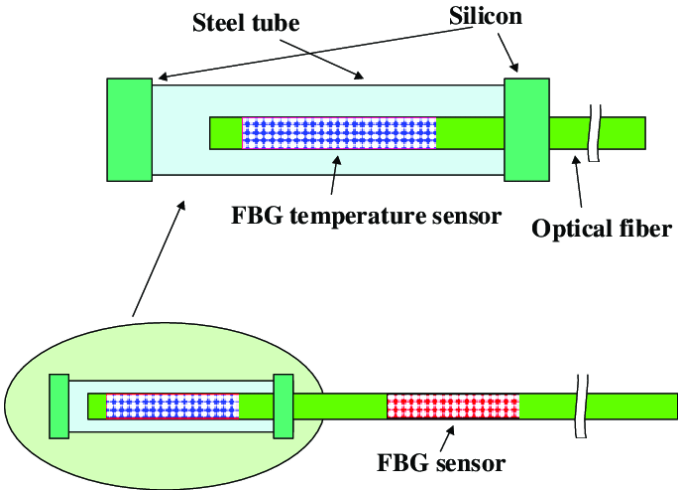

Figure 5.7. Schematic diagram of FBG temperature and strain sensors [18].

Results

From the obtained real-time graph (figure 5.8), it is possible to distinguish the different phases of the experiment. In detail, the heat-up process is increasing the value of wavelength following the formula expressed before, as soon it reaches the material breakpoint. At that point, the measurement becomes noisy. After that, the sensor was left to cool down naturally and, as it is possible to see on the graph, the value of the wavelength decreases following a branch of the hyperbola style until it reaches its idle value.

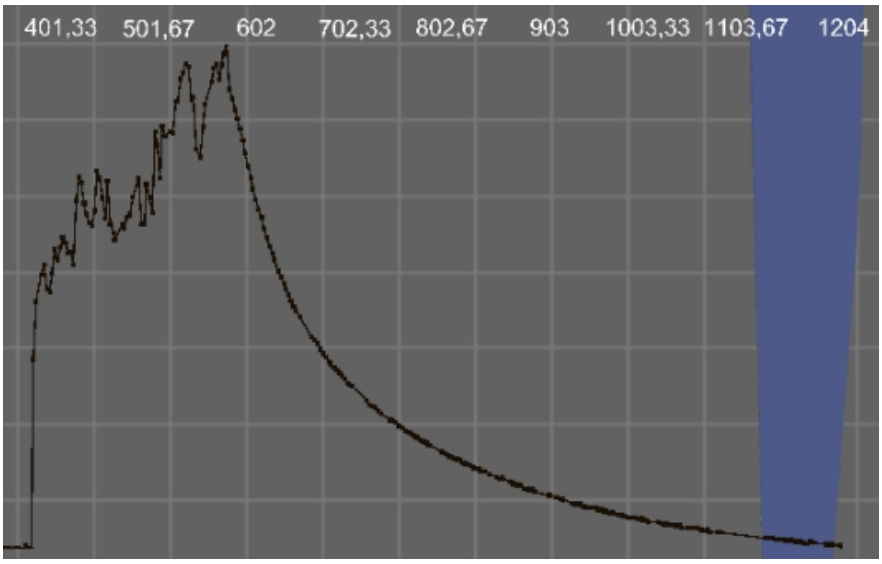

Figure 5.8. Thermometric Experiment data collected during the experiment.

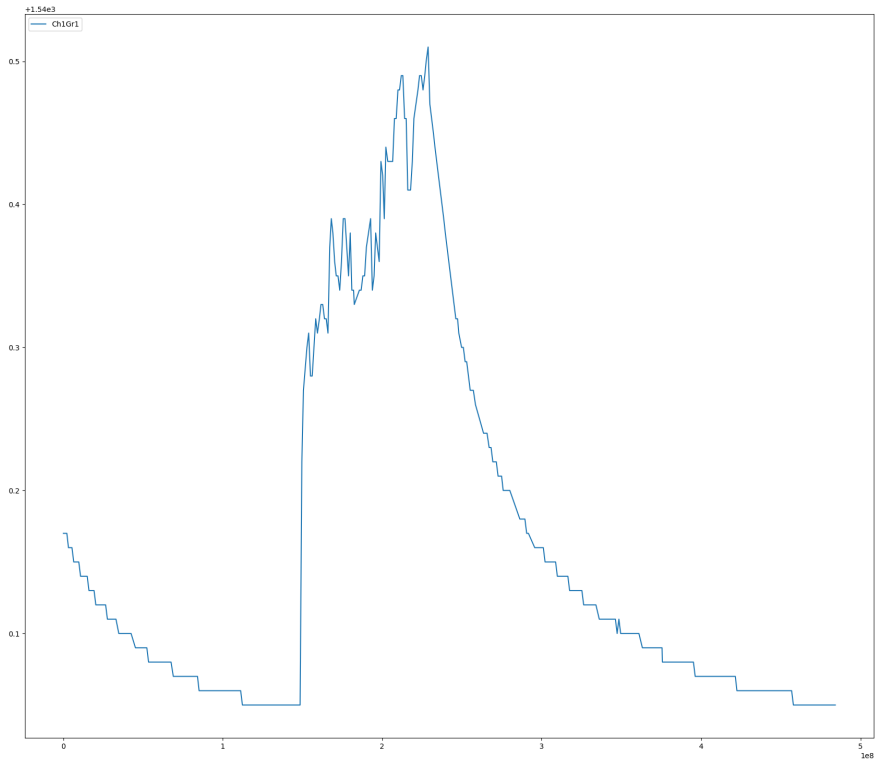

Figure 5.9. Post-processed graph of the Thermometric Experiment.

The same behavior can be seen on the post-processed graph (figure 5.9). It is possible to extract the values of the wavelength from the Viewer logs or the post-processed graph. For this experiment, the baseline of the wavelength is 1540.00 nanometers and the maximum that the sensor can reach is 1540.50 nanometers.

This experiment, with the minimum number of sensors, shows the constraints of the solution. The delay of the registration of the data and its actual visualization in the Viewer is in the mean value around 180 ms, with a lower value delay of 39 ms and some spike of 1320 ms. In table 5.1 the delay in milliseconds of the test can be shown.

Number of sample	381
Mean value	186.7270 ms
Fastest delay	39 ms
Slowest delay	1320 ms

Table 5.1. Delay (in milliseconds) of the thermometric probe test.

5.3.2 Tension Bar

This test was conducted to analyze how the system behaves with structural monitoring. The wing of the aircraft was represented as a flexible aluminum bar that could be bent in two directions: forward and backward. This flex reflects the strain that the aircraft can be subjected to during the flight. As depicted in figure 5.10, the left wing was chosen for positioning the virtual sensors.

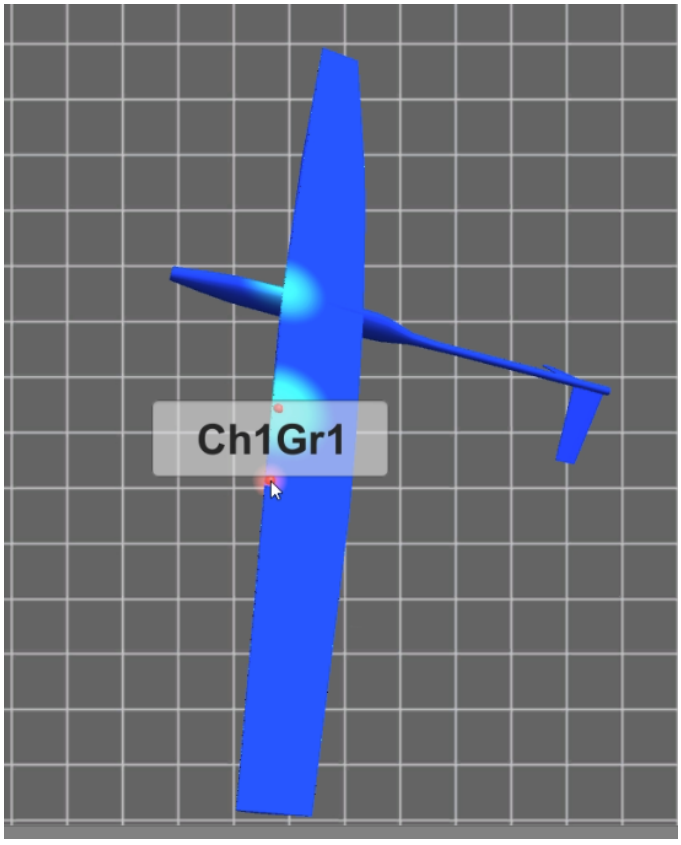

Figure 5.10. Sensors virtual position on the Viewer for the "Tension Bar" test.

Used Instruments

The aluminum bar was equipped with 2 FBGs on the same fiber cable (figure 5.11). The FBGs were connected to the SmartScan interrogator on Channel 1. The bending forward and backward was done using the table as the pivot.

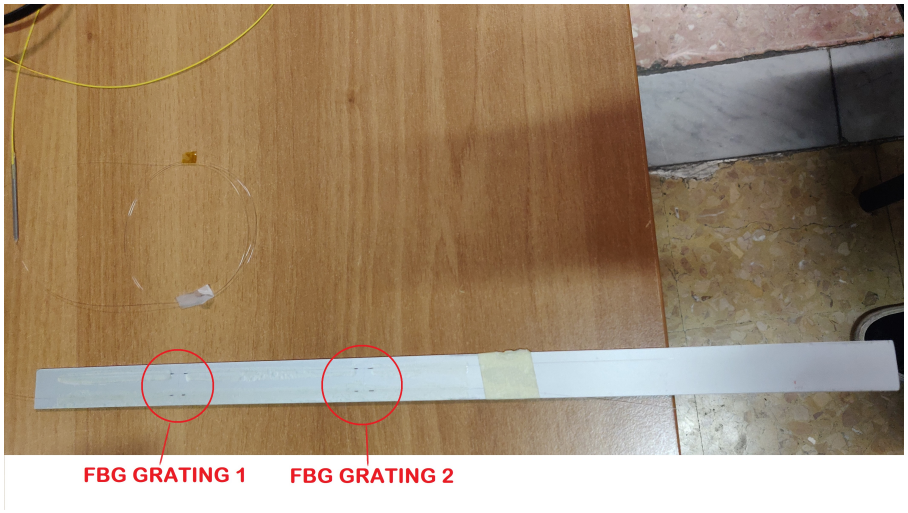

Figure 5.11. Aluminium bar with 2 FBGs sensors in series used in the experiments.

Results

Differently from the previous experiment, the wavelength could have negative values with respect to the idle value, achieved by flexing the bar backward (figure 5.12). In the post-processed graph represented in figure 5.13, it is possible to spot how the different wavelength behaves in the same channel, at different base wavelengths.

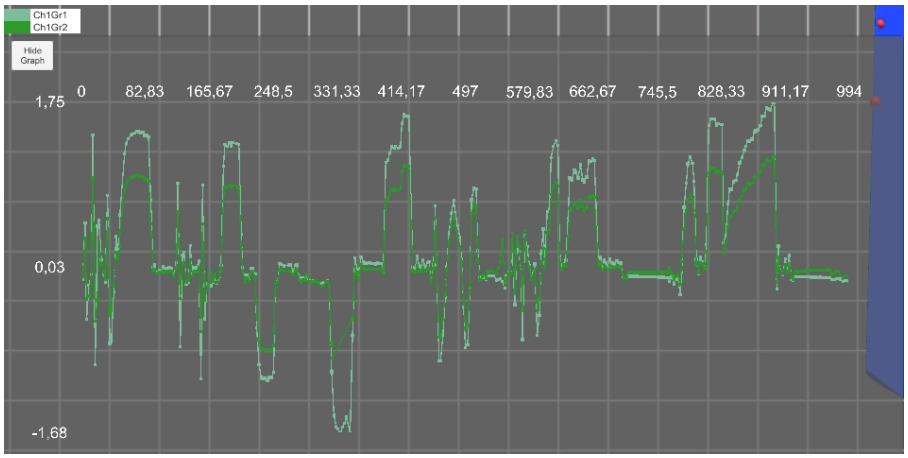

Figure 5.12. Tension Bar real-time graph during the experiment.

In detail, the most distant sensor from the pivot (Ch1Gr1) has an idle wavelength of 1547 nm, meanwhile, the sensor at the second grating (Ch1Gr2) has an idle wavelength of 1530 nm. As the two sensors are connected at the same bar, the strain applied at one is directly proportional to the other, with the difference that the closest one has less variance than the other one. In the table 5.2 are shown, in detail, the wavelength data collected during this test, with the focus on the mean, max, and min value.

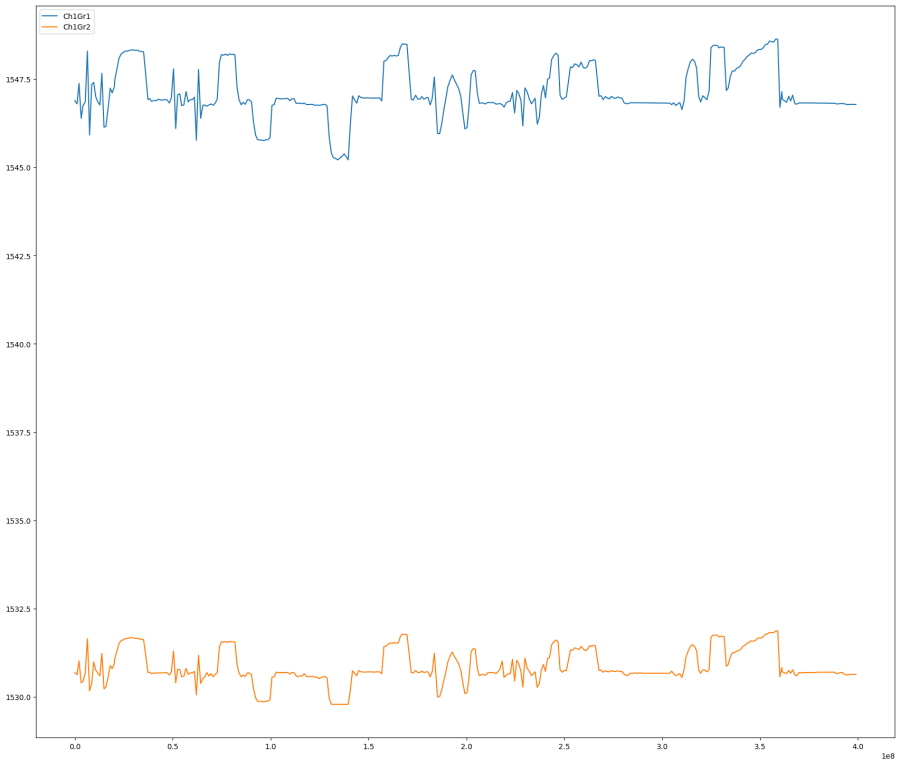

Figure 5.13. Post processed graph of the Tension Bar experiment.

	Ch1Gr1	*Ch1Gr2*
Idle value (nm)	1547	1530
Mean value (nm)	1547.1319	1530.8544
Max value (nm)	1548.634	1531.87
Min value (nm)	1545.21	1529.79

Table 5.2. Wavelength data collected during the Tension Bar experiment.

In this test, the delay between the capture of the information from the interrogator and the visualization in the Viewer presents the same variance as in the previous test. In particular, the mean delay of 133.81 ms, with some slow data coming at 1334 ms, while the fastest arrived with a delay of 21 ms.

5.3.3 Thermometric Probe + Tension Bar

This test was designed to reproduce a real case scenario in which the system can have different types of sensors and forces applied. In particular, it shows the importance of thermal variation during the strain of material. One of the possible heat sources in a real case scenario can be, for example, the engine of the aircraft.

Used Instruments

The current test is a mix of the previous two, using the previously used aluminum bar depicted in figure 5.11 on the first channel, and the thermometric probe depicted in figure 5.6 on the second channel. The 2 sensors were closed to each other, in order to reproduce how the different forces can affect the monitoring of the system.

Results

The graph derived from this test, which can be seen in figure 5.14, has three components: the 2 FBGs of the aluminum bar and the FBG of the thermometric probe.

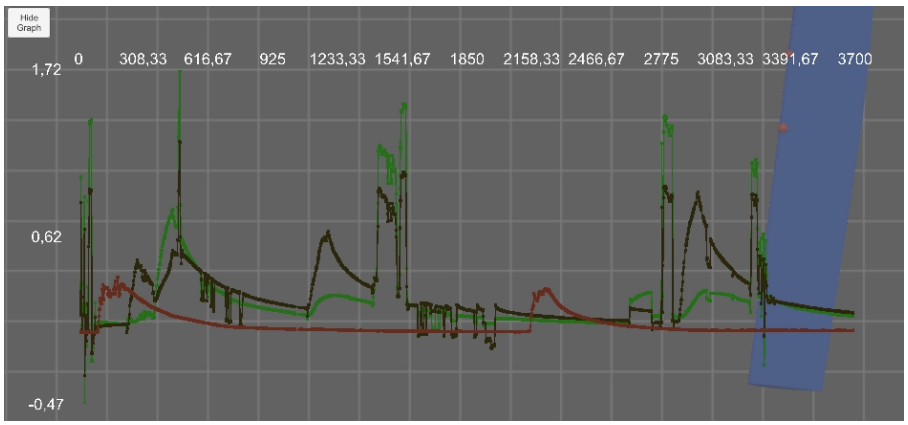

Figure 5.14. Tension Bar + Thermometric Probe data collected during the experiment. The lines in green and black represent the data coming from the aluminum bar, while the brown one represent the data coming from the thermometric bar.

The trend of the thermometric probe is similar to the one of the first experiment, spike when it is warm-up and branch of the hyperbola while is left to cool down. However, the trend of the curve for the bar is different because it involves the process of the heat-up/cool-down of the material hosting the FBG. The cool-down period for the bar is shorter because the aluminum is more thermally conductive than the plastic hosting the thermometric probe. During the cool-down of the bar, little strain in the proximity of the sensors was performed in order to test their sensitivity. This behavior results in drops from 0.2 to 0.4 or 0.65 0.8 in the graph, which can be seen in figure 5.15.

The wavelength data collected have the same trend of the previous tests. The idle wavelength of the bar are 1547 nm and 1530 nm for Ch1Gr1 and Ch1Gr2 respectively, while for the thermometric probe (Ch2Gr1) has 1540 nm as idle wavelength. Table 5.3 shows the data collected during this test.

	Ch1Gr1	*Ch1Gr2*	*Ch2Gr2*
Idle value (nm)	1547	1530	1540
Mean value (nm)	1546.9531	1530.8642	1540.0711
Max value (nm)	1548.434	1531.848	1540.38
Min value (nm)	1546.245	1530.317	1540.02

Table 5.3. Wavelength data collected during the Tension Bar + Thermometric Probe experiment.

The delay is in line with the previous result, the mean delay of 160 ms with some spikes that go up to 2438 ms, and in fast cases, the delays decrease to 1 ms delay.

5.4 General results

The scope of these tests was to measure the overall delay of the solution and to check if it is suitable for the project. As expressed in figure 5.3, the delay has been calculated as the difference of the timestamps collected from the visualization of the data in the Viewer with respect to the data processed in the Middleware. These data are relevant to the thesis scope because the Server

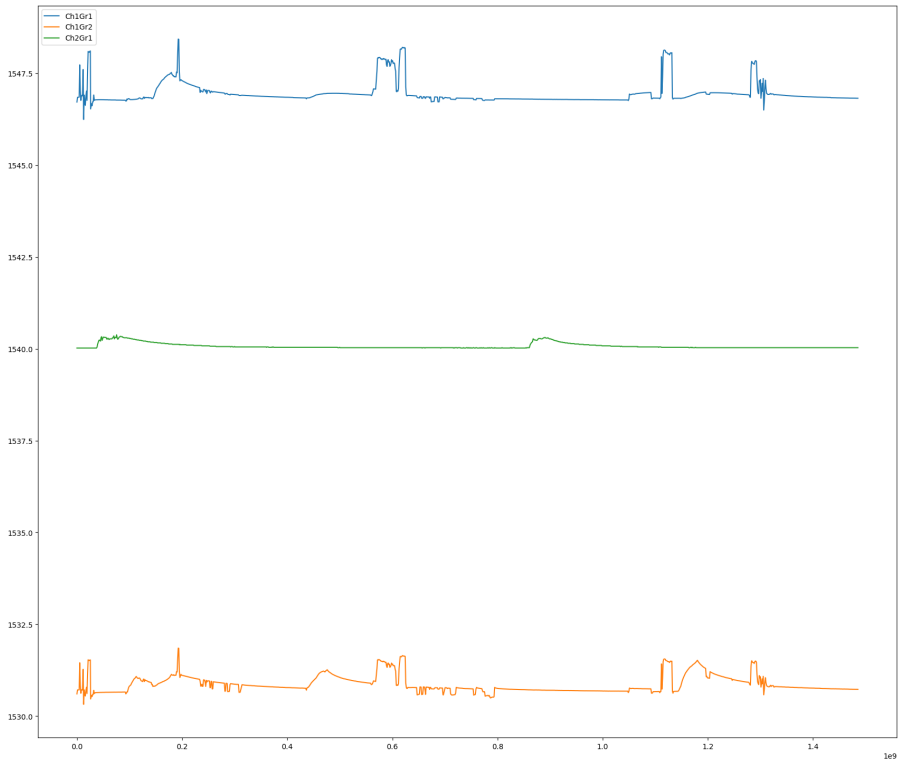

Figure 5.15. Post processed graph of the Tension Bar + Thermometric Probe experiment.

is hosted in the same module of the Middleware, and the delay due to this connection is irrelevant. Table 5.4 shows the mean delay and its standard deviation for each test case. For the laboratory test, where the equipment changed only on the sensor side, it is possible to spot a pattern on the tolerance of the system.

Test name	*Mean Delay (ms)*	*Standard Deviation (ms)*
Synthetic Test	1100	725
Thermometric Probe	183.72	182.15
Tension Bar	133.81	142.56
Thermometric Probe + Bar	160	198.80

Table 5.4. Overall delay from all tests performed, focusing on the mean value of the delay and its standard deviation.

Concerning the previous solution, the delay has increased. The factors that influenced the most on the overall delay are the following:

- Wi-Fi Bandwidth from the VR application to the Server;
- insertion time on MongoDB instance;
- data replication on MongoDB instance for creating Change Stream feed;
- Desktop/AR application computation;
- the number of sensors to monitor.

Chapter 6

Conclusion

This final chapter will draft some ideas for future improvements of the project and the conclusions about the work made.

6.1 Future work

This section highlights all the possible improvements and new features that can be implemented in the future. All the components of the thesis have been developed using Git, which allows coordination between the developers, bugs tracking, and integration of new features. All repositories are hosted on Bitbucket and managed by the DAUIN department.

6.1.1 Missing features

The entire project is working as design, however, it still requires fixes and improvements. The network connectivity between the components was designed to support the 4G/5G network but, due to the COVID pandemic, tests were made using a Wi-Fi network. With the 4G/5G network, the monitored system can be carried outside the low range field offered by Wi-Fi and, in the case of an aircraft, it can allow longer flight sessions. The interrogator library could also be improved to enhance data throughput and system stability. In particular, an abstraction layer, useful to isolate any implementation from the chosen interrogator device, could be implemented. In the following subsection, there will be proposals about other possible future work.

6.1.2 Cloud-Based Solution

An improvement of the proposed solution could be to integrate this implementation with the one proposed in previous iterations of the project [1]. In this thesis, it was proposed a solution using an in-house Server that allows having full control on the feature to implement. However, resource allocation and server maintenance could slow down the development process. Switching to a Cloud-Based solution, there is no need to manage space allocation and maintenance of the infrastructure because it is handled by the service provider. Other strong points on implementing a Cloud-Based solution are:

- Data accessibility;
- Application scalability;
- Security and data backup;
- Optional services for real-time and offline data monitoring.

The solution proposed was KaaIot, a complete IoT framework that interfaces to the cloud services hosted by a remote cluster. It provides a wide range of features as customized micro-services, allowing the developer an easy implementation of new features. The problem is that the open-source version was deprecated years ago in favor of other forms, like platform-as-a-service (PaaS) or self-hosted, both not free.

6.1.3 Viewer performance

For the development of the Desktop/AR application, the Unity Engine has been chosen as the real-time 3D creation platform. Unity is appealing for creating 3D Desktop applications thanks to its easy-to-use interface and its great support for mobile, VR/AR, and cross-platform game development. It is based on C# programming, which does not allow any memory management by the developer. No control over memory means that the garbage collector can be triggered at any time, ruining performance: this can be critical in the case of real-time health monitoring systems. One solution can be to switch to Unreal Engine, which recurs to C++, a lower-level programming language that provides more control over memory management to the developers. On top of this, Unreal Engine gives to developers full access to the C++ source code, allowing any editing and upgrading of the system. The only drawback of using a solution based on Unreal Engine is the less effective Hololens development tool. Unity, in the other hand, provides a more mature platform thanks to their partnership with Microsoft.

6.2 Conclusion

The purpose of this thesis was to enhance the Photonext project, providing a reliable solution for real-time and non-real-time data. The expected goal was to design and build a full system able to retrieve data from a generic interrogator, enabling real-time data visualization for structural health monitoring and temperature analysis. During the development stage, many different aspects were considered, from the analysis of the previous work to the modeling of a solution compliant with the requirements of the project. The proposed solution, based on MongoDB and Change Stream, is the result of a careful and deep study of the currently available technology. Each layer of the platform was analyzed to determine the required features and to identify potential development issues.

The project is part of a collaboration with the DAUIN (Department of Control and Computer Engineering) and DIMEAS (Department of Mechanical and Aerospace Engineering) of Politecnico di Torino, as part of the POLITO Inter-Departmental Center for Photonic technologies (PhotoNext). The entire framework is designed to analyze and monitor an aircraft engineered by the ICARUS team of the Politecnico di Torino, for the scope of their project called "ANUBIS".

The final result performs as expected, with overall good performance in respect to the initial requirements, both in real and simulated case scenarios. Both interfaces were considered easy to understand, and the different features pretty straightforward to learn and remember.

Bibliography

[1] Mauro Guerrera. Algorithms and methods for fiber bragg gratings sensor networks. Master's thesis, Politecnico di Torino, 2018.

[2] Maria Giulia Canu. Mixed Real-Time Visualization Framework for FGB IoT sensors. Master's thesis, Politecnico di Torino, 2019.

[3] Photonext. Photonext. http://www.photonext.polito.it/it/.

[4] Nobuo Takeda and Yoji Okabe, "Fiber bragg grating sensors in aeronautics and astronautics", Fiber Bragg Grating Sensors: Recent Advancements, Industrial Applications and Market Exploitation, January 2011 DOI 10.2174/978160805084011101010171

[5] Hong-Nan Li, Dong-Sheng Li, and Gang-Bing Song, "Recent applications of fiber optic sensors to health monitoring in civil engineering" Engineering structures, 26(11):1647–1657, 2004 DOI 10.1016/j.engstruct.2004.05.018

[6] Ebrahim Al-Fakih, Noor Azuan Abu Osman and Faisal Rafiq Mahamd Adikan, "The Use of Fiber Bragg Grating Sensors in Biomechanics and Rehabilitation Applications: The State-of-the-Art and Ongoing Research Topics", Sensors, 12(10):12890-926, DOI 10.3390/s121012890

[7] Verónica García-Vázquez, Felix von Haxthausen, Sonja Jäckle, Christian Schumann, Ivo Kuhlemann, Juljan Bouchagiar, Anna-Catharina Höfer, Florian Matysiak, Gereon Hüttmann, Jan Peter Goltz, Markus Kleemann, Floris Ernst and Marco Horn, Navigation and visualisation with HoloLens in endovascular aortic repair, De Gruyter DOI 10.1515/iss-2018-2001

[8] Infibra Technologies. FBG sensors overview, http://www.infibratechnologies.com/technologies/fiber-bragg-gratings.html

[9] What is a Fiber Bragg Grating?, https://www.hbm.com/en/4596/what-is-a-fiber-bragg-grating/

[10] Icarus Team Polito, https://icaruspolito.altervista.org/progetto-acc-icarus-polito/

[11] SmartScan© from SmartFibres©, https://www.smartfibres.com/products/smartscan

[12] SmartSoft© from SmartFibres©, https://www.smartfibres.com/products/smartsoft-software

[13] Raspberry pi https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

[14] KaaIot https://www.kaaproject.org/

[15] Replication https://docs.mongodb.com/manual/replication/

[16] M. A. A. da Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque, "A reference model for internet of things middleware" IEEE Internet of Things Journal, 5(2):871–883, April 2018. DOI 10.1109/JIOT.2018.2796561

[17] Snyde A W and Love J D 1983 Optical Waveguide Theory (New York: Chapman and Hall)

[18] Yung Bin Lin, Jenn Chuan Chern, Kuo-Chun Chang, Yin-Wen Chan and Lon A Wang "The utilization of fiber Bragg grating sensors to monitor high performance concrete at elevated temperature" Smart Materials and Structures, 113(13):140-112 DOI 10.1088/0964-1726/13/4/016

[19] Installing the MongoDB C Driver (libmongoc) and BSON library (libbson) http://mongoc.org/libmongoc/current/installing.html

[20] Install the mongocxx driver for Linux http://mongocxx.org/mongocxx-v3/installation/linux/

[21] Install MongoDB Community Edition on Ubuntu https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/