Abstract—This document describes the proposed solution for a software framework and integrated user interface to be deployed in the existing telemetry system installed in the vehicle prototype of IST team that competes in the Formula Student Championship. The solution has two deployment devices, one located in the racing vehicle (referred to as mobile station), and the other located in the pit stop (referred to as pit station). The ultimate goal is to enable the data sent from the mobile station to be stored and displayed to users in the pit station either in real-time or at a later time. Data is acquired from several sensors installed in the car and interconnected by a CAN bus network. This project consists essentially on the evolution of a previous developed software framework to better cater the end user needs. The work comprises important improvements on several modules, such as the communication protocol used between the two stations, the database used to store all the information gathered, the required processing over the gathered data, and the adaptation of a graphical user interface to better fit the user profiles and requisites.

1 INTRODUCTION

In 1998, the Formula Student Championship [1] was born in the UK under the jurisdiction of the Institution of Mechanical Engineers. The goal of this competition is to give the chance for engineering students to demonstrate their technical, engineering design, and manufacturing skills, while developing other capabilities such as team working, time management, among others. Each team is challenged to conceive, design, build, cost, present and compete with a prototype of a single-seat racing car in a series of static and dynamic events. The teams are evaluated in terms of the cost (which must be low), ease to be maintained, reliability, and performance in terms of acceleration, braking and handling qualities of the build prototype.

In 2001, a project (denominated by FST [2]) was initiated by a group of students from IST with the aim of joining the Formula Student Competition. Since its foundation, the team has already built five prototype vehicles, three of which have traditional combustion engines while the lastest two have electric engines.

To provide students with the means for an early and real-time problem detection on the vehicle, and consequent ability to make improvements or adjustments during testing sessions, a series of projects have been developed which consisted in creating the hardware infrastructure and the software application that would make this type of analysis possible.

The existent work counts with a fully featured telemetry system that is installed in the vehicle, which is composed by an embedded-computer connected to a CAN bus and a GPS. The system collects data from several sensors which are interconnected to the embedded-computer through the CAN bus, used to collect the data. This data is subsequently used for analysis by a software in the pits, which receives it by means of a wireless network connection. The hereby presented work, is a follow-up to a series of previous works done related to the software that enables the analysis of sensor data gathered by the telemetry system installed in the vehicle and sent over the wireless network connection to the pits.

2 MOTIVATION AND OBJECTIVES

It is very important to have well defined system requirements when developing any system. The previous framework was the result of a series of independent works focused on some particular characteristic to be added to the system, and not an attempt to build a
single tool for racing analysis tailored to the users. Therefore, it was necessary to gather these requirements and determine the user needs and conditions to meet.

To cope with the collected requirements, several enhancements had to be introduced. The communication module had to be adapted to support multiple sensor samples in a single network packet, and to make it independent of the adopted operating systems.

A new configuration module was introduced to allow the users to manage the sensors used in their analysis, allowing the users define sensors that are installed in the car and respective signal processing formulas, but also define sensors that would represent a mathematical manipulation over the sensors installed in the car.

Improvements were also introduced in the DBMS, namely by changing it, in order to increase the speed performance for storing the acquired data, in a multi-transactional environment.

It was also necessary to define the interface between the system and the users, to promote a familiar working environment where they can work at their best.

This work represents a significant added-value to the work performed by the IST FST team, in the sense that it will enable the team to detect failures in the car and perform the necessary adjustments, in an easier, faster and more flexible way. It also represents an added-value to the data gathered by the car sensors, since the integrated management, manipulation and processing of data, enables the users to have their analysis environment configured to match their preferences.

This project constitutes an evolution of a previous version of the framework already developed for the IST FST team, which can only be used for a limited and static set of analysis. The main goals for this project can be enumerated as follows:

1) In order to allow the usage of the system in multiple operating systems, the implementation of the communication module will be adapted to be operating system independent.
2) Change the DBMS, in order to achieve better performances in terms of speed in a multi-transactional environment.
3) Development of a flexible sensor management system, providing the user with a friendly interface for adding or removing sensors from the desired analysis layout.
4) Integration of a mathematical engine, allowing users to introduce their own signal processing routines.
5) Development of new types of graphs and gauges, to allow the analysis of data in several kinds of views suited for specific purposes.
6) Integrate all components into a unique software framework, with a dedicated graphical user interface.

3 RELATED WORK AND SUPPORTING TECHNOLOGIES

3.1 Industrial/Commercial And Academic Solutions

3.1.1 McLaren

As the standard ECU supplier chosen by the FIA, McLaren plays a key role to all teams in Formula 1 today. Alongside the ECUs, a software system called Atlas was developed with the goal of enabling teams to obtain, display, and analyse nearly real time data that is received from the car or to perform posterior analysis on logged data. Atlas allows users to access and visualise data in a diverse set of views, including Waveform, Circuit, Bar, Numeric, Scatter, Loadmap, Histogram, Summary, FFT, Map, and InPlace. Data is received, displayed and analysed, and the user is able to select the data to be presented from a browser, or drag from another display. Furthermore, data can be easily navigated using a specialised scroll bar, which provides users with a graphical timeline that allows them to select and see information of a specific time or distance in a specific lap. This software also allows the users to create and manipulate their own functions, based on the type of received data; to display the result in different views provided by the system; and to automatically check the state of the car and engine. Session data can be exported and imported in various formats, including Matlab. Additionally, by using a specialised DLL, it is possible to write specific drivers to access other formats of data.

3.1.2 MoTeC

MoTeC is a top level motorsport technology systems manufacturer founded in 1988, dedicated to the design of efficient, reliable and versatile engine management and data acquisition systems. Among its products, the MoTeC i2 Standard and i2 Pro and the MoTeC Telemetry Monitor are the ones most relevant to this project.

The Motec i2 Standard and i2 Pro are the two MoTeCs tools that enable analysis of previously logged data. The first one is freely available to all customers, and offers the read of logged data facility from a MoTec Data Logger or ECU. The second tool requires an optional Pro Analysis upgrade or Feature License, and provides the additional functionalities of advanced mathematics, multiple overlay laps, and unlimited components, workbooks, and worksheets.

The MoTeC Telemetry Monitor tool enables real time monitoring of the vehicle conditions. The data is transmitted from the data acquisition system installed in the vehicle to a PC in the pits. This allows the engineers to monitor its condition on the fly, enabling a faster problem detection and preparation of changes to improve the racing performance.
3.1.3 Academic Solution

The previous academic solution that was developed at IST also provides the means to acquire and analyse data gathered on the vehicle, either in real time or after a running session. This solution is a result of an on-going academic work performed by several students in the context of their master thesis. However, it is mostly an aggregation of various independent and not integrated software solutions, each one developed with a specific goal in mind, to provide or demonstrate some particular functionalities of the system. The existing solution has the capability to read data gathered in the mobile station, either coming from a wireless communication channel transmitting real time data, or from a data file containing previously logged data. This tool also provides the means for a subsequent simple data manipulations, since the data gathered by the several existing sensors is sent in different formats. The necessary transformations over the data are done regarding each specific type of sensor. However the set of possible transformations is static and the user is not able to change it, which forces the set of sensors in the car to be also static.

The application provides a set of graphs and gauges composed by: a car widget, for quick observation of the state of the car; dial gauge widget; bar gauge widget; and time graph.

Additionally, there is still the possibility to analyse the data without running the application, by exporting it to a PDF file.

3.2 Supporting Technologies

In order to achieve the defined objectives, some technologies were used that allow for the implementation of the system.

Starting with Qt Framework [9], facilitating with the development of the graphical user interface and operating system portability.

The Qwt Library [10], which also helped in the development of the graphical user interface, particularly by providing GUI elements useful for presenting the information from the sensors.

The PostgreSQL [8], which was the DBMS used to store the data.

And finally the muParser [11], a library which was helpful in parsing and evaluating mathematical expressions.

3.3 Existing Hardware Infrastructure

Given the need to detect early signs of failures in the vehicle, make adjustments and improve the vehicle performance, it was decided by the FST team to create a system that would enable the transmission of information from the car to the pits for subsequent analysis. This work will use the existing hardware infrastructure which consists on two platforms, namely the set of components installed in the vehicle, called Mobile Station Platform, and a personal PC or laptop located in the pits, called Pit Station Platform. The communication between both stations is ensured by a IEEE 802.11 WiFi transceiver which is installed in the pits alongside the pit station platform.

The mobile station platform comprehends the aggregation of five independent parts that support the mobile station side of the presented framework. These are a single board computer [6] which controls the operations in the car, a CAN bus [5] used to acquire data from the sensors, a CAN to USB transceiver [6] used to convert data from the CAN bus to USB, a GPS device [7] used to track the vehicle position, and a communication infrastructure [6], used to enable the transmission of data between both stations.

The pit station does not represent any specific part of hardware. In most situations, this will be a computer belonging to some member of the IST FST team, which means a recent PC or laptop. The data gathered in the mobile station is received, stored and displayed to the users in this station.

4 System Architecture

The proposed solution comprises two important units: a mobile station, where data is acquired and transmitted, and a pit station, where the data sent from the mobile station is stored and analysed.

In this architecture, communication is performed in both directions, both from the mobile station to the pit station and vice-versa. However, the most important communication flow is from the mobile station to the pit station since data is acquired by the former and stored in the latest, creating a client-server style architecture. Figure 1 illustrates a layered view of the overall system architecture for the proposed system, identifying a boundary which represents the subset of modules that are affected by this project, and additionally the bridge of communication between both stations.

4.1 Communication

The communication between the two stations is performed by using a Wi-Fi connection. This is accom-
plished by having a Wi-Fi USB adaptor connected to the mobile station and by having a wireless router next to the pit. The communication flows from the mobile station to the pit station through the router [6]. Both stations send and receive packets. This means that while the team is performing a test session, the vehicle will be sending messages to the pit station and vice-versa. However, during testing sessions the mobile station will mainly act as client, while the pit station will mainly act as server. This is justified by the fact that relevant information about sensor’s measurements is only sent from the client (the mobile station), to the server (the pit station).

Although the new version of the software framework also adopts the existing communication protocol (see [6]), some improvements and modifications have been introduced. These relate to the design of the protocol, but also to some specific implementation details.

Regarding the design of the protocol, it was agreed with the FST team members to change the structure of the used network packets. This modification was necessary in order to make the communication protocol as independent as possible from the content transmitted between the two stations. This way, it is possible to change what is transmitted in the packets without changing their structure. Particularly, it allows to send data from an arbitrary number of sensors in just one single network packet, thus avoiding network delays introduced by transmitting a bigger number of packets. The new structure is illustrated in Table 1.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Number</td>
<td>The number of bytes indicated by the size field</td>
</tr>
<tr>
<td>size</td>
<td>Data (sensor id and measurement)</td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 1: New packet structure.**

- The packet number field is used to assure the order of the packets and to detect lost ones;
- The size header field indicates the length (in bytes) of the data field;
- The data field is where the identifiers of the sensors are placed alongside their measurements;
- The checksum field is used to ensure the integrity of the packet.

Regarding the implementation of the protocol, it was changed to enables the software to be used on multiple operating systems.

4.2 Mobile Station

This is the part that is responsible for acquiring data from the vast set of sensors connected to the CAN bus and from the on-board GPS device, as well as to send this data to the pit station. The main functionalities of the mobile station are: reading data from the CAN bus, reading data from the GPS receiver, transmitting the data over the network. Its architecture is depicted in Figure 2.

![Fig. 2: Architecture of the mobile station side of the framework.](image)

The depicted elements are described as:

- **Wireless USB Device**: Used to ensure a network connection in the mobile station.
- **Wireless Communication Protocol**: Used by the framework for the transmission of messages through the network. This directly uses the device mentioned above, for transmitting and receiving messages over the network.
- **Framework Communication Protocol**: Deals with all communications between both stations at the framework layer. This module will be mainly concerned with sending the data gathered from the sensors. The new version counts with the already described adaptation to include the improvements outlined in the previous section. It works together with the Wireless Communication Protocol and the Wireless USB Device. In practice, this was the only module of the mobile station that was affected by this work.
- **GPS Receiver**: Performs the communication with the GPS satellites, calculating the absolute position of the mobile station on Earth [7].
- **GPSd**: Independent daemon and device driver that runs in the operating system layer, which makes the interface between the GPS receiver connected to the mobile station and the framework. It reads coordinates data from the receiver and converts it to convenient data structures, which are passed to the framework [7].
- **GPS data acquisition**: Receives the coordinates data from the GPSd daemon and formats it accordingly, in order to be subsequently used by the
Framework Communication Protocol, before being sent to the pit station [7].

- **CAN/USB transceiver**: Connected to the CAN bus, it allows the sensors installed in the car to have their measurements transmitted to the single board computer. This is done by transferring the data received from the CAN Bus into the USB bus.

- **CAN sensors data acquisition**: Responsible for receiving the sensors data from the CAN/USB transceiver. Just like the GPS data acquisition module, this module also formats the received data accordingly, by using the Framework Communication Protocol to send it to the pit station.

Only one module, the Framework Communication Protocol, is affected by this work. In the previous version of the framework, this module used the network packet structure presented in [6]. It forced sending one network packet per measurement gathered from the sensors. To overcome this, the design of the Framework Communication Protocol was changed, particularly to accommodate a new network packet structure, which would allow data from an arbitrary number of sensors to be sent at once.

### 4.3 Pit Station

This is the part of the system that is responsible for receiving, processing and displaying the data from the sensors and GPS receiver to the users. Its main functionalities are: displaying information to the users, store sensor data on the database, managing the data stored in XML files, processing data from the sensors and from the GPS device. The pit station structure can be seen in Figure 3.

**Fig. 3: Architecture of the pit station side of the framework.**

The elements contained in the architecture of the pit station are described as follows:

- **Wireless Communication**: Communication infrastructure contained in the pit station, which enables the transmission of messages to the mobile station.

- **Framework Communication Protocol**: Deals with all communications between both stations at the framework layer, thus being responsible for ensuring the reliable transmission of data over the network, performed when data is being acquired in real time. The new version also counts with its adaptation to include not only the improvements outlined in Section 4.1, but also with the introduced modifications to provide an operating system independent application. It works together with the Wireless Communication module.

- **Database Management System**: This is the module that represents the actual database. The framework uses this module to make all data related to the sensors and the GPS receiver persistent. It provides easy means for accessing previously logged data and analyse it at any time. This module is cross platform, hence available for the operating systems required by the end users.

- **Database Operator**: This module is responsible for all operations, storing and querying, related to the manipulation of the sensor data over the database at the framework level. Further more, it makes use of the Database Management System to achieve this purpose, and enable the data to be available at any time. The data to be stored belongs to the sensors installed in the car as well as sensors composed of mathematical manipulations over the formers.

- **XML Data Manager**: This is the module mainly related to the flexibility provided to the users in what regards to their preferences. These preferences may include sensor information, information related to the GUI, or user personal preferences (such as loading the last project at start-up). This is mainly useful, for instance, to enable the GUI to adapt to each user profile and display only the required information. This allows the user to choose which information should be loaded, corresponding to the sensor data, and hence avoid loading unnecessary data. It uses information gathered from user input, to manipulate data in the corresponding XML files. It also provides interfaces for all processing operations related to read, write and update user preferences into these files.

- **Sensor Data Processor**: The aim of this module is to process all the data related to the sensors connected to the CAN bus at the mobile station as well as the data representing posterior manipulations over these sensors, before this can be stored and displayed. This module uses the XML Data Manager module to get any information related to the sensors that it needs to perform the operations over their data, and the Database Operator module to store any data in the database.

- **GPS Data Processor**: It is responsible for processing and storing all the data related to the positioning of the car, gathered from the GPS receiver in the mobile station [7].

- **Operation Manager**: This module has two main
purposes: one consists on being interface between the Graphical User Interface module, and the remaining modules (except the Framework Communication Protocol), when some user initiated operation needs to be performed (e.g. load previous session), or to fetch data to display to the users; the other is to work as an interface between the Framework Communication Protocol and the Sensor Data Processor modules, when data is being acquired in real-time. Basically, it coordinates all the operations performed in the framework, either initiated by the user or caused by new data arriving from the mobile station, by issuing the corresponding operation to be performed on the module responsible for the task when an external event occurs.

- **Graphical User Interface:** This module is the direct point of interaction with the end users, where they request the operations to be performed, and analyse the data that is displayed. It manages all the data associated to the user interface, provides the users with means to adapt the interface to their needs, and works alongside the Operation Manager to adjust to this set of needs, by requesting the necessary information to generate a representation to be displayed to the users. This module makes use of several widgets to provide the user with all of its functionalities, which can be divided into three categories: manage information related to a particular session (e.g. sensor information, user preferences, mathematical manipulations); display of graphs and gauges showing sensor data; display of sensor data values alongside other session parameter values defined by the user.

In the previous version of the framework, This station lacked key features that prevented the team from using the application. Namely, the team needed a cross-platform framework, since not all team members use the same operating system; the set of features available did not fulfil all the users requisites, since they were not able to manage the information related to the sensors through the application; the graphical user interface, was not designed to accommodate the user working styles.

For the new version, these difficulties were overcome by using a supporting application framework to help making the pit station implementation operating system independent. The implementation counts with new modules that allow the users to manage all the information related to sensors through the application, and a new graphical user interface that is designed to adapt to each user profile and to allow the users to perform the needed operations for the type of analysis that is hereby considered.

5 **Pit-Station Implementation**

The pit station is where the data gathered from the sensors is stored and displayed to the users. That being said, the main objectives of the pit station are concerned with providing the users with a comfortable working environment, while providing the required set of functionalities for them to perform a useful analysis.

5.1 **Cross Platform Application Framework**

In order to assist in the development of the pit station, the Qt Framework was used. It was adopted, because it provides easy means for creating graphical user interfaces, and because it provides support for developing cross platform applications. Hence, this framework was used mainly to fulfil the requisite of making the pit station implementation independent of the adopted operating system, and to make the graphical user interface development easier.

5.2 **Framework Communication Implementation**

The only relevant requirement that has to be satisfied regarding the implementation of the communication protocol is concerned with the portability of the software. This was satisfied by using the Qt’s sockets API.

The execution of network operation is performed in the context of a dedicated thread. This means that, this module lives in one dedicated thread. In order to communicate with other threads, a signal/slot mechanism provided by Qt API is used. Both the process of receiving and transmitting messages are performed using this mechanism. There is a dedicated thread for network operations which has particular methods called slots, that are executed upon receiving specific signals for message transmission and reception. When a network message arrives to the pit station, a specific signal is emitted by Qt, and a slot in the network communication thread is executed to process the new packet. After processing the packet, it emits a signal, to mark that new data is ready to be processed. This data is then processed by another thread, responsible for the data processing. As for transmitting messages, a slot is executed in the network communication thread, upon receiving a signal marking that a message should be sent. The implementation uses the class QUDPSocket provided by Qt Framework, which provides platform independent methods for network operations.

5.3 **Database Management and Implementation**

All the data received from the mobile station is made persistent through the PostgreSQL DBMS, to enable its posterior analysis. This provides a standardized mechanism of storing, accessing and changing data. Since this access to data is made by using SQL, this also provides independence between application, since this is a well known standardized language for database management.
This module is responsible for using PostgreSQL to make all the gathered data persistent in the database. Its implementation was entirely made by using the QtSQL interface provided by Qt Framework. This interface provides a set of operating system independent operations to manage data through the DBMS.

This module is used by the Operation Manager and Sensor Data Processor modules, which issue all operations over the database to be performed.

5.4 XML Data Manager

This module implements the new management mechanism that provides the users with the necessary flexibility to make configurations over the data. All the information regarding these configurations is stored in dedicated XML files, from which they are retrieved when necessary. These contain information such as: user preferences, namely if the last project should be loaded upon system start-up or automatically saved on exit; the tuning parameters used in the car for a specific session, such as the camber angle for a wheel; the existing GUI elements associated to some session; the list of existing sensors, including those obtained from mathematical manipulations over other sensors; the implicit post-processing or data conversions of the sensors installed in the car.

The implementation of this module was performed by using the Qt XML module. This module offers a set of operations to read and write data in the XML format. The choice for the XML representation was based on the desire for an easy and portable mechanism to transfer some kind of application definitions between the end users. That said, by using this approach the users can easily exchange any information contained in the files or even the files themselves between one another. Since XML is a user readable format, there is even the possibility for the user to interpret the content of some file and transmit to another user. In line with this, it is also easy to change the content of a file, even without the support of the framework.

5.5 Sensor Data Processor

This module implements the set of operations that are performed in the pit station over the sensor data. This data can either belong to a single sensor, meaning one installed in the car whose data is not dependent on other sensor data, or to a combinational sensor, whose data is defined as a composition of mathematical manipulations over single sensors data.

This processing is performed by using the muParser, a library that parses and executes mathematical expressions.

For the single sensors, this means applying the post-processing formulas defined by the users, that should be executed on the received data before it is stored and displayed to the users.

For the combinational sensors, this means applying user defined formulas that combine data from one or more sensors. For the case in which the formula is composed of data from more than one sensor, an interpolation operation is performed over the included sensors in order to obtain the values corresponding to the intersection of their timestamps. Only then is the mathematical formula evaluated using these interpolated values.

5.6 Operation Manager

This module coordinates all the operations over the data. It manages all the modules, with exception to the GUI and Framework Communication Protocol modules, which live in their dedicated threads. Its main task is to act as a broker, between these two modules and all the other modules.

This module implements a set of slots that are executed in response to user interaction over the GUI, and to process newly received sensor data from the mobile station. The communication between the two above mentioned modules and the Operation Manager is made through the signal/slot mechanism provided by Qt Framework. As for all the other modules, they are controlled by this module, and hence live in the same thread. Thus, the Operation Manager just issues the correct operation to be performed on one of these modules, when a signal is received from the GUI and Framework Communication Protocol modules. It also emits a signal to the GUI in case any information needs to be displayed, as part of the operation requested by the GUI.

From the point of view of the GUI module, this is the entity from which it gets the required information to be displayed to the users. From the point of view of the Framework Communication Protocol, this is the entity that starts or ends the communication between both stations and processes the relevant data that is received.

5.7 Graphical User Interface

This is the module that implements the interface with which the user directly interacts with to perform their analysis. The GUI provides several mechanisms, which provides the users to view and manipulate the data.

The GUI, shown in Figure 4, is divided into three main zones:
In the upper most part, it has a tool bar which provides several possible general data and session management actions. This is where the user is able to perform operations such as: creating new sessions, or loading previous ones; start live mode; configure the sensor information; zoom, in and out, the time range to be displayed in the graphs.

In the central part, it is where the relevant information is displayed. This comprises: a slider, used to browse data in the time range of a session; a tab widget, where the graphs can be displayed in several tabs; and an area used to display the several available gauges.

In the lower part, there is a status bar, which provides the user with information about the type of project that he is currently working, and the specific instant in time he is analysing.

All the GUI elements are supported by both Qt Framework and Qwt Library. The latest is responsible for providing the technical elements, which are directly used to display or browse through the sensor information. Both libraries offer platform independent APIs that can be used in any of the target operating systems. The GUI is based on a set of operations based on the signal/slot mechanism provided by the Qt Framework, which deals with the necessary updates to the information displayed on screen. When the user requests an operation, a signal is emitted to the Operation Manager, so that it executes the required action and generates the information to be presented to the user by the GUI. Upon the generation of this information, the Operation Manager emits a signal to the GUI and a corresponding slot is activated, to handle the representation of the received information. Hence, the GUI module is structured as a set of slots, that respond to signals that are generated either by user input or by the Operation Manager, and a set of signals to request information from the Operation Manager.

6 RESULTS

6.1 Testing Environment

The winter season is usually used by the Formula Student teams to develop and modify their prototyping vehicles in the lap. As a result, the opportunities to test and evaluate the car, and the proposed framework in particular, in a real racing environment were very few. Consequently, most of the conducted evaluations had to be done in the FST laboratory in IST. The materials used for these test scenarios were:

- Mobile station platform, installed in the lab.
- A set of sensors of the car, used to evaluate the platform in the real-time operation mode.
- Personal computer to act as pit station.
- Communication infrastructure composed by the wireless router that manages the communication between both stations, and the Wi-Fi USB adapter that enables the communication to and from the mobile station. The first was installed in between both stations, while the second was connected to the mobile platform mini-computer through a USB interface.

6.2 Experimental Tests And Results

6.2.1 Database Insertion Speed

To ensure an effective storage of the received data, the rate of insertion of sensor samples in the database must be enough to cope with the maximum rate of reception of data from the network (see [6], [7]). To measure the capability of the system to cope with the data reception rate, a sample testing procedure was used to ensure the validity of the database insertion speed.

The performed test measures the database insertion speed for the new implementation by using the QtSql module. The test consisted in evaluating the capacity of the system to store data received from the mobile station. The performance is evaluated by using several sampling rates. A subset of the results obtained is illustrated in Figure 5. It can be observed from the graph that the database management implementation can easily handle the rate at which data arrives at the system.

Fig. 5: Speed performance of PostgreSQL, evaluated with several sampling rates.

6.2.2 Network Communication Speed

To evaluate the performance of the framework communication protocol, a sample test was performed in which
data was transmitted from the mobile to the pit station. The test consisted in transmitting a set of data with 16 MB using several network packet sizes, and evaluate the time taken to transmit the whole data. The results are depicted on Figure 6.

![16MB Data Transmission](image.png)

**Fig. 6:** Transmission time required to send 16MB to the pit station, by using several network packet sizes.

It can be observed from the figure that there is a significant network overhead when the packet size is small, causing a larger number of packets to be transmitted. Thus, it can be concluded that there is a clear advantage of including more data within a single network packet.

### 6.3 Mathematical Engine Performance

The evaluation speed of the `muParser` was analysed, in order to make sure that it would be enough for the gathering rate of the sensor measurements. A set of benchmarks, illustrated in Figure 7, were used for this purpose, showing the performance of this library for some of the processing formulas used by the team for post-processing the received data. The test consisted in evaluating this set of formulas, considering different data reception rates. The library should be able to cope with the maximum reception rate (see [6], [7]).

![muParser speed performance](image.png)

**Fig. 7:** `muParser` speed performance, evaluated for several sampling rates.

As it was expected, there is an increase in the time taken to evaluate the expressions, as the sampling rate grows. Nevertheless, it can be observed that for the maximum possible data acquisition rate (see [6], [7]), the time needed for evaluating the expressions is not an issue.

### 6.3.1 Multiple Operating System Support

One of the main objectives of this project was to turn the pit station side of the framework into an operating system independent application. As described earlier, this was obtained by using the Qt Framework in the development of the new version, which provided the necessary cross-platform API to implement the needed functionalities.

Hence, tests were conducted in all the targeted operating systems to prove the framework capability of being used in all of them. Figure 8 shows the framework look and feel for the different target operating systems. The cross-platform features are managed by Qt Framework in all of them.

![Framework main screen under Linux](image.png)

(a) Framework main screen under Linux.

![Framework main screen under Mac](image.png)

(b) Framework main screen under Mac.

**Fig. 8:** Framework look and feel, for analysis performed under Mac and Linux.

### 6.3.2 Sensor Data Analysis

The most important characteristic of the framework is to provide the means to analyse the data gathered in a testing session, from the sensors installed in the car. Figure 9 illustrates an offline analysis session, demonstrating the capacity of the system in handling the data that is gathered from the car sensors in a practice session.

It can be observed that the GUI can accommodate a diverse set of graphical widgets to display the information gathered from the sensors. The users are able to manage
the graphical layout, by choosing which widgets they want to display information and which data they want to be presented in these widgets, and hence, adapt it to their specific needs.

Fig. 9: Offline session with scatter plot. This type of plot is an X-Y graph, in which each axis is filled with values from one sensor. This graph is used to evaluate the variation of one sensor in relation to another.

7 Conclusion
This work aimed to offer an added value to the FST IST team analysis sessions tools, used to analyse the data that is gathered from the car sensors in order to prepare its vehicles for competitions. In particular, it aimed to obtain a great improvement on the quality of the data analysis and a consequent reduction in the time needed to prepare the vehicle prototypes. This objective is achieved by providing a framework capable of performing several management and processing operations over the data gathered in the mobile station, with an integrated GUI where the users are able to manipulate the data. It is a follow-up on a series of independent works that were previously performed, with the goal of integrating them all in a single and unified framework with a dedicated GUI.

It offers the users with the possibility to analyse data in real-time and load data that was previously logged. The user is able to manipulate this data by adapting the framework to his needs, namely, he can chose which sensors to have their information displayed, which post-processing operations to be performed on the gathered data, define mathematical formulas combining data from several sensors, and change the graphical layout by choosing the most suitable widgets to perform the analysis.

8 Future Work
This project has been initiated some years ago, and since its inception it has been target of several improvements. However, there is always room for extra improvements, and these do not mean improving only the system capabilities, such as performance, but also extending the set of provided features, by adding more relevant functionalities, or by improving system usability.

The new version of the framework does already cater the most relevant needs for the team to perform a useful analysis, either in real time or from data previously stored. The team can already take some benefits from the analysis made in the framework, and have its productivity improved. The following items describe possible improvements to be made to increase the advantages of the framework for the analysis sessions:

- Although the existing graphics that are made available to the user already fulfils the user’s needs, the set could be extended by adding 3-D graphics.
- Add a zoom-by-dragging mechanism, which would allow the user to pick a zone in the graph, which he wants to zoom in. While selecting the zone, this would be highlighted until user finishes his selection, after which the graph should be zoomed in to show the selected zone in more detail.
- Put more data in network messages sent from the mobile station, by including data from more than one sensor in a single network packet. This would reduce the number of messages sent from the mobile to pit station and vice versa.
- Video and audio should be introduced into the framework, in order to be able to give instructions to the driver while performing a testing session, as well as to analyse how the driver is behaving on the track, by view images from the race in real time.

References
[9] Qt Development Framework, Qt Reference Documentation, qt-project.org/doc/qt-5.17