INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Formula Student Racing Championship: design and implementation of the management and graphical interface of the telemetry system

Paulo Fernandes Mendes

Dissertação para obtenção do Grau de Mestre em
Engenharia Informatica e de Computadores

Júri

Presidente:	Doutor Alberto Manuel Rodrigues da Silva
Orientador:	Doutor Nuno Filipe Valentim Roma
Co-Orientador:	Doutor Moisés Simões Piedade
Vogais:	Doutor Alberto Manuel Ramos da Cunha

Outubro de 2011

Abstract

This document addresses the design and implementation of the telemetry system targeted to a racing car that will compete in Formula Student championship. This system is based on two stations: the base station that will stay at the pitstop and the mobile station that will be in the car. The function of the first is to receive, log and present the data received from the mobile station. The mobile station will be connected to an existing CAN-bus network and will transmit, to the base station, the several signals that are acquired by the sensors installed in the car. Due to the nature of this competition, this system must be able to deal with vibrations and bad weather conditions and should be lightweight and have a low cost.

Keywords

Telemetry, Formula Student, CAN-bus, resilient communication protocol, Database Management System, User Interface

Resumo

Este documento aborda o desenho e implementação de um sistema de telemetria para um carro de competição que vai competir no campeonato Formula Student. Este sistema é composto por duas estações: a estação base vai ficar nas boxes e a estação móvel vai estar no carro. A função da primeira é receber, gravar e apresentar os dados recebidos da estação móvel. A estação móvel vai estar ligado a um barramento CAN pré-existente e vai transmitir, para a estação base, os varios sinais adquiridos pelos sensores instalados no carro. Devido à natureza desta competição, este sistema tem de ser capaz de lidar com vibrações e condições de mau tempo, devendo no entanto ser leve e ter um custo reduzido.

Palavras Chave

Telemetria, formula Student, barramento CAN, protocolo de comunicao resiliente, sistema de gesto de base de dados, interface com o utilizador

Contents

List of Figures

List of Tables

1 Introduction

Contents

Telemetry refers to a technology that allows remote measurement and reporting of data. It has been used since 1912, when it was applied to monitor electric power distribution. Meanwhile, its use has been widespreaded to many different areas such as: agriculture, water management, defense, enemy intelligence, motor racing, medicine, wildlife research, retail business, as many others. In all the areas mentioned before, telemetry systems often have a common factor: the ability to provide precise data in real time.

A telemetry system is composed of several components: the sensors, the data storage, the processing unit and normally a communication medium to transmit the data. This makes it a very adaptable idea since its bases are very simple. The basic idea is to have sensors collecting data, that will then be treated by the processing unit. The treated data is then sent to a place where it is shown and stored.

The presented work is concerned with automotive telemetry. In this specific area, telemetry gives the ability to faster achieve better setups of the car. Without this technology the car setup will be mostly driven by the feedback provided by the drivers, with all the limitations that may arise. The motorsport where it is most often used is formula one (F1). Each F1 car has about two hundred sensors. To deal with the amount of data that all those sensors generate, racing teams often require dedicated hardware to deal with such large quantities of information as well as specific, software to process the acquired data. With all this, it is possible to measure the car different setups, allowing a good evaluation to improve overall the performance.

1.1 FST

The formula student competition (FST) challenges university students to design, build, develop, market and compete as a team, with a small single seat racing car. The event is held by the Institution of Mechanical Engineers (IMechE), in partnership with various well known companies in the industry. The idea of the competition is for the contenders to imagine that a firm has engaged them to create a prototype for evaluation. This car should be low in cost, easy to maintain, and reliable [20] [8].

There are three categories in this competition: class1, class1A and class2. Class1 is for fully constructed and running cars. These are evaluated in terms of design, presentation, cost, acceleration, skid pad, sprint, endurance an fuel economy. Class1A is aimed at low carbon emission vehicle, where the teams are free to develop new powertrain technologies that reduce the cars CO2 emissions. The difference from class1 is that the cost evaluation is replaced by a sustainability event. Class2 is for teams that want to enter a vehicle design, where points are awarded for design, presentation and cost [8].

1.1.1 Scope and requirements

The scope of this work is design and implement a telemetry system to be incorporated in a prototype electric car that will compete in formula student competitions. Therefore, the part of this system that will be installed in the car must be able to deal with vibrations, heat, liquids and electromagnetic interference and still be lightweight and cheap. Such mobile station will receive data from a previously installed CAN network and transmit it in real time to the pitstop. Moreover, it should also record such data in a local storing device, to prevent data losses due to communication failures. In the pitstop, the base station should provide the user with the tools that will enable him to analyse the data in an easy way. The data must me Storeyed and easily retrevable.There should also exist a way to solve problems with the system without having to stop the car at the pitstop.

1.2 Objectives

The main objectives of the presented work is to provide a set of improvements to a previously implemented prototype of the telemetry system [4]. The areas that will be focused by the improvement are:

- The communication system, to make it more reliable;
- The data storage: a database management system will be used to allow an easier access and better data consistency;
- The user interface, to give more and different alternatives to present and analyze the acquired data;
- The hardware by using a of more powerfull and flexible hardware, allowing more areas of expansion such as video and audio transmission;
- The creation of a fault resolution system, to allow the recovery while the car is in the track and at the pits.

1.3 Dissertation outline

This thesis is divided in seven chapters. In chapter one the scope and requirements of this work are presented. In chapter two, the state of the art for automotive telemetry is given. In chapter three, the mobile station is presented, giving focus on the connection to the CAN-bus and the hardware and software requirements. Chapter four presents the base station, along with its several components: the database and the user interface. The fifth chapter focus on the communication infrastructure, giving focus to its hardware and to the developed software and configurations created for it. The sixth chapter gives the results of this work, evaluating its several components. The seventh chapter presents a conclusion about the work that was developed.

2 Available industrial and academic solutions

Contents

A telemetry system architecture is normally composed of remote stations acquiring data and transmitting data to a base station. Nowadays the transmition is mostly carried out by a wireless interfaces. It is at the base station where the storage and the analyses of the data are done. A typical architecture of a telemetry system can be seen in figure 2.1.

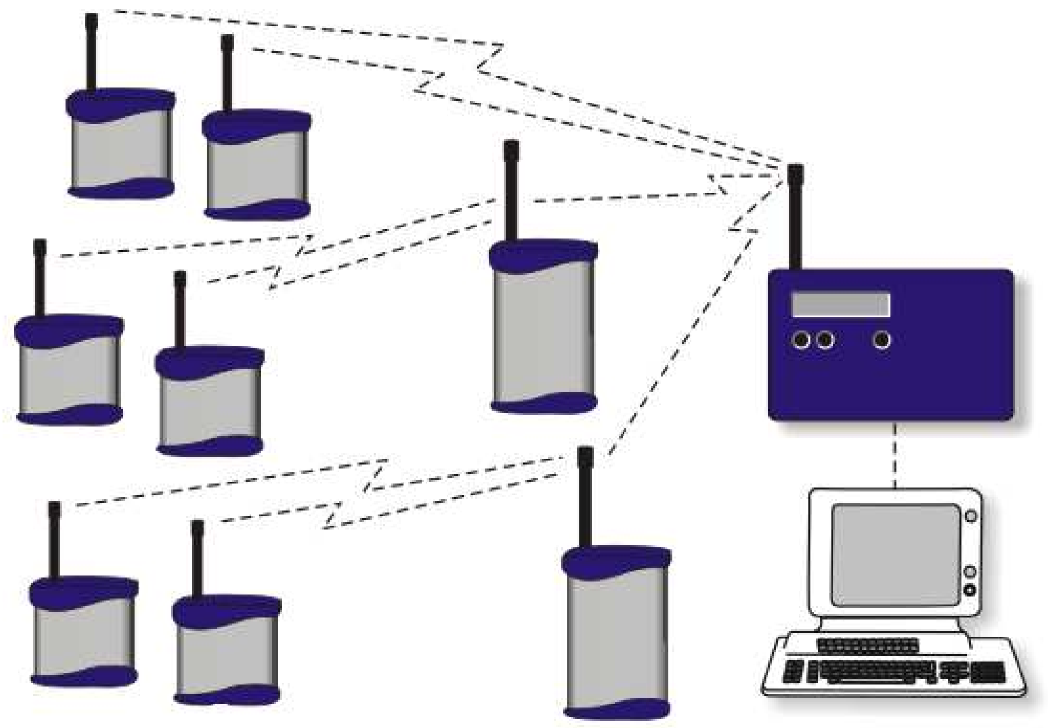


Figure 2.1: Typical architecture of a telemetry system.

2.1 Industrial solutions

Many industrial solutions exist in the market for automotive telemetry. In the following sections it will be presented two existing alternatives, developed by Cosworth and McLaren. The presentation of these solutions will be divided beetwen hardware and software.

2.1.1 Cosworth

2.1.1.A Hardware

Cosworth provides data loggers with up to 128MB of internal flash memory for storing the acquired data (see Figure 2.2). This data can be downloaded through an Ethernet port. To avoid system shutdown due to power losses, it is equipped with a dual redundant power supply. Three accelerometers are incorporated in the unit. Other sensors can be connected to the data logger, such as:

- Accelerometers to analyse the G forces that the car is handling.
- Gyroscopes to analyse the orientation of the car.
- Inertial measurement unit this sensor uses a combination of accelerometers and gyroscopes to measure the velocity, orientation, and gravitational forces.
- GPS to acquire absolute location of the car.
- Speed sensors these sensors use the rotational speed of the wheels to measure the linear speed.
- Laser ride height sensors to measure the height of the car relative to the ground.
- Pressure sensors for oil and fuel to monitor the pressure of the engine oil and fuel.
- Torque sensors to measure the force that the engine is transmiting to the wheels.
- Air temperature sensors to measure the external temperature and the temperature at the entries and exits the engine.
- Infrared temperature sensors to measure the temperature of the tires.
- Linear displacement sensors used to measure the suspension and pedals usage.
- Rotational displacement sensors this can be used to measure the steering angle.
- Lap sensor to know the lap time: several sensors can be used in other to get split times.

Figure 2.2: Cosworth data logger.

In order to provide a have real time telemetry system, a radio transmitter has also to be acquired. This hardware does not offer guarantee of data delivery. To overcome this it is possible to configure the size of the transmitted packages. This way package losses have less impact. The data is acquired by a computer in the pits through a serial port [6]. Such unit can be seen in 2.3.

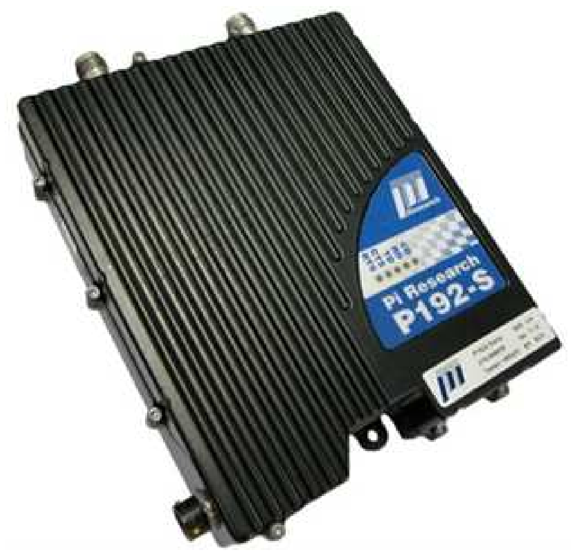


Figure 2.3: Cosworth radio unit.

This vendor also presents a version that works using 2.4GHz wireless Ethernet communication at 54Mbps (IEEE 802.11g) and 11Mbps (IEEE 802.11b). Although providing less range, the bandwith is increased when compared with the solutions that use radio signals. This solution is mainly oriented to offload recorded data from the data logger.

2.1.1.B Software

Cosworth Pi Toolbox software allows the display of data in the form of: maps, dials, bars, tabular outing report, split report, excel report, event display, histogram, navigator, summary and X/Y display. It also allows viewing of video streams fully synchronized to data [6]. A screenshot of this software can be seen in figure 2.4.

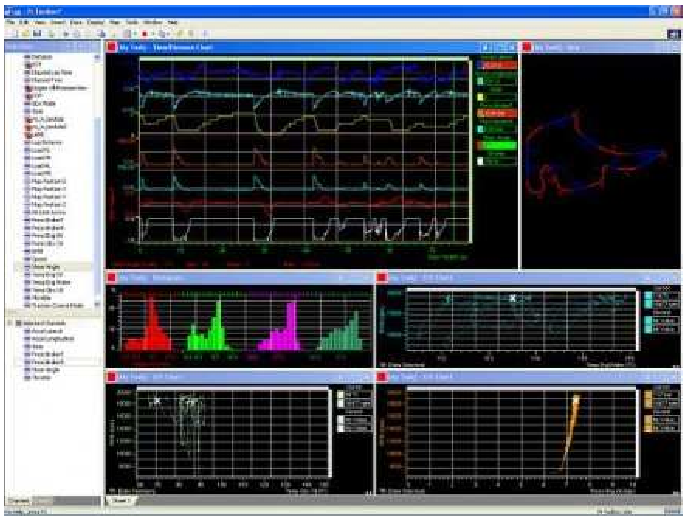


Figure 2.4: Screen shot of cosworth software framework.

This software enables the distribution of data and session configurations through files that are small enough to be transmitted over email. Data can be extracted to Pi (a Cosworth proprietary file), MATLAB or ASCII files. Development of new widgets by the users can be done using Visual Basic or Visual C++. Mathematical operations combining several sensors can be done, as the obtained result can be used like any other sensor. Data filters can be used to modify the data. A typical application of this is to remove noisy data. The storage of the acquired data is done using the Pi propriatery files.

2.1.2 McLaren

2.1.2.A Hardware

McLaren provides data loggers with 2GB of on-board flash memory. The recorded data can be later transferred through an ethernet connection. It is possible for the user to develop programs to be run in these units. Hardware accelerated front-end signal processing functions, such as filtering and down-sampling, are available [17]. The data logger can be seen in 2.5(a). The available sensors are:

- Accelerometers to measure the forces that are being applied to the car.
- Gyroscopes to get the orientation of the car
- Humidity sensors used to analyse the weather conditions.
- Oxygen sensors. Used to analyse the oxygen levels that enter and exit the engine.
- Linear displacement sensors to measure the use of the suspension.
- Rotational displacement sensors to measure the steering angle.
- Aero pressure sensors. This can be used to measure the downforce generated by the car.
- Fluid pressure sensors to monitor the pressure of the fuel and oil pressure.
- Tire pressure sensors to measure the pressure inside the tires.
- Speed sensors to analyse the speed of the car, using the rotational speed of the wheels.
- Infrared temperature sensors to measure the temperature of the tires.
- Air temperature sensors to measure temperature of the air inside the tyres.
- Lap sensors to know lap times.

It is also possible to order specially designed sensors.

McLaren has two main solutions for real time transmition of data. The top of the range is the CBT-610. Which is a L-Band radio transmitter, capable of data rates up to 2 Mbps. It has a 48MB internal memory to support re-transmission. It is possible to achieve data handshaking, for improved lap coverage, by using a small companion UHF modem. The other solution is the CBX-460 that relies on a 3G modem that can achieve bandwiths up to 3.2Mbps. The 3G network can be public or private. It can also be used to achieve the handshaking capality of the previous model. The CBT-610 can be seen in figure 2.5(b).

(a) Data logger. **(b)** Radio unit.

Figure 2.5: McLaren hardware.

2.1.2.B Software

McLaren Atlas software allows the display of data in: waveforms, circuit, bar charts, numbers, scatter plots, histograms and summary of lap or section.

McLaren software provides several types of graphs to display the data. This software is ActiveX compliant and makes most of its commands available, making it highly customizable [15]. A screenshot of this application can be seen in 2.6.

To further help the analyses of data it is possible to combine the data of two or more sensors in a mathematical formula and the result can be displayed in plots. Data can be imported and exported from MATLAB files. It is also possible to access data in other formats using a provided DLL. The software can be used for real time and offline analyses.

The storage of data can be done using McLaren SQL Race, which is an API for Microsoft SQL Server 2008. This enables several users to access the data at the same time. SQL Server Service Broker enables the use of several databases and still keep them synchronized with each other. This way it is possible to have database at the race track and another in the factory. The use of SQL Server makes it possible to extract and analyse data using querys. Furthermore, programs can be developed to use the acquired data in other ways, for example simulation programs.

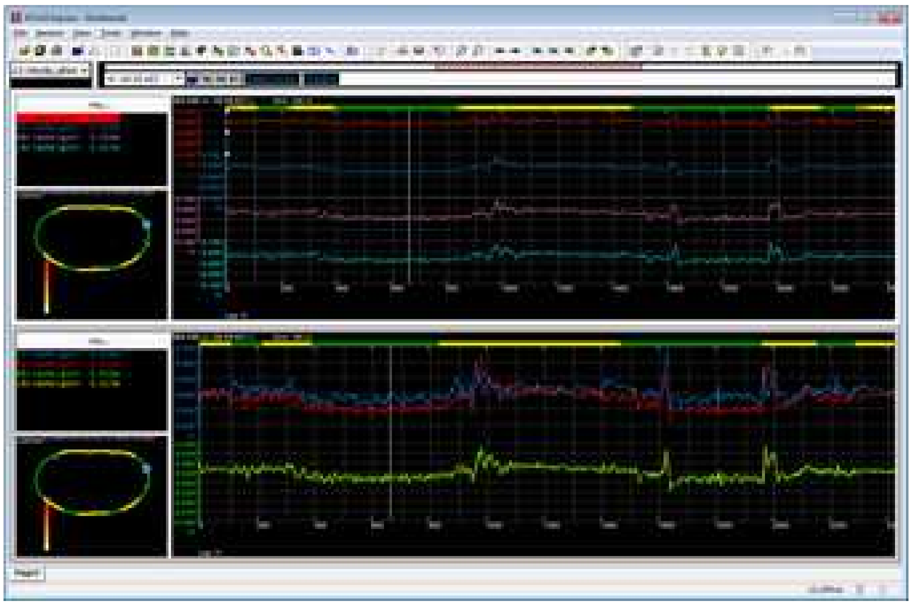


Figure 2.6: Screen shot of McLaren software.

2.2 Academic prototype

2.2.1 Overview

Just like the industrial prototypes, the academic prototype that was developed at IST is composed of a base station and a mobile station. The mobile station is the station that will remain in the car to receive, log and transmit data received from the CAN-bus. The base station will remain in the pitstop and will receive the data transmitted by the mobile station. It will also present the acquired data to the users in a comprehensible way and log it for later analysis.

2.2.2 Acquired signals

This prototype can acquire several several types of sensors, such as:

- Tire temperatures. In order to calibrate several parameters such as caster.
- Suspension displacement. To show how the suspension is working to better adjust the dampers.
- User input brake, throttle and direction.

The detailed list of signals that can be acquired with this prototype can be seen in table 2.1.

2.2.3 Mobile station

The mobile station of the previous academic prototype is an embedded system supported on a Microchip PIC18F4685 8 bit microcontroller. This system provides a CAN-bus and an USB interface.

Table 2.1: Set of acquired signals

ID	Description	Sample frequency[Hz]
0	Synchronization	-
32	Speed DD	10
33	Speed DE	10
34	Speed TD	10
35	Speed TE	10
36	LVDT DD	20
37	LVDT DE	20
38	LVDT TD	20
39	LVDT TE	20
41	RPM	10
42	TPS	20
43	ECT	1
44	Neutral	-
45	Steering angle	20
51	IR DDi	5
52	IR DDe	5
53	IR DEi	5
54	IR DEe	5
55	IR TDi	5
56	IR TDe	5
57	IR TEi	5
58	IR TEe	5
61	Pressure DD / Temperature DD	1
62	Pressure DE / Temperature DE	1
63	Pressure TD / Temperature TD	1
64	Pressure TE / Temperature TE	1
71	Acceleration X	20
72	Acceleration Y	20
73	Acceleration Z	20
74	Orientation X	20
75	Orientation Y	20
76	Orientation Z	20
81	Air humidity	1
82	Air Temperature	1
83	Atmospheric Pressure	1

The Controller Area Network (CAN) is a serial communications protocol, developed by Bosch, that is widely used in automotive industry. In this protocol, every node is able to send and receive messages, but not simultaneously. All nodes are free to transmit if the bus is free. If two or more nodes start to transmit at the same time, the message with the most dominant ID will prevail and overwrite the other messages, so that in the end all nodes will receive the most dominant message. The CAN-bus is used to receive the data from the car sensors, which is subsequently stored in an USB flash drive, and uploaded to the base station.

The communication means to the base station was supported on a wireless connection implemented using a ZigBee module. Table 2.2 depicts the information about the used ZigBee link. The implemented transmission protocol does not offer a reliable transmission. As a consequence, to acquire all the data of one given session there is the need to download it from the flash memory. To house the mobile station, a commercial box was modified so that it could fit the CAN-bus transceiver, the USB memory drive, the antenna and the power switch. Three LEDs were added to give information about the state of the mobile station. The presented informations are power on/off, logging on/off and USB activity on/off. The mobile station can be seen in figure 2.7.

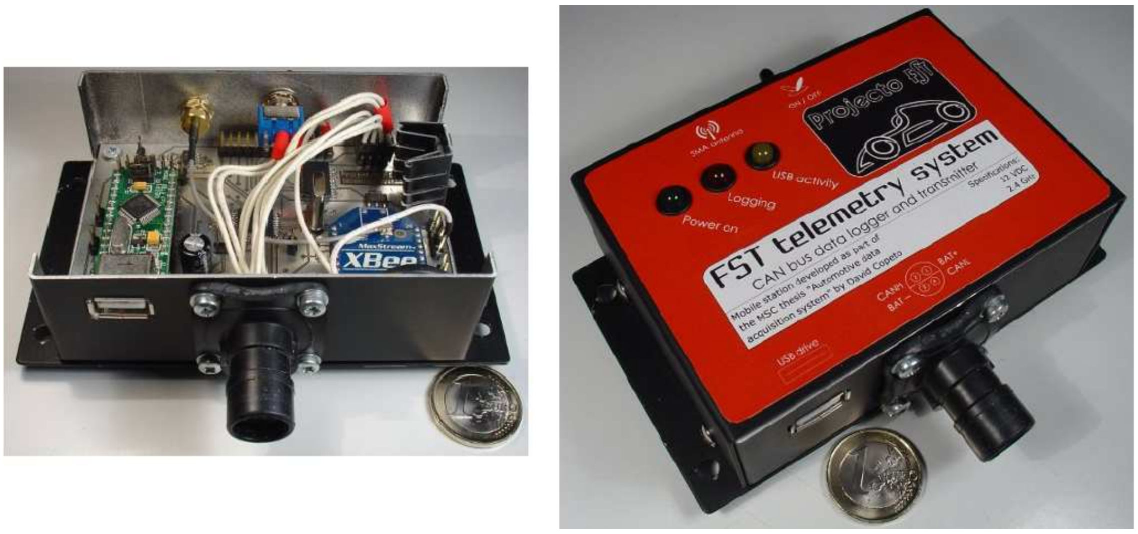

XBee
MaxStream
Projecto FST
SMA antenna
ON / OFF
USB activity
Logging
Power on
FST telemetry system
CAN bus data logger and transmitter
Mobile station developed as part of the MSc thesis "Automotive data acquisition system" by David Copeto
Specifications: 12 VDC 2.4 GHz
CANH BAT+ CANL BAT−
USB drive

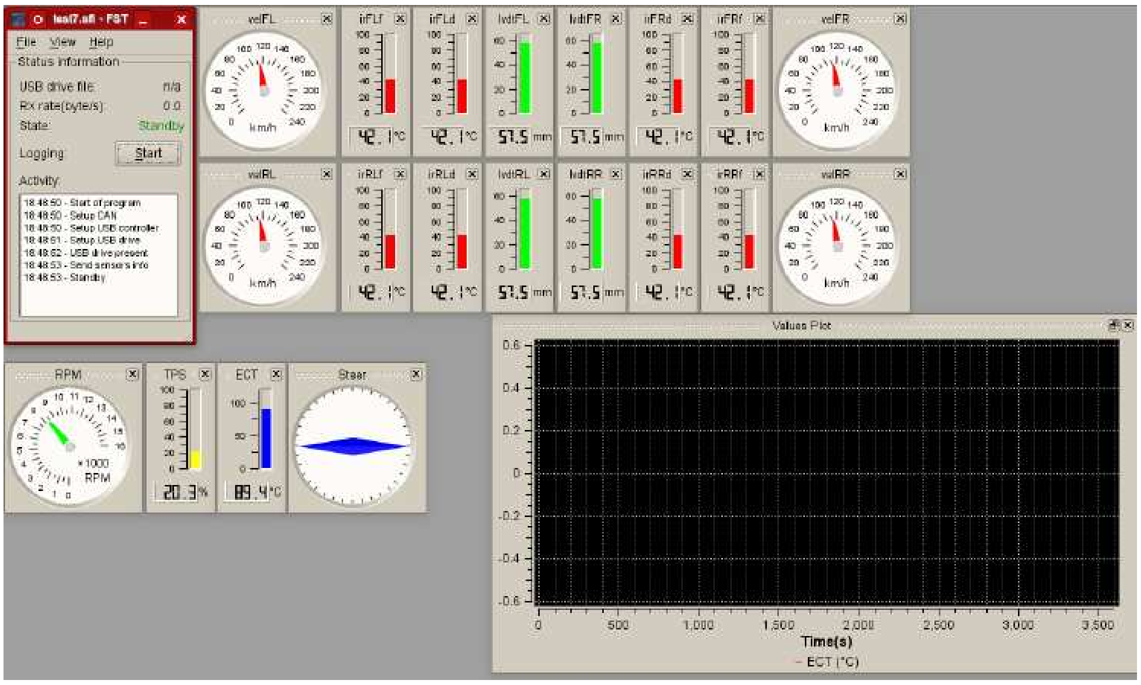
Figure 2.7: Base station

Table 2.2: Characteristics of the ZigBee communication line

Indoor/Urban Range	Up to 100 m
Outdoor line-of-sight Range	Up to 1500 m
Transmit Power Output (software selectable)	60 mW (18 dBm) conducted
RF Data Rate	250.000 bps
Serial Interface Data Rate (software selectable)	1200-115200 bps
Receiver Sensitivity	-100 dBm (1% packet error rate)
Supply Voltage	2.8-3.4 V
Transmit Current (typical)	215 mA (@ 3.3 V and 18 dBm output power)
Idle / Receive Current (typical)	55 mA (@ 3.3 V)
Power-down Current	$<$ 10 A
Operating Frequency	ISM 2.4 GHz band

2.2.4 Base station

The base station is the platform that will present the acquired data to the user. This data may come from the wireless communication line, from a flash drive or from a data file corresponding to previous sessions.The stored data is accommodated in two types of files, with extensions .sfl and .txt. The .sfl is used to store information about the logging session, such as date and time, number of sensors, among others. The .txt is used to store the details and data received for each sensor, where each sensor has its own .txt file. When the user wants to receive data from the wireless connection, some information has to be introduced. This information is concerned with the name of the sensor, its type and CAN-bus ID. Moreover, the user should also specify which sensors will have their information displayed in the plot (this can also be loaded from a previous session). The developed interface offers the option of showing the information in several widgets, as can be seen in figure 2.8. Thus allowing for a fast analysis of the state of the car. It also shows a plot for an easier analysis of the evolution and interaction of the various parts of the car. This plot also allows the user to select from which sensors the data will be displayed, to zoom, in interesting regions axis dragging, display of coordinate values and to save the current plot as an image file. The user is also able to start a fresh session with the same configuration of a previous one. In case there is the need to change the topology of the CAN-bus, such changes can be introduced during a live session.


Status information

USB drive file: n/a
Rx rate(byte/s): 0.0
State: Standby
Logging: Start

Activity:

```
18:48:50 - Start of program
18:48:50 - Setup CAN
18:48:50 - Setup USB controller
18:48:51 - Setup USB drive
18:48:52 - USB drive present
18:48:53 - Send sensors info
18:48:53 - Standby
```

Figure 2.8: Example of some base station widgets

In case the user prefers to run an offline session, he has to select the file where the desired session was logged and the logged data will be available in the plot. In this case the user has to select the USB drive, and then put the type of sensor that each CANbus ID corresponds to.

3 Mobile Station

Contents

3.1 Mobile Station architecture

The mobile station incorporates several components. In order to have an over of the system its architecture can be seen in figure 3.1.

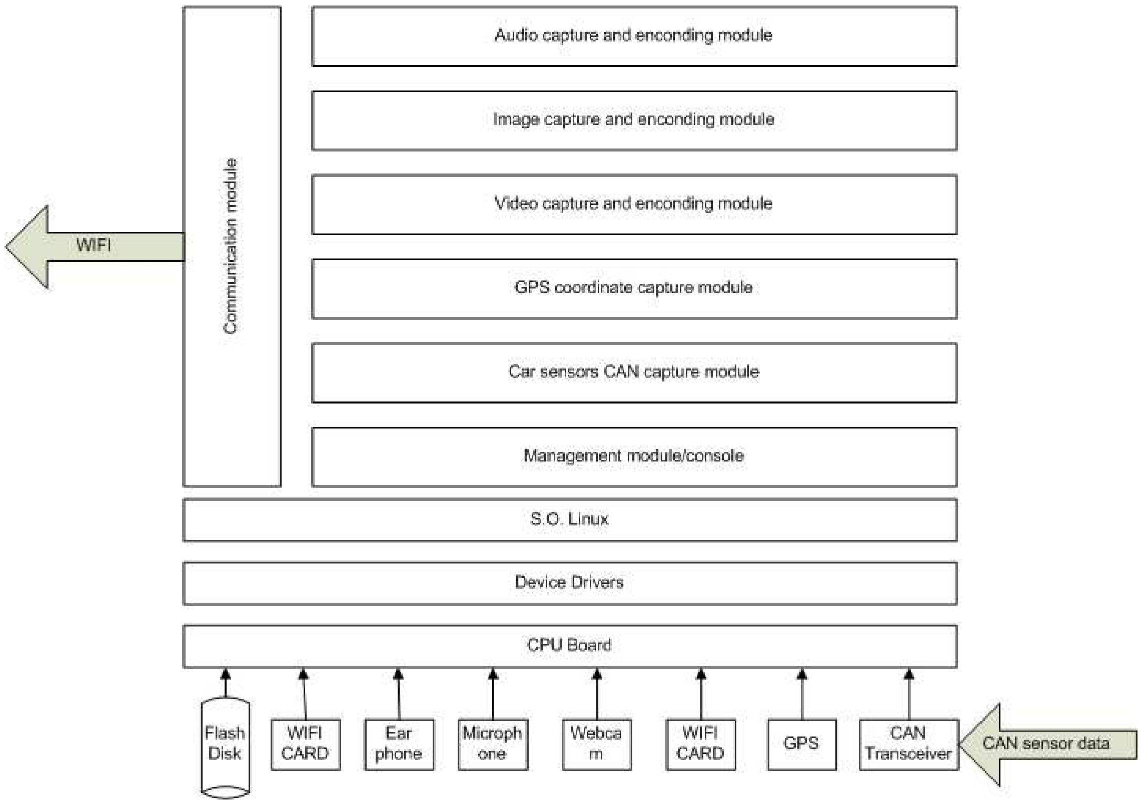

Audio capture and enconding module
Image capture and enconding module
Video capture and enconding module
GPS coordinate capture module
Car sensors CAN capture module
Management module/console
Communication module
WIFI
S.O. Linux
Device Drivers
CPU Board
Flash Disk
WIFI CARD
Ear phone
Microph one
Webca m
WIFI CARD
GPS
CAN Transceiver
CAN sensor data

Figure 3.1: Overview of the mobile station architecture

- The audio module is responsible for acquiring and encoding sound that is sent to the base station.
- The image capture module periodicly takes pictures using the webcam, encodes them and sends to the base station.
- The video capture module acquires video trough the webcam, encodes it and sends it to the base station.
- The GPS coordinate capture module acquires the GPS position using the GPS sensor and sends that data to the base station.
- The CAN capture module receives the CAN data through the CAN transceiver and then sends it to the base station.
- The management module enables the access to the mobile station by the base station, either by ssh or by serial console.

- The operating system together with the device drivers deliver abstraction levels that ease the software development.
- The CPU board is a small and low power board that makes available several interfaces, such as USB, earphone and microphone jacks, IDE, SATA, among others.
- The Communication module works together with the WIFI card in to reliably comunicate with the base station. This module deals with all the communication between the stations, except for the management console.
- The flash disk in this case takes the function of hard drive.

3.2 Requirements

To cope with the set of requisites imposed by the Formula Student Championship several hardware and software requirements have to be meet, as described in the following subsections.

3.2.1 Hardware requirements

For the implementation of the improved telemetry prototype we need the following hardware characteristics have to be assured:

- Low cost: since the formula student competition has a cost competition, the low cost of the solutions is very important.
- Compact in size: the mobile station needs to be small, in order to minimize the impact of its inclusion in the design of the car.
- Low power consumption: this hardware has to be powered by the battery at the car; as a consequence, low power consumption is a very relevant aspect.
- More flexible: the previous prototype [4] allowed a very small flexibility. This was caused by two main factors: lack of computational power and very few interfaces. The chosen platform cannot have these limitations.

The selected hardware that meets these needs was an ultra low-power C7 general purpose processor from VIA. This processor will be integrated in a VIA EPIA-P700 compact board with 1GB of DDR2 RAM memory. The most important features of this board are:

- It is a compact board, with measures of 10cm x 7.2cm. This ensures that the board will easily fit in the car, avoiding the need to redesign the car.
- Has a low power consumption. This feature is very important since the power supply of this hardware will is a battery installed on the car.

- The C7 general purpose processor works at 1GHz and is able to interpret x86 code. This feature, alongside with the 1GB of RAM memory, gives the system the capability to run most operating systems, which brings many advantages. One advantage of using an operating system is that it will deal with the access to the peripherals. More benefits of using an operating system and its choice will be discussed further in this document. Another advantage offered by this processor is the capability to run more powerful software, since more processing power is available.

- The board provides many interfaces, such as IDE and S-ATA II ports, Gigabit Ethernet, four USB ports, serial port and CRT display. This allows for many choices in terms of the peripherals that may connect to it, specially in terms of storage and communication.

This approach also leads to a system whose parts are highly updateable and easily replaceable. This is achieved since spare and similar parts are easily found in the market.

The versatility of this board offers many design options. One of them is concerned with the storage medium that will accommodate the operating system. The considered options were USB flash drive, a convencional hard drive or a SSD card. The hard drives were discarded because they cannot cope with the vibrations that will be generate by the car. The SSD cards are still to expensive to be an option. As a consequence, it was decided to use a 4GB USB flash drive. Such devices are cheap and have enough storage for what is needed. The board can be seen in figure 3.2.

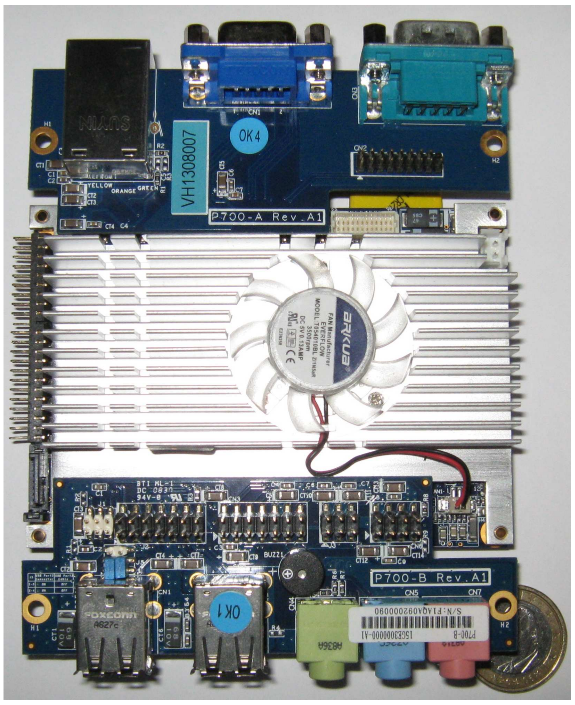


Figure 3.2: Mobile station processing board.

3.2.2 Operating system requirements

As it was said before, the use of an operating system brings many advantages. One of such advantages comes from the fact that it brings an abstraction layer to the hardware. This greatly eases the development of the software, since there is no need to directly deal with registers, giving the programmer higher level instructions. This also makes the software more hardware independent, since the differences in the access to the different hardware devices are hidden by the operating system. The operating system also provides several components of software, avoiding the need to develop them. For example, free remote access programs are available. Without them, a specific solution to access the mobile station while the car is on track would have to be developed.

With these advantages in mind the selection of the operating system had to into consideration the following aspects:

- Cheap: Just like the hardware, this is a critical point because of the cost competition and the available budget.
- Light: due to the power consumption limitations, the operating system must use a low computational power.
- Small: because the storage will be confined to a 4GB USB flash drive, the operating system cannot take much space.
- Adequate to be run in a read-only flash memory. Since the USB flash drives have a limit on the number of writes cycles done during its life, the operating system has to do very few writes to the physical storage.
- Include the device drivers that are required to access the several peripherals. In order to ease the development of this and other projects, it should come with the needed device drivers, so that there is no need to search and install them. Which some times reveals to be a big problem, since some times the vendors do not give drivers for all operating systems. Making the only solution to find it on the internet, or try and develop the drivers.

In terms of cost, the selection was quite simple. Having in consideration Microsoft Windows, Linux and MacOS, the option was simply to use a Linux distribution since it is the only option that is free. From here, a small Linux distribution that also provided the other required characteristics had to be found. There were many options, such as Knoppix (Debian based) and Flash Linux (Gentoo based). Since both meet the needs, it was just a question of preference and it went to Knoppix, which is more friendly and there was already some experience in a similar instalation.

On Linux Operating system the most common way to define which services are to be started runlevels are used. There are 5 runlevels, they are:

- Runlevel 1. This is used only to repair and maintain the system.
- Runlevel 2. In this level most of the services are started, still networking is not started.
- Runlevel 3. This runlevel all services are started except the Graphical User Interface. So a command line is how the user interacts with the system. This runlevel is mostly used by servers.
- Runlevel 4. Is mostly used as a costum runlevel. By default it will start a few more services than runlevel 3.
- Runlevel 5. This is the runlevel that most of the workstations are started. It starts the printing services, Graphical User Interface and 3rd party services.

So the operating system was configured to start in runlevel 3.

The other requisite was to start the program for the acquisition and transmition of signals, a boot time. An entry was added to the rc.local file to automatically start this program. The configuration of these options can be seen in Appendix A.

3.2.3 Programming language requirements

A programming language with these two caracteristics:

1. Leed to fast running programs. In order to save battery and computational power on the mobile station.
2. Compatible with Linux. Since the operating system is Linux.

With this in mind the choices were C or C++. The option went to C++ because it is object oriented making the software easier to maintain.

3.3 Data acquisition

The mobile station will acquire three different types of data. One is the data acquired by the several sensors and transmitted using the CAN-bus. The other is audio and video data that will be acquired through a microphone and an USB webcam. The third is the GPS data that will be provided by an USB GPS sensor.

3.3.1 Sensors data

3.3.1.A Acquired signals

The set of acquired signals in this prototype can be seen appendix A. The several types of sensors available on board are:

- Steering position: the steering position is a good indicator of the quality of the setup. If a car has a bad setup, the driver will struggle and one of the ways this is visible is by sudden changes in the steering position.
- Suspension displacement: the suspension displacement shows the travel that is being used; if one of them shows a very different pattern, it indicates a possible problem.
- Tire pressure: the tire pressure is a very important aspect to be analyzed, since it will affect several things such as grip of the tire, energy consumption and how the car deals with irregular parts of the track.
- Tire Temperature. The car has eight tire temperature sensors, where each tire is equipped with three sensors. Tire temperature sensors are used to check that the tires are within the desired window of temperature. The reason to have three sensors in each tire is to acquire information about the quality of the camber setup and tire pressure.
- Internal tire temperature: the temperature inside the tire will change the pressure of the tire. As such, it is important to monitor this value to see how it changes, allowing to better adjust the tire pressure.
- Throttle and brake position: the analysis of the throttle and brake position sensors. Serves to analyse how the car reacts to these controls.
- Wheel speed sensors: there are four wheel speed sensors, one for each wheel. It is based on the data provided by these sensors that the speed of the car is evaluated.
- Acceleration in the different axis: this information is very important it allows to observe the limits of the car when it turns, accelerates or brakes.
- Engine RPM. The engine RPM is a very important aspect, since the maximum performance can only be achieved in a specific RPM interval.
- Engine temperature: it is important to have a close look at this value because if the temperature gets to high serious problems can be made to the engine.
- Engine voltage: monitoring this parameter is very important in order to evaluate the battery autonomy.
- Battery voltage: by monitoring this value, it is possible to evaluate how much energy the battery is transferring.
- Battery current: this value shows how much charge the battery still has.

3.3.1.B CAN protocol

To acquire the signals, the mobile station is connected to a CAN-bus. Next, it is presented a brief description of the CAN protocol. The maximum communication speed of CAN, in version 2.0B, is 1Mbit/sec. The CAN uses a CSMA/CD protocol. CSMA stands for Carrier Sense Multiple Access. This means that every node in the network must monitor the network to find a period of no activity. Only when this period is found are they able to send the message. Also, in this period every node has an equal opportunity to send the message. The CD stands for Collision Detection, which means that if two nodes transmit at the same time, appropriate measures will be taken in order to assure that only one transmit. As such, the CAN protocol uses a non destructive bitwise arbitration method. This allows that even when collisions are detected, the message with more priority will remain intact after the arbitration. The CAN defines that the message with the lower value in the Message Identifier has higher priority.
The CAN protocol is based on messages not addresses. This means that all nodes will receive the message and acknowledge it if the message was properly received. It is up to each receiver to decide if it wants to discard it or to keep it to be processed. In a CAN based network, it is also possible for a node to request information from other nodes. This is called a Remote Transmit Request (RTR).
The CAN protocol has four types of messages (or frames), which are: Data Frame, Remote Frame, Error Frame and Overload Frame. The last two are for handling errors and will not be discussed in this document. The most common is the Data Frame, used to transmit data from a node to the network. The second one is the Remote Frame, which is basically a Data Frame with the RTR bit set to mean that it is a Remote Transmit Request. The Data Frames are composed of Arbitration Fields, Control Fields, Data Fields, CRC Fields, a 2-bit Acknowledge field and an End Frame, as it can be seen in figure 3.3.

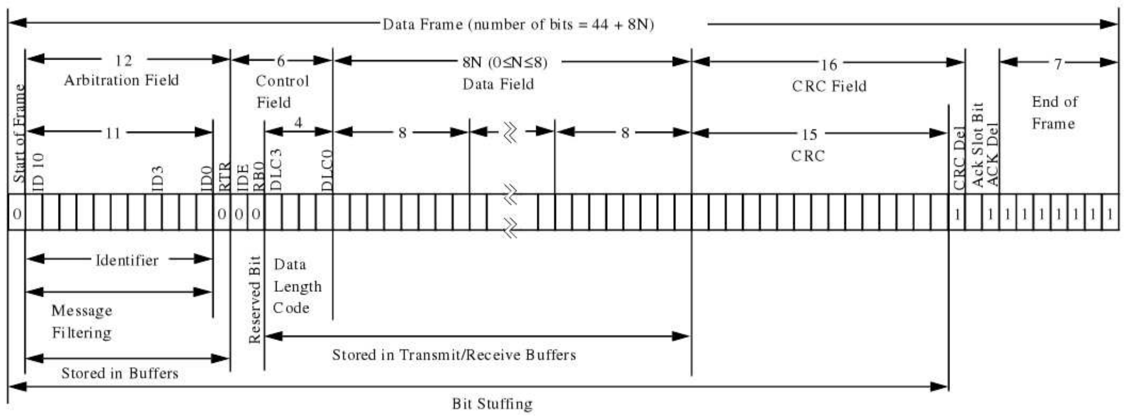


Figure 3.3: CAN data Frames structure

3.3.1.C Connection to the CAN-bus

In order to connect to the CAN-bus an adaptor needed to be found since the board does not have a CAN interface. In the beginning it was considered the use of a RS232 to CAN-bus transceiver since this would be the more direct solution. Since there was no RS232 to CAN-bus transceiver available at the lab, it was used an USB to CAN transceiver that was built in the scope of a previous thesis [7]. This transceiver was created having in mind the need to easily test and interact with the CAN-bus using a laptop. In order to achieve this a protocol was developed for the communication of the transceiver with the laptop. The several types of messages that were created can be seen in table 3.1.

Table 3.1: Defined messages to communicate with the transceiver.

Name	Value	Description
READ_VERSION	0x00	Message from the laptop to the transceiver asking for the firmware version.
PC2CAN_MSG	0x01	Transmission of a message to the CAN-bus by the laptop.
CAN2PC_MSG	0x03	Transmission of a message to the laptop by the CAN-bus.
READ_STATUS	0x05	Ask the reading status of the CAN controller by the laptop.
SEND_ERROR	0x06	Message from the USB-CAN interface to the laptop when an error occurs in the message transmission to the CAN-bus.
RESET	0x07	Send an order to reset the CAN controller by the laptop.
SETUP_BITRATE	0x08	Message sent by the laptop to the transceiver with the values to reconfigure the CAN controller bitrate.
UNKNOWN	0xFF	Answer from the one of parts when a message is received with an unknown identifier(possible when used different versions).

This transceiver is illustrated 3.4.

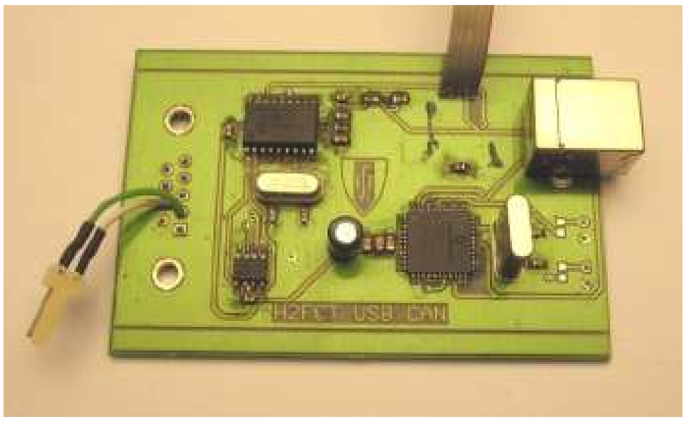


Figure 3.4: USB-CAN transceiver

This transceiver does not signals the arrival of new data. Consequently, a dedicated thread is used to read data from this interface.

Each sensor in the car has a timer in order to send a timestamp together with the data acquired. In order to have these timers sincronized, they have to be reset. To do so, a message with an id 0 is sent. This is done at the startup of the mobile station and upon the receiving of a message issuing the start or stop the transmition of data to the base station.

The structure of the message received from the CAN-bus through the transceiver is a CAN2PC_MSG. The message is composed of the identifier of this type of message followed by the CAN message. The CAN message sent by the sensors is be composed of 16 bits for the timestamp and 16 bits for the acquired data. Only the sensor for the inside temperature sends a different message with 8 bits for the temperature, 8 bits for the pressure and 16 bits for the timestamp. The time-unit that is used in the timestamp is hundredths of a second, so the timestamp will overflow in 10 minutes and 55 seconds. Because a training session or a competition easily exceed this time, a workaround had to be done. As a consequence at the mobile station the timestamp is increased to 32 bits. This gives a timestamp that can measure up to 11 hours and 55 minutes, without overflow. However since the CAN protocol does not guarantee ordered delivery, it is possible to receive messages with a timestamp that is smaller than the latest received message. Consequently the increase of the timestamp resolution to 32 bits implies some calculations. The implemented solution is to check the remainder of the last timestamp of the sensor and check if it is smaller than the new timestamp. In this case, the new timestamp for the sensor is the result of the integer division by biggest unsigned int plus the received timestamp. If the previous condition is not satisfied then the new timestamp for the sensor is the result of the integer division plus the biggest unsigned int plus the received timestamp.

3.3.2 Audio and Video acquisition

Nowadays audio and video links are a common features in competition cars. Audio is very important since it is the fastest way of communication beetween the driver and his team. Many information can be exchange using the audio channel, such as:

- The team can give instructions regarding the strategy;
- Tell the pilot about his position in the race;
- The team can warn the pilot of possible events, such as the possibility of rain or the safety car being deployed;
- Information about how the opposition is behaving can be given and consequent strategies to deal with that;
- Warn of problems with the car: this information can come from the pilot or from the team that is watching the telemetry data. After this, solutions to the problems can be discussed.

The video acquisition is not as important to the outcome of the race, since few information is extracted from it. But it still plays an important role in the sport, since it is thanks to it that more people becomes interested with the sport, considering that it gives a better perspective of the driver's point of view. The interest leads to more sponsorship, which is vital in the automotive sports, since this kind of sports need a big budget.

With these advantages in mind an audio and video acquisition system was developed to install in the car [3].The audio acquisition system is made through a microphone that is connected to the board. The video acquisition is composed by an USB webcam that is connected to the board as well. This webcam uses the video4linux driver, provided by the Linux distribution.

Here is a good example of the importance of the board, since using a micro-controller to achieve this would make the solution much more expensive and complex.

The audio and video acquisition program has several parameters that can be seen in table 3.2.

Table 3.2: Audio and video acquisition options

Option	**Parameter description**	
Images		
Capture images(snapshots) in JPEG format	Frame rate	Set frame rate in FPS(How many images per second)
	Resolution	Set frame height in pixels
		Set frame width in pixel
Video stream		
Capture a continuous sequence of video	Frame rate	Set frame rate. Range is [0.2-30]FPS. Optimal value is 14 FPS
	Resolution	Set frame height in pixels
		Set frame width in pixel
Audio stream		
Capture a continuous sequence of audio	Sampling rate	Set sampling rate. Range is [4000-4800]Hz

The communication with the program that makes the acquisition and encoding of the audio and video signals is made using an Unix socket. During the setup stage the values corresponding to the several parameters are sent through the socket. From that point, the encoded data will start arriving. If while the program is running it is necessary to change these parameters, it is simply required the new configuration.

3.3.3 GPS integration

The use of a GPS terminal in a telemetry system allows the preception of the location of the car in the track. This way the users do not need to try and guess where the car is, having only the data from the car sensors as base. With this aspect in mind , it was decided to integrate a GPS system to the solution.This was achieved by adding a thread to the mobile station program that deals with the GPS data acquisition. In order to have the timestamps sincronized with those that come from the CAN-bus, a variable retains the last timestamp that was read from the CAN-bus sensors, and this is used by the GPS as the considered timestamp. This way, both systems stay sincronized. By using the GPS UTC time would be possible use the real time, but since this signal is acquired with a sampling of 1 Hz. Since this resolution is much lower than the resolution used by many other sensors, it is impossible to use it.

3.4 Communication

All the acquired data is immediately sent to the base station to be analyzed while the car is on track. This is important since it greatly improves the development time in the testing sessions, with the team being able to call the driver to fine tune the car using the information that was acquired. In competition this is also very important, since the team can give instruction to the driver of possible problems in the car and ways to manage these problems. The communication infrastructure and protocol are discussed in detail in chapter 5.

3.5 Data recording in the USB drive

In case the previously mentioned communication infrastructure fails, it is important to still be able to retrieve the acquired data. The solution that was found to solve this problem is the use of a flash pen drive to store the acquired data. Since Knoppix does not take the whole capacity(4GB) of the pen it permits the selection of an amount of space that will be dedicated to data storage and used as a regular pen drive. This storage space was used to store the data acquired when the mobile station is on. With this approach, it was avoided the need for a another USB pen drive. Furthermore, the access to this stored data is easy, since it is only needed to connect the pen to the computer. In order to identify the session that the file refers to, the name of the file will be the date that the mobile station was turned on. In case the base station sends a signal to start a new session, the currently open file is closed and a new file is created, using the session identification as name, and the transmition of acquired signals starts. When the base station requests to stop sending data about the session, then the currently open file is closed and another one is opened with the creation time as name, and the transmition of data is stopped. Since some times the energy of the car is suddenly cut off, a battery was added to support the board while it shuts down. And a sensor was created that detects that the energy is coming only from the mentioned battery, uppon this it starts sending a signal through the CAN. Upon receiving this message the base station starts to shut down. While this operation takes place, the mobile station is powered by the battery. This way, all the data acquired by the car is acquired and easily accessible. This solutions makes the mobile station work as a black box and totally independent from the base station.

3.6 Administration channel

The establishement of an easy and reliable way to connect with the mobile station has vital importance, because the car has no keyboard or screen. Since problems can arrive and they have to be quickly solved, most of the time that requires having access to a console on the mobile station. This way, the problem can be identified and solved.

3.6.1 Wireless console

The easiest and quickest way to have access to the mobile station is using a ssh client. This client has access to the mobile station using the wireless communication that is described in chapter 5. This way, even with the car on track problems, can be easily solved. To have this solution working, a ssh server was installed in the mobile station. After this the ssh server was set to begin at startup by altering the /etc/rc.local file. More details can be seen in Appendix B. Through this solution more track time can be achieved since the car can stay on track while problems are being solved.

3.6.2 Wired console

The wireless access is a good solution. However, it cannot solve problems regarding the wireless channel. As a consequence a solution that does not depend on the WiFi infrastructure had to be found. Altough it needs access to the car, the solution found is the use of a serial line to access a console by using the serial port. Since most computers nowadays do not have a serial port, a transceiver USB - RS232 was used to connect the computer to the mobile station board. For the serial console, two files in the mobile station had to be edited.
Some configurations had to be done so that the serial console was configured. One of them is concerned with the boot loader. So the /mnt-system/boot/syslinux/syslinux.cfg was edited to make possible to interactively control the boot loader from the serial console. The kernel was also configured by changing the /mnt-system/boot/syslinux/syslinux.cfg file, directing the console output not only to the screen but also to the serial port. A getty client was necessary, since it monitors serial lines, waiting for a connection. It then configures the serial link, sends the contents of /etc/issue, and asks the connecting user for his login name and password. If the user does nothing, getty or login hang up and getty goes back to waiting. The /etc/inittab file was also edited since it is here that the serial line speed, the serial port and the type of terminal emulation are specified.
This is a very reliable and easy to use solution, since it is just needed to connect the cable and start the minicom in the computer to get access to the console.

4
Base Station

Contents

4.1 Requirements

The requirements for the base station are very basic, since there is not any size and power consumption limitations. In terms of hardware a general purpose computer with, a monitor, a 60GBhard drive and a wireless interface or an USB interface is enough. For the operating systems, a linux distribution system was chosen since it was the only free option. Otherwise other operating systems would have been considered.

4.2 Database

The data that is received from the mobile station is stored in a database management system. This approach provides independence to the application programs, allowing future works to use the same common database and easily change and access the acquired data. The access to the stored information is done by a query language. This makes easier the retrieval of the data.

4.2.1 Database requirements

Among the several database management systems available in the market, the choice was made based on:

- Cost, leading to an open source solution, since these tend to be good solutions and free.
- Fast, since it will have to deal with a significant amount of information arriving at a considerable rate.
- Easy to move from one computer to another. Since it will not always be the same person to go to the tests or the competions, so it is likely that different computers will be used to store the data. In order to keep the data integrity it is good to always use the same database.
- SQL features. It is important that most of the SQL features are implemented in the database management system.
- Administration tools. Good administration tools greatly ease the maintenance and creation of the database.

Having in mind these needs, the considered options were MySQL, PostgreSQL and SQLite.

4.2.1.A MySQL

MySQL is a relational database management system that runs in most operating systems. Most languages also include libraries to access MySQL databases, providing more choices in terms of the language to use, guaranteeing easy access to the data. Guided User Interfaces such as MySQL Administrator and the MySQL Query Browser are very useful in the construction of the database, allowing a more intuitive use of MySQL. It also provides the normal database management system features. However what distinguishes it from the other database systems is the possibility to choose among several storage engines according to the needs [19].

4.2.1.B PostgreSQL

PostgreSQL is an object-relational database system that runs in most operating systems. It is easy to integrate since most languages have native programming interfaces to connect with. It is fully standard compliant. As MySQL, regular database management features are included. The distinction comes from the adopted GiST (Generalized Search Tree) that brings together a wide range of different sorting and searching algorithms including B-tree, B+-tree, R-tree, partial sum trees, ranked B+-trees and many others [10].

4.2.1.C SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. Its main feature is it speed, since it reads and writes directly to ordinary disk files. As such, a complete SQL database with multiple tables, indices, triggers, and views, are contained in a single disk file. The database file format is cross-platform, so it is possible to freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. However, this comes at the price of some SQL features not being available. These features are: right outer join, full outer join, complete alter table support, foreign key constraints and writing to views [11].

Both PostgreSQL and MySQL provide the features that are required, since both are open-source, have powerful administration tools and have all SQL features. The only drawback may be that they are not capable of offering the needed insertion speed. At this respect MySQL is ahead of PostgreSQL and in some cases even of SQLite. When comparing MySQL and SQLite, SQLite lacks the full SQL features and MySQL administration tools are better. On the other hand MySQL writes in several files, while SQLite in only one file. It is expected that with the increase in the number of sensors and subsequent increase in the number of tables, MySQL will become too slow to deal with the increasing volume of insertions. The fact of storing data in a single file also eases the backup and the transfer of the database. As such SQLite was chosen as the database management system that better meets the needs of this project.

4.2.2 Database structure

The database stores the CAN-bus configuration, information about each session and data from the sensors. It is important to note that SQLite always creates a numeric primary key for each table. Having that in mind, it was adopted the primary key that is generated. So no primary key will be given representing the table since SQLite will ensure that.

To store the CAN-bus configuration, a specific table was created. The table structure is:
`CONF_CAN(nome,ID,tipo)`

- name - stores the name that was given to the table and the sensor.
- ID -the identifier of that sensor in the CAN-bus.
- type - Indicates the type of that sensor. This is important since many sensors send information that is not treated and some operations have to be made to translate it into values that make sense for the user.

Each session has its data stored in a table with this structure:
`session(name, dateTime, place, localTemp, weatherCond, driver, toe_fr, camber_fr, caster_fr, KPI_fr, weight_fr, SpringRate_fr, LSC_fr, HSC_fr, LSR_fr, HSR_fr, tyre_fr, toe_fl, camber_fl, caster_fl, KPI_fl, weight_fl, springRate_fl, LSC_fl, HSC_fl, LSR_fl, HSR_fl, tyre_fl, toe_br, camber_br, caster_br, KPI_br, weight_br, springRate_br, LSC_br, HSC_br, LSR_br, HSR_br, tyre_br, toe_bl, camber_bl, caster_bl, KPI_bl, weight_bl, springRate_bl, LSC_bl, HSC_bl, LSR_bl, HSR_bl, tyre_bl)`
The underline fr corresponds to front right, the underline fl to front left, the underline br to back right and the underline bl to back left.

- Name - Registers the name given to the session.
- dateTime - The time that the session started.
- place - The place where session is being held.
- localTemp - The air temperature for the session.
- weatherCond - The weather conditions for the session.
- driver - The driver of the car.
- toe - The toe used for the session. Toe is the angle of the wheels when viewed from the top. Toe settings affect three major areas of performance: tire wear, straight-line stability and corner entry handling characteristics.
- camber - The camber used during the session. Camber is the angle of the wheel relative to vertical, as viewed from the front or the rear of the car. The cornering force that a tire can develop is highly dependent on its angle relative to the road surface, and so wheel camber has a major effect on the road holding of a car.

- caster - The caster the car was setup for the session. Caster is the angle to which the steering pivot axis is tilted forward or rearward from vertical, as viewed from the side. Caster affects the straight-line stability and the steering effort.
- KPI - The KPI the car used during this session. KPI is the angle between vertical and the imaginary line joining the center of the upper ball and lower ball joints.Too much KPI will result in high steering effort and poor tire-contact patch during cornering.
- weight - The weight that was place in a corresponding spot during the session.
- SpringRate - The spring rate used during the session Spring Rate. Spring rate is the amount of force needed to compress a spring a certain distance.
- LSC - The low speed compression. This regulates the amount of energy the suspension necessary for the suspension to compress.
- HSC - The high speed compression becomes active when a big hit is taken like going throw a bump.
- LSR - The low speed rebound regulates the speed that the suspension takes to return.
- HSR - The high speed rebound regulates the speed that the suspension takes to return when big hits have been taken.
- tyre - The tyre used for the session.

Each data acquired by the sensors is stored in a table with this structure:
`Sensor_name(value, timestamp, session_id)`

- value - Corresponds to treated data after being sent by the sensor.
- timestamp - The timestamp at which the data was acquired, This timestamp has a resolution of 64 bits.
- session_id - The identifier of the session. This is related to the primary key of the session.

4.2.3 Insert data in the database

Simply using insert statements to insert data into the database was not giving enough insertion speed in order to cope with the acquisition rate. This is because SQLite waits for the data to be safely stored on the disk surface before the transaction is complete and each statement normally requires two complete rotations of the disk platter to commit. On a 7200RPM disk drive, this limits the number of transactions to 60 per second. In a 5400RPM disk drives, even less transactions per second will be possible. To overcome this two solutions may be considered.
One is to group several insert statements in a single transaction. This way, it can be achieved

much greater insertion speed. The other option is to tell SQLite not to wait for the data to be written to the disk surface. This will allow a faster insertion speed at the cost of safety (example: if a power loss occurs in the middle of a transation, the database file may become corrupt).
As such, it was choosen to group the insert statements in a single transaction. To achive it, statements are written into a string, as the data arrives. To do the commit of the statetements a timer is set and assures that data is committed at each second. With this interval of time, much less than 60 transactions per second are commited. But this option was to ensure that any computer would cope with it.

4.3 Data translation

The need for data translation comes from the fact that some sensors do not send directly the data that the user wants to know. For example, the suspension displacement sensor sends the potential difference. It is in the base station, that this translation is made. This could be done in the mobile station, but it makes much more sense to do this in the base station due to the fact that it has much more computational power. This translation is achieved by doing mathematical operations over the acquired data. Because each type of sensor requires diferent mathematical operations for the translation, the selection of the operation to be used is done based on the “type” of sensor. This way, the translation is independent of the CAN-bus id of the sensor. The data is stored in the database already translated, avoiding future translatation when the data is retrieved from the database. All the translation are done using a double. For example the translation of the speed sensor is:

$$\frac{778125}{CAN\_DATA} * 2 * 3.14 * 25 * \frac{3.6}{1700} \tag{4.1}$$

4.4 User interface

4.4.1 Requirements

The adopted programming language was C++ due to the fact that it is a lightweight language, object oriented, portable and has good libraries. As a consequence, a framework to develop the user interface in C++ was also considered. This framework should be operating system independent, offer a plot construction tool, video integration tools, as well as an user graphical interface to create new widgets and good support. The considered options were wxWidgets, GTK+ and Qt. All of them are operating system independent. The main difference is that wxWidgets uses the platform controls and utilities, which makes the interfaces developed using it to have a better visual integration with the rest of the applications. Despite the fact that all of them have good support, there seems to be a more active community around Qt. For the last reason Qt was adopted [13] [21] [14].

4.4.2 Features

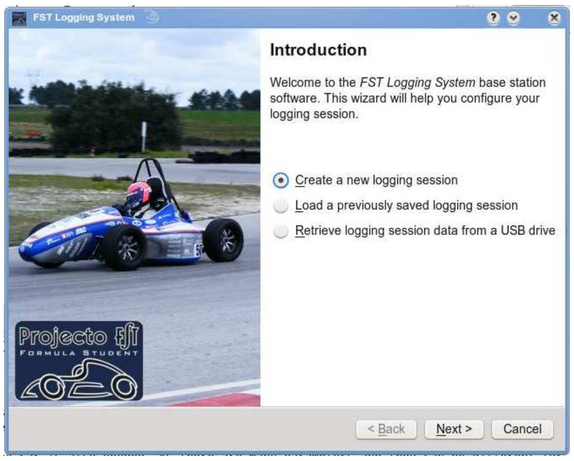
This version of the base station does not need the user to insert the CAN network configuration, since this data is loaded from the database. This limitation lead to the fact that the previous version only recorded the data from the sensors that had been configured. The translation of the data received from the mobile station was done at the widgets. Due to this the widgets were always needed, even if in the background, in order to have the translation. Now the translation are done in the same thread that deals with the reception of data from the mobile station. With this solution after the translation the widgets that need to be informed that new data has arrived are informed throw a feature of Qt that enables sending data to the widgets. This change was made to ease the future development and maintenance of the interface, since now all the translation is located in a single place.

4.4.3 Layout and functionality

The user interface has two main parts, the wizard and the main window with its widgets. The wizard is responsible for acquiring the data needed to configure the session. There are tree types of session:

- Live session - Used to acquire and show the data while the car is on track.
- Offline session - Used to analyse the data stored in the database.
- Upload data - Used to upload the stored data, This data normally comes from the USB drive used in the mobile station.

The first screen of the wizards asks the user which of these three sessions is wanted, as can be seen in figure 4.1.



Introduction

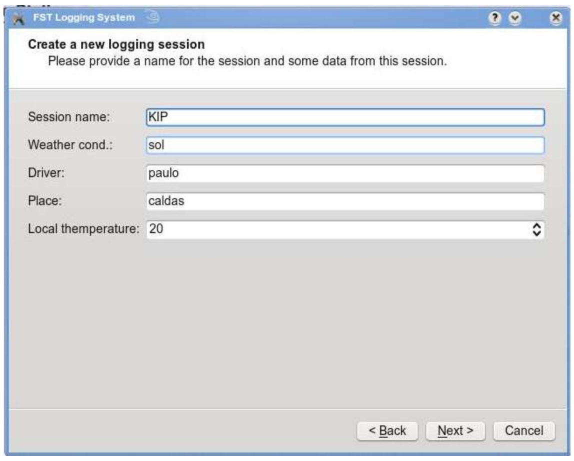
Welcome to the *FST Logging System* base station software. This wizard will help you configure your logging session.

[✓ Create a new logging session]
Load a previously saved logging session
Retrieve logging session data from a USB drive

Figure 4.1: Wizard intro

4.4.3.A Live session

In a live session the data is acquired through the wireless communication infrastructure, stored in the database and presented to user. The user is asked to enter the data about the car setup and session in the following screen, that is illustrated in figure 4.2.


FST Logging System

Create a new logging session
Please provide a name for the session and some data from this session.

Session name:	KIP
Weather cond.:	sol
Driver:	paulo
Place:	caldas
Local themperature:	20

Figure 4.2: Acquisition of data about the session

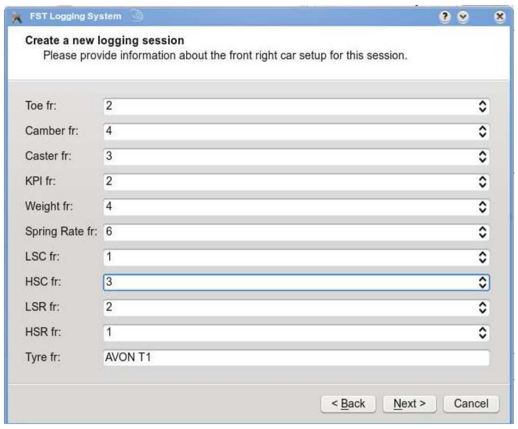
The filled information refers to the session name, the weather conditions, the driver, the place where the session is held and the local temperature. All the data, except the session name, are previously inserted with data from the last session, by retreiving it from the database. This information is important, since it gives the context to the data from the sensors and the configuration of the car.

The information about the setup also needs to be introduced, since there is no other way of getting this information. As there are many parameters and they can be different for each wheel of the car, a screen is presented for each wheel. To avoid the work of introducing the configuration of the car everytime a new session is created, the last configuration is fetched from the database. This way, the user only has to introduce the changes that were made to the previous setup. The layout of the screen can be seen in figure 4.3(a).

The user should also configure the data that he wants to see displayed on the plot. To do this, the user has to add the sensors that he wants displayed using the interface shown in figure 4.3(b). He can also put a name in the plot. By the end of this operation, the user is present with the screen shown in figure 4.4(a).

The widgets are dockable and can be presented in tabs. This way several presentation options are possible, as shown in figure 4.4(b).

Because the data being analyzed has many different natures, the user can use several plots at the same time. So, in order to add a plot the user only has to select add -> plot at the main window of the live session.



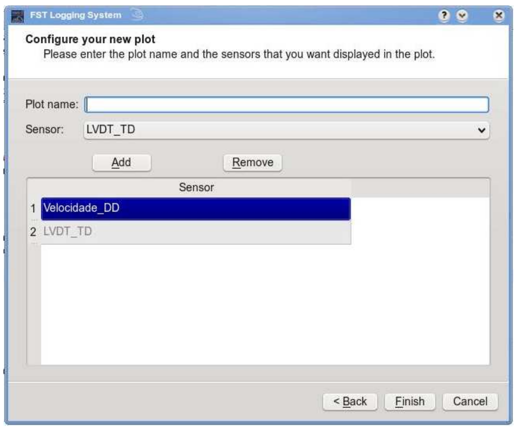
FST Logging System

Create a new logging session

Please provide information about the front right car setup for this session.

Toe fr:	2
Camber fr:	4
Caster fr:	3
KPI fr:	2
Weight fr:	4
Spring Rate fr:	6
LSC fr:	1
HSC fr:	3
LSR fr:	2
HSR fr:	1
Tyre fr:	AVON T1

(a) Acquisition of data about the setup



FST Logging System

Configure your new plot

Please enter the plot name and the sensors that you want displayed in the plot.

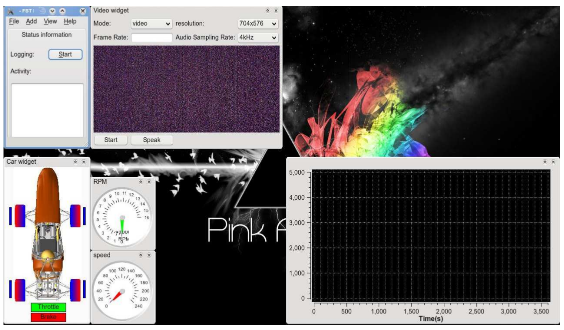
Plot name:

Sensor: LVDT_TD

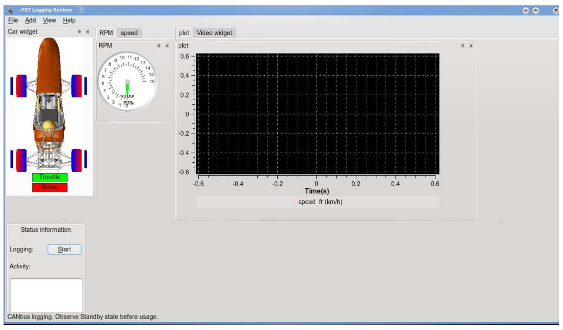
	Sensor
1	Velocidade_DD
2	LVDT_TD

(b) Plot configuration

Figure 4.3: Live session setup



(a) With floating widgets



(b) With docked widgets

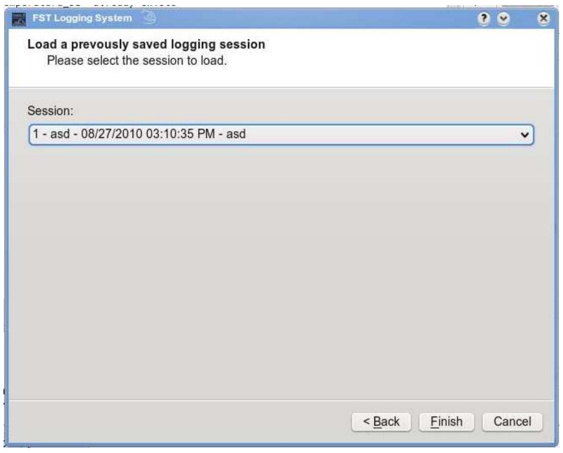
Figure 4.4: Live session

4.4.3.B Offline session

The offline session has the function of analysing the data that is stored in the database. In order to ease the selection of the session, a window like the one in figure 4.5(a) is presented with the stored sessions in decreasing order of the session date.

After this, the data is loaded into the plot and displayed. The user then has the possibility to multiply and give an offset to the values, in order to ease the analysis of the data. It is also possible to select which sensors have the data displayed in the plot. The window of the offline session can be seen in figure 4.5(b).

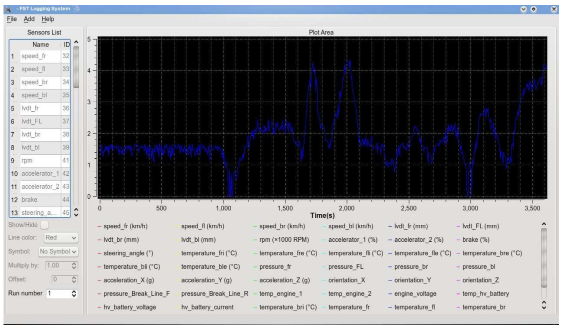
Although this is a good solution, from the analysis of the needs collected from the users, it was clear that it is desirable to extract the data from the database, so that it can be analyzed in specific external programs(e.g.MATLAB). As a consequence an easy way to extract the data was also developed and is described later.


FST Logging System

Load a prevously saved logging session
Please select the session to load.

Session:
1 - asd - 08/27/2010 03:10:35 PM - asd

< Back Finish Cancel



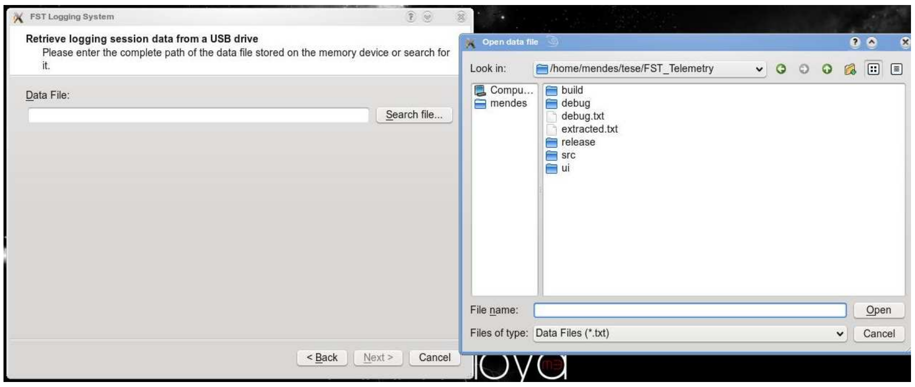
(a) Session selection

(b) Interface

Figure 4.5: Offline session

4.4.3.C Upload data

This option lets the user upload data to the database, by uploading a file with data from the sensors. As said before, these this files are created by the mobile station while it is on. This way, if the wireless communication infrastructure fails the data does not get lost. To upload data, the user has to insert the file to be uploaded by using the screen displayed in figure 4.6.


FST Logging System

Retrieve logging session data from a USB drive
Please enter the complete path of the data file stored on the memory device or search for it.

Data File:
Search file...

< Back Next > Cancel

Open data file

Look in: /home/mendes/tese/FST_Telemetry

Compu...
mendes

build
debug
debug.txt
extracted.txt
release
src
ui

File name: Open
Files of type: Data Files (*.txt) Cancel

Figure 4.6: Selection of file to upload

After this, if this file does not have a session associated with it (a file that was created without a session being started from the mobile station), then the user has to enter the data about the session in the same way that is done in a live session. In case the file already has a session associated with it, the data from the sensors regarding that session is deleted and the data contained in the file are uploaded.

4.4.3.D Car widget

The main objective of the car widget is to give a quick idea of the state of the car. Having that in mind, an image of the car was taken and edited so that information could be displayed on it. The result of this is a widget that displays the following information:

- The external tyre temperature. The temperature is indicated using colors, with blue being the colder and red the hotter temperature. As the temperature increases, the color slowly changes from blue to red. Since each tyre has two temperature sensors, one in the exterior part and the other on the interior, a gradient is done beetwen the interior and exterior of the tyre with the color that represents the temperature in that point.
- The steering position. This information is showen by turning the wheels into the angle indicated by the sensor.
- The suspension displacement. This is indicated by using bars next to the wheels. The bigger the blue bar is, the more suspension travel was used.

This widget can be seen in figure 4.7.

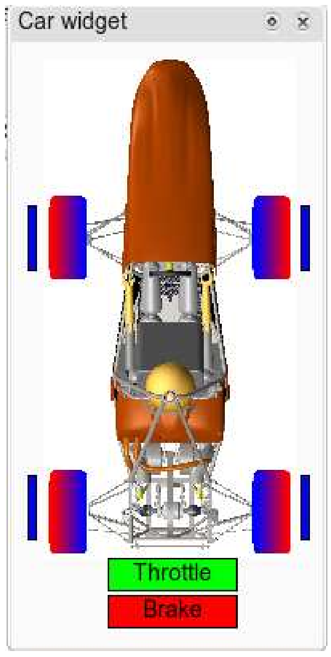


Figure 4.7: Car widget

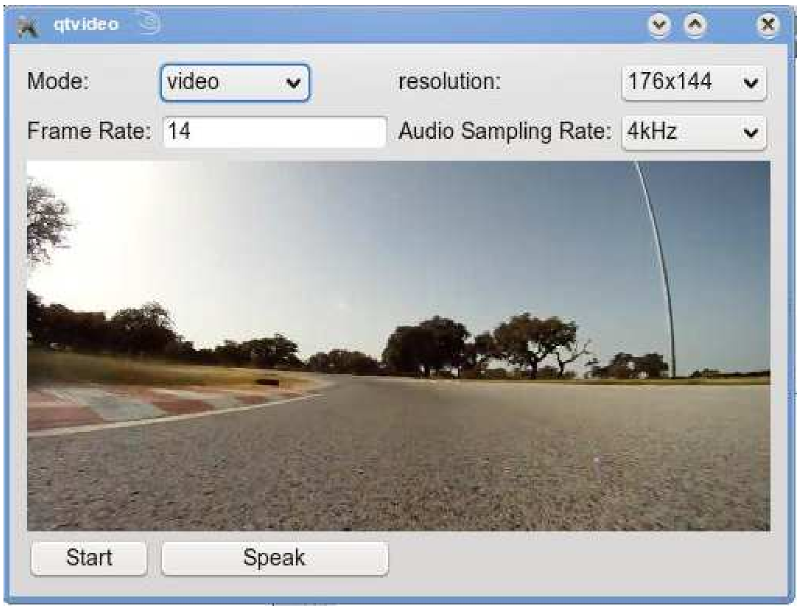
4.4.3.E Audio and video widget

For the new audio and video sensors a new widget to display and control these sensors had to be developed. The controls for these sensors are:

- Start and stop the capture of the audio and video signals.

- Choose between video or image snapshots, taken with a periodic delay.
- Video and snapshot resolution. This alllows the user to choose the image quality to use in any of these modes.
- The periodic delay for the snapshot capture.
- The frame rate for the video capture.
- The audio sampling rate to be used for the audio capture.
- A button to capture audio from the box side and send it to the car.

The widget can be seen in figure 4.8.



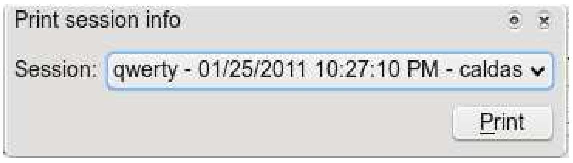
Mode: video
resolution: 176x144
Frame Rate: 14
Audio Sampling Rate: 4kHz

Figure 4.8: Video widget

In order to display the snapshot and video, a mplayer instance is started with commands to display photos or video. After this, the mplayer is attached and positioned on the widget, providing for a slimeless integration. For the audio, a mplayer instance is also started and attached to the widget.

4.4.3.F Information printing

One of the requirements that was received from the team was the application to be able to produce printed sheets with information regarding the setup of the car. Since all this information is stored in the database, it is now possible to print this information. This way, the team will have the information at hand, while still keeping it concentrated and coherent in a single place. The user just has to select the session he wants to print the setup. The screen to do this can be seen in figure 4.9.



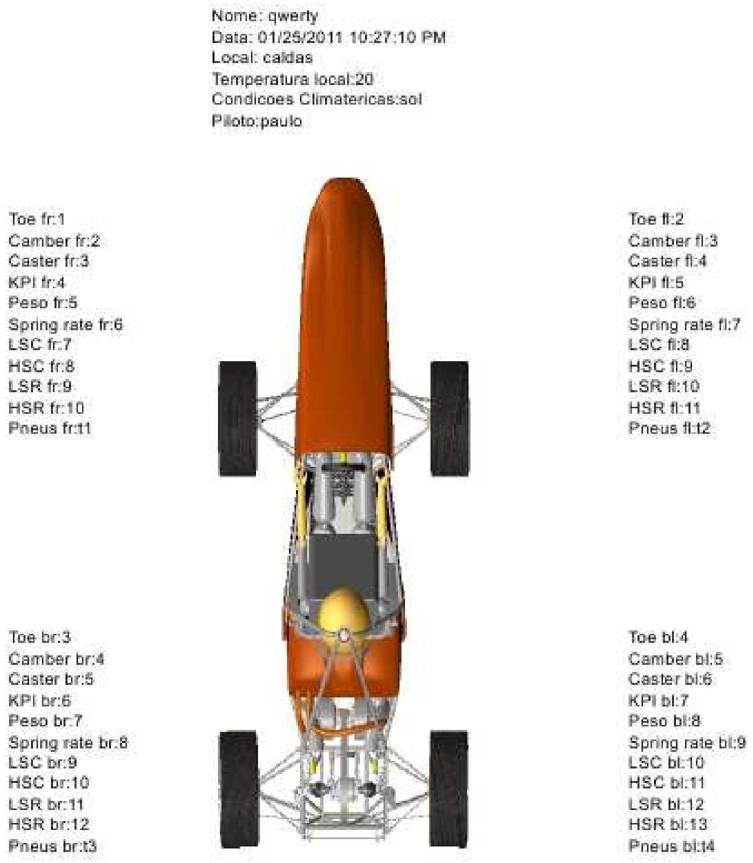
Print session info

Session: qwerty - 01/25/2011 10:27:10 PM - caldas

Print

Figure 4.9: Print session interface

The printed result can be seen in figure 4.10.



Nome: qwerty
Data: 01/25/2011 10:27:10 PM
Local: caldas
Temperatura local:20
Condicoes Climatericas:sol
Piloto:paulo

Toe fr:1
Camber fr:2
Caster fr:3
KPI fr:4
Peso fr:5
Spring rate fr:6
LSC fr:7
HSC fr:8
LSR fr:9
HSR fr:10
Pneus fr:t1

Toe fl:2
Camber fl:3
Caster fl:4
KPI fl:5
Peso fl:6
Spring rate fl:7
LSC fl:8
HSC fl:9
LSR fl:10
HSR fl:11
Pneus fl:t2

Toe br:3
Camber br:4
Caster br:5
KPI br:6
Peso br:7
Spring rate br:8
LSC br:9
HSC br:10
LSR br:11
HSR br:12
Pneus br:t3

Toe bl:4
Camber bl:5
Caster bl:6
KPI bl:7
Peso bl:8
Spring rate bl:9
LSC bl:10
HSC bl:11
LSR bl:12
HSR bl:13
Pneus bl:t4

Figure 4.10: Printed information

This result is achieved by importing the image of the car and scaling it to the desired size. After that text is written in text boxes according to the data they represent. The printing is done through an interface made available by Qt that enables to print an image in the same way images are created for the screen.

4.4.3.G Data extraction

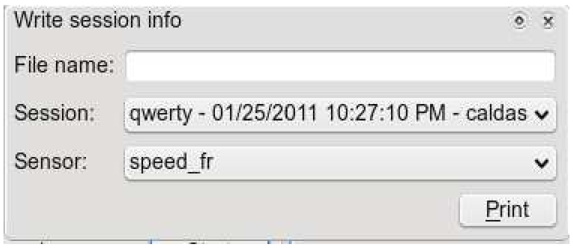
The FST team members use specific software to analyze the data coming from the car. A pratical and easy way to integrate with this software was one of the big lacks in the previous iteration of this project. To overcome this, it is now possible to easily extract information from the database. The user only needs to enter the filename that he wants the information to go to, the session and the sensor. The database in then queried and the data is extracted to the text file. The file has the following format:

value1;timestamp1

value2;timestamp2

...

The interface to do this can be seen in figure 4.11.



Write session info

File name:

Session: qwerty - 01/25/2011 10:27:10 PM - caldas

Sensor: speed_fr

Figure 4.11: Printed information

4.5 Communication

The communication between the two stations is a vital part of this project. It is the communication that lets the box transmit information in real time data regarding the state of the car and strategies to the race. A thread is launched in order to deal with all the communication. This thread also deals with data conversion and the storage in the database. This is done to avoid the locking the user interface while the other work is done. The communication infrastructure used is discussed in chapter 5.

5 Communication Infrastructure

Contents

In this chapter the hardware used to communicate between the two stations is described. Also the software that enables to have a reliable communication even after connection losses is described.

5.1 Hardware

To implement the communication between the car and the pitstop a WiFi connection will be used, by applying a WiFi USB adapter at the mobile station. Since it is connected through an USB port, an USB cable can be used to place the WiFi transceiver in the best place of the car in order to provide the necessary communication conditions. From the wide range of offers, the conducted selection was based on Linux compatibility, power and price. The part that best fitted this requirement was a SMC EZ Connect G wireless USB 2.0 adapter 54Mb. With this approach, it was achieved more bandwith, making it possible to have more exchange of information beetwen the car and the pits. This approach also presents more flexibility, since as the WiFi technology evolves, it will be possible to simply buy a new USB adapter and install it. The adopted WiFi USB adapter can be seen in figure 5.1 and its characteristics can be seen on table 5.1 [24].

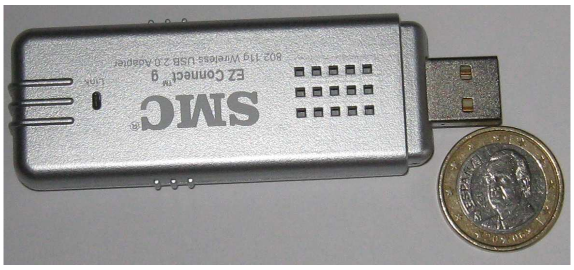


Figure 5.1: Wifi PCI adapter

Table 5.1: Characteristics of the WiFi USB adapter

Indoor range	75m at 1Mbps / 40m at 54Mbps
Outdoor range	350m at 1Mbps / 60m at 54Mbps
Transmit Power Output	16dBm at 1Mbps / 12 dBm at 54Mbps
Max. Data Rate	54 Mbps
Receiver Sensitivity	-92dBm at 1Mbps / -65 dBm at 54Mbps

The initial idea was to have two wireless adaptors working in ad-hoc mode, but this mode has bad support in Linux. As a consequence it was decided to put a wireless router next to the computer at the pitstop. This way, the Formula Student Championship rule that states that nothing can be placed in the track is not violated. The router characteristics can be seen in table 5.2 and the router can be seen in figure 5.2.

Table 5.2: Characteristics of the WiFi router

Transmit Power Output	15+/- 1dBm
Max. Data Rate	108 Mbps
Receiver Sensitivity	-88dBm at 6Mbps / -70 dBm at 54Mbps

Figure 5.2: Adopted router.

5.2 Configuration

The communication channel needs to be automatically setup when the operating system starts, so that the user only has to switch on the board. To accomplish this, some configuration files had to be created or edited:

- /etc/network/interfaces

  This file sets the configurations for the network. In the case the wireless card is configured to have a static ip, work in managed mode and use channel 11.

- /etc/init.d/fstnet

  This is a script that will execute the instructions that we needed at startup. Besides the configuration of the wireless link, this file can also be used to start all the programs. What this script does is kill the Network Manager, so that it will not interfere with the wireless configuration. The next step is to stop the wireless interface. Then it starts this interface according to the configurations in the /etc/network/interfaces file. The process of stopping and starting the interface is done using the ifdown and ifup command.

- /etc/rc.local

  This is the file used to indicate which scripts are required to run at startup. In this case, the /etc/init.d/fstnet is required to run.

This way, the communication channel will be available as soon as the system finishes the boot process.

5.3 Connection Recovery

With the configuration mentioned above, it is obtained a system that will have a communication channel at the end of the boot process, provided that everything goes well. But in case the router is turned off, the link can only be reconnected if the system is restarted. And more important if the connection goes down, normally because the system is out of range, it will not be able to recover the connection.

In order to solve this problem, a shell script was created that will act as a watchdog. What this script does is to check for the state of the connection using the iwconfig command. Then, whenever the connection is down, it will stop the wireless network interface and then restart it according to the configurations in the /etc/network/interfaces file. If the connection does not have any problem, nothing will be done. After this, the script will pause for 5 seconds. This protocol can be seen in figure 5.3.

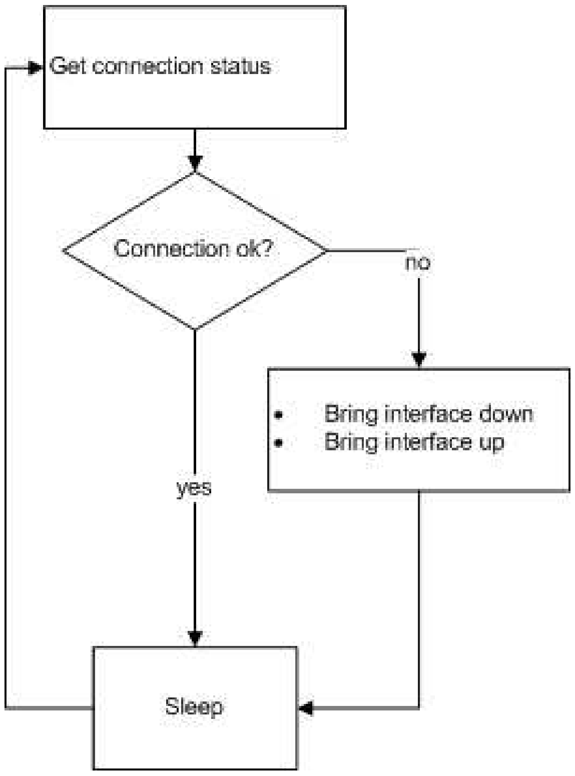


Figure 5.3: Connection recovery protocol

5.4 Protocol and message encapsulement

A reliable communication channel between the two stations is needed. Since connection losses will probably occur, it was required to define a resilient protocol that can deal with it. The adpoted protocol uses the package structure that can be seen in Table 5.3.

16 bits	8 bits	16 bits	The number of bytes indicated by the size	8 bits
Package Number	Size	Type	Data	Checksum

Table 5.3: Package structure that is used in the communication beetween the two stations

- The package number field is used to give the order of the packages and detect lost packages. The package number starts in a randomly generated number by the base station. This number is then sent to the mobile station and from this point it is incremented by each package sent.
- The size field indicates the size of the data field in bytes. This is important so that the receiver of the package knows where the data field ends, since it has a variable size.
- The type field is used to indicate the type of the package and the sensor from where the data came. For example, the CAN identifier used by a given sensor is what is sent in this field. The CAN identifier occupies 11 bits, so the rest is available for other sensors, such as the video and audio acquisition, and for control messages.The control messages have the last 16 addresses reserved.
- The data field is where the information that is sent to the other station is placed. As said before, this field has a variable size indicated in the size field. A fixed size could be used, but that would lead to a waste of bandwith.
- The checksum field is used to ensure the integrity of the package. This is achieved by the sender computing the checksum, and putting it in this field of the package. The receiver will then receive the message and compute the checksum of the data field. If it is equal to the checksum field of the received message, then no corruption of the message exists. Otherwise, corruption exists and the message must be discarded.

The resilient communication protocol can be split into two different parts: the sender and the receiver. Although both stations will act as sender and receiver, the mobile station will act much more as a sender, since it is where most of the data is being acquired.

5.4.1 Sender

The sender dispatches the packages as soon as they are available and stores them on a circular list, so that they are available in case of a retransmission is needed. When an acknowledge of the reception of that package is received, that package is removed from the list. If in a certain period of time an acknowledge for a package is not received, it is considered lost, in which case

it is retransmitted. If the sender receives a retransmission request, that package will be also retransmitted.
The use of a circular list with fixed size enables to save the allocation time upon the addiction of a new package. This circular list has also the particularity that it is no possible to write over an element that has not been erased, which is done to avoid writing over a package that has not been acknowledged.

5.4.2 Receiver

The receiver makes use of two different lists. One is for storing the packages that have been received, when the previous packages to it have not been received yet. In this case the package is stored in that list to avoid the retransmission of this package - the list is called a waiting list. The other list is for packages that are in order and from which all the previous packages have been received - this one is called available list. This is the one that stores the packages for the user to read.
The protocol for the receiver is as follows: after receiving a package it will check its integrity using the checksum. If it is not valid, a package request will be sent and that package is discarded. Otherwise an acknowledge is sent. After this, the package number will be checked to see if it is equal to the expected package number. In case it is different, it will be checked to see if it is in the boundaries of the reception window. The window has the same size as the list in the sender. If it is in that boundary, the package is stored in the waiting list; otherwise it is discarded. If the package number is the expected, the package is stored in the available list and the package number is increased. After that, the waiting list will be searched to find a package with the expected number; if it is, found the procedure is the same as when a package with the expected package number is received. During this search, if a package that is not within the boundaries of the window is found, that package is removed from the waiting list. The receiver protocol is illustrated in figure 5.4.

The receiver uses a consumers-producers concurrency protocol in order for the consumer to be blocked when there is no available packages. With this approach, the user can have a dedicated thread responsible for receiving the packages.

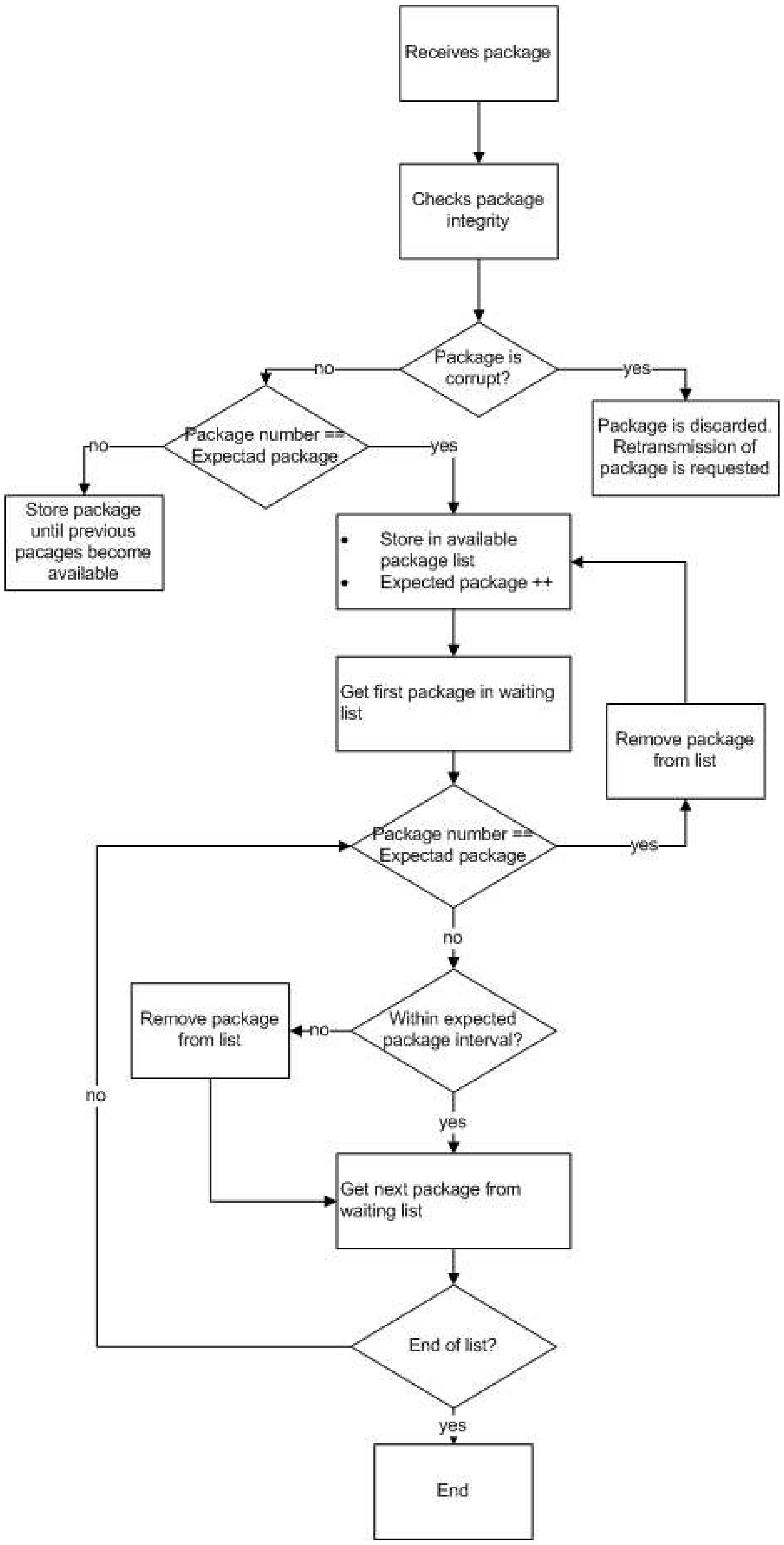

Receives package
Checks package integrity
Package is corrupt?
no
yes
Package is discarded. Retransmission of package is requested
Package number == Expectad package
no
yes
Store package until previous pacages become available
Store in available package list
Expected package ++
Get first package in waiting list
Remove package from list
Package number == Expectad package
yes
no
Within expected package interval?
Remove package from list
no
yes
Get next package from waiting list
End of list?
no
yes
End

Figure 5.4: Receiver protocol

5.5 Priority queues

On top of this protocol, a set of priority queues were developed. The reason for this priority queues is to simplify the development and to maintain a small size of the sender list. Three types of priority queues were created: high, medium and low priority. This way, the packages are not sent in a first in, first out order, but in a way that the more relevant packages will be transmited first. To ease the definition and separation of the priority of the acquired signals, this will be done using three intervals, by using the identifier number as the priority level. This way by only setting the identifier number the priority can be changed. The intervals can be seen in table 5.4.

Table 5.4: Package priority

Priority status	**ID**
High	0 - 32768
Medium	32769 - 49152
Low	49153 - 65536

With this division the packages coming from the CAN-bus will have higher priority, since the CAN identifiers go from 0 to 2048. The medium and low priority queues are specially tailored to audio and video respectively. The medium and low priority queues have a small size, when these are full and a new package arrives the oldest packages is discarded. This is the normal behavior in live streaming of audio and video. With this approach incorporated we can add new sensors without interfering with the signals from the CAN.
With the adopted protocol using the priority queues the message is only sent if the priority queue of that package and the higher priority queues are empty. In case the send fails, probably because the sender list is full, the package is stored in its priority queue. This protocol can be seen in figure 5.5.

When a package is to be removed from the sender list, the queues will be checked in decreasing order of priority to see if they have packages to send. If that is the case, the package is replaced by the first to be found on the queues (starting to lock in the higher priority queue). The protocol for this operation can be seen in figure 5.6.

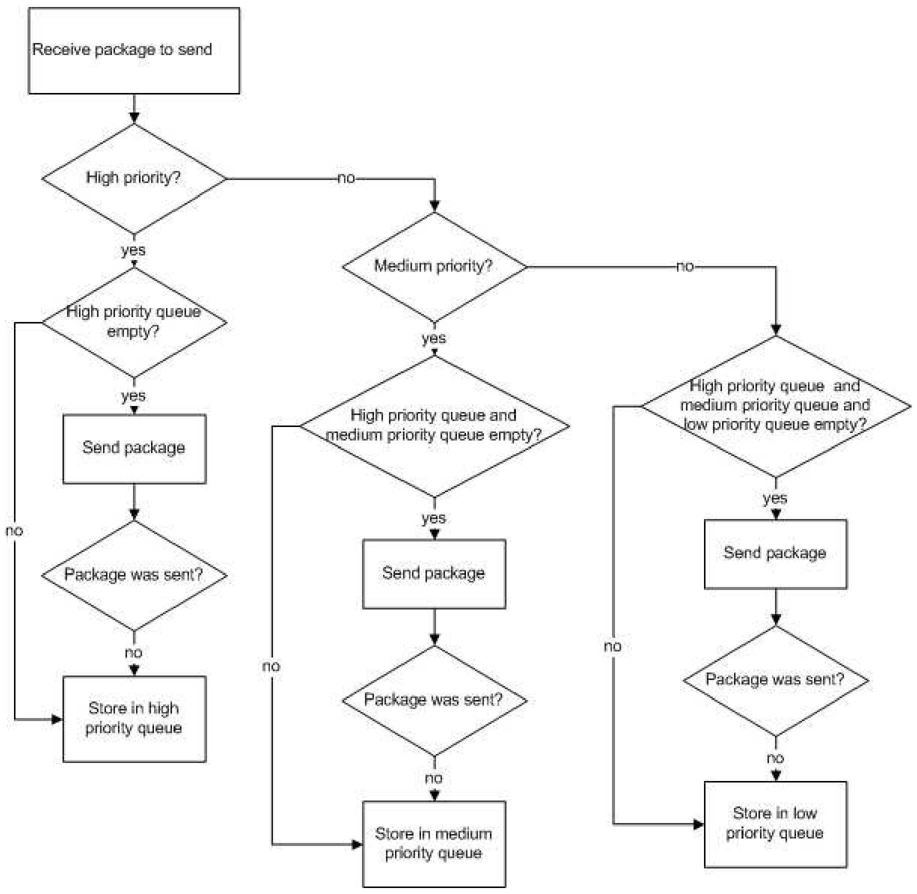

Receive package to send
High priority?
no
yes
High priority queue empty?
yes
Send package
no
Package was sent?
no
Store in high priority queue
Medium priority?
no
yes
High priority queue and medium priority queue empty?
yes
Send package
no
Package was sent?
no
Store in medium priority queue
High priority queue and medium priority queue and low priority queue empty?
yes
Send package
no
Package was sent?
no
Store in low priority queue

Figure 5.5: Priority queues insert message

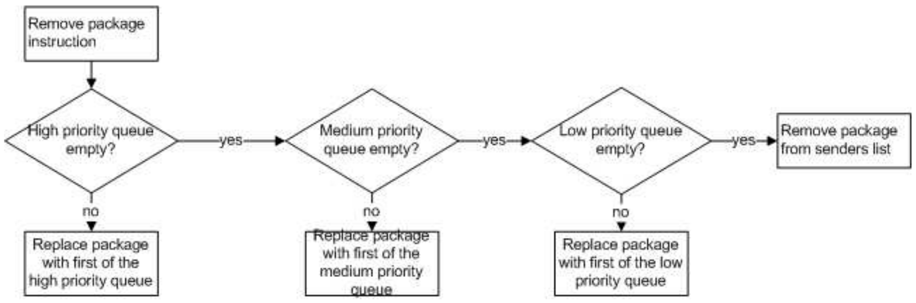

Remove package instruction
High priority queue empty?
yes
Medium priority queue empty?
yes
Low priority queue empty?
yes
Remove package from senders list
no
Replace package with first of the high priority queue
no
Replace package with first of the medium priority queue
no
Replace package with first of the low priority queue

Figure 5.6: Priority queues remove message

6 Results

Contents

In order to evaluate the entire system's performance tests where made to evaluate the system. This tests tried to simulate the worst cases that the system will have to handle.

6.1 CAN acquisition rate

It is important to know the speed at which the data is acquired from the CAN-bus to evaluate the rate at which the database will grow and that the available communication bandwith is enough. Considering that each CAN-bus package actually has 76 bits, 44 from the protocol and 32 of data, and the maximum CAN-bus speed being 1Mbit/sec, it leads to the maximum number of samples per second to be:

$$\frac{1 * 1024 * 1024}{76} = 13797.05 samples/s \tag{6.1}$$

This corresponds to an useful data rate of:

$$\frac{13797.05 * 32}{1024 * 1024} = 0.42 Mbits/sec \tag{6.2}$$

6.2 Database speed

To analyse the insertion speed of the database, a benchmark program was run to see if the database provides enough insertion speed to deal with the rate at which the data is acquired from the car. The equivalent to 1 minute of acquired data at the maximum speed of the CAN

$$13797.05 * 60 = 827820 signals \tag{6.3}$$

took 4 second to insert. This proves that database insertion speed is not an issue.

6.3 Database grow speed

The database speed grow in relation to the recorded data is:
32 bits - timestamp
64 bits - value
16 bits - session id

$$(32 + 32 + 16) * 13797.05 = 1103764 bits/sec \tag{6.4}$$

$$\frac{1103764}{8 * 1024 * 1024} = 0.13 MB/sec \tag{6.5}$$

So with a 10 minutes session we will gather:

$$0.13 * 60 * 10 = 78 MB \tag{6.6}$$

The size of the session conditions and setup is not static, since it depends on the size of the text values entered. Nevertheless, these values can be discarded, since they are much smaller than the sensor data.

6.4 Communication range

Since the racing circuit is not always available for testing the system, a good solution for testing the wireless connection and to overcome this setback is to choose a place that is not necessarily a racing circuit but that has the same obstacles that may interfere with the communication between both stations. It was considered an hypotetical worst case scenario by making the tests in a place with obstacles such as concrete structures, trees and uneven ground. Figure 6.1 shows the area around the installations of Instituto Superior Técnico in Taguspark - Porto Salvo, Portugal; where the tests were made. The base station is located about 2 to 3 meters high. The mobile station moves around the area colored as green and red. The green area is where the signal sent from the base station is strong enough to recover the connection to the mobile station (connection range). The red area is where the signal gets so weak that in case of connection lost, it becomes impossible to re-connect. The X marks the exact spot where the connection was lost in this particular test. The main reason is that there is a concrete structure and trees between the two nodes, plus the spot is located in a cavity about 2 meters below the ground level. This corresponds to the worst case scenario. The signal output power is 15 dBm. Around the green area the average data rate is 24 Mbps and is 18 Mbps in the border between the green and red zone,which is about 150 meters away from the base station. When the mobile station is close to the X spot (which is about 30 meters further), the data rate is about 2 Mbps. The highest data rate in the red zone is 11 Mbps. It should be noted that the highest distance from the pit (base station) to another point in the racing circuit where the tests are usually made is about 300 meters. Not having enough range to cover the hole track was expect, that was the reason for the creation of the communication protocol. Furthermore there are usually less obstacles and the ground is more even.

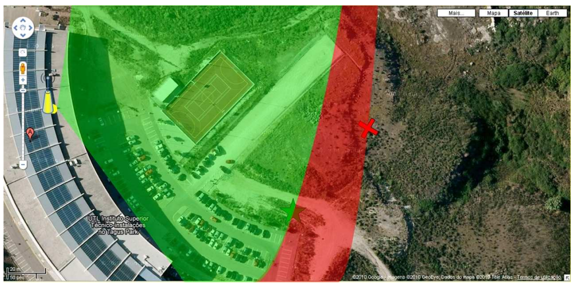


Figure 6.1: Results of the range test.

6.5 Communication protocol performance

The communication protocol has two main parts to test: the speed and the resilience.

6.5.1 Resilience test

The resilience was tested during the communication range test. This was done by runing a program at the mobile station sending video and a text file (to simulate a CAN sensor) with 4Mbits in a infinite loop. The base station received both data and checked that the text file was received in the right order.
During this test all the text messages where delivered in the right order, even after the connection was recovered. This shows that the protocol can correctly deal with connections losses. Also, the priority queues worked as expected, since the video packages where the last to be retransmited.

6.5.2 Speed test

To analyse the speed of the communication protocol a file with 175.2 MB was transferred beetwen the two stations using the resilient communication protocol. This test was made with the two stations near each other and the times were measured. Using the TCP/IP it took 2m09s to transfer, while using the resilient protocol it took 5m20s. With this result, it can be calculated that the transfer speed using the resilient protocol is of

$$\frac{175.2 * 8}{5 * 60 + 20} = 4.38 Mbits/sec \tag{6.7}$$

For the TCP/IP the tranfers rate was of

$$\frac{175.2 * 8}{2 * 60 + 9} = 10.87 Mbits/sec \tag{6.8}$$

Although the degradation introduced by the use of the resilient protocol is significant, the achived bit-rate is still much bigger than the maximum data rate of the CAN-bus, which is of 0.42 Mbits/sec. This gives a margin of about 3.96 Mbits to transfer the data that could not be transmitted, while the connection was lost, and the data of the sensors outside the CAN-bus.
As an example the acquired data during a lap at Palmela circuit, and having in account that each takes about 1 minute, is:

$$60 * 0.42 = 25.2 Mbits \tag{6.9}$$

Considering a 24Mbtis/sec bandwith, wich is the bandwith that is available during most of the time, the data from an entire las is transfered in;

$$\frac{25.2}{4.38} = 5.75 sec \tag{6.10}$$

So the communication infrastructure can clearly handle the acquisition rate of the CAN-bus as well as other sensors.

7 Conclusions and future work

Contents

7.1 Conclusions

The presented thesis proposes a telemetry system designed to be integrated in a car to enter a Formula Student competition. The system was designed with two stations: a mobile and a base station.

The mobile station was improved by using a more powerfull hardware. Which together with the several interfaces, greatly improved the flexibility of this station. The new hardware allowed the instalation of an operating system that greatly eases the development of new software. So it is possible to run other programs beyond the ones presented. This resembles the solution presented by McLaren that allows other programs to be run. The access to this station is now easier, since it is possible to connect through a ssh client or a serial console. The connection to the CAN-bus was achieved through a CAN-USB transceiver, since the new hardware has no CAN interface. This station now works totally independent from the base station, functioning also as a black box. Since it logs all data received from the CAN without the need for the base station to request the start of the logging. No data is lost due to improper shutdown, due to the extra battery and sensor. With this feature the solution become more similar to the behavior of professional data loggers, that can work without a base station. In therms of communication the previous solution presented package losses in the test track. The current solution uses Wi-Fi in order to achieve more bandwith. To solve the loss of packages a protocol was developed that ensured reliable communication. With the new communication it is possible to have live telemetry in wider tracks. In therms of data storage 3GB are available - which is more than the presented industrial solutions have. Three new sensors where added: a camera, a microphone and a GPS.

The data storage in the base station was improved in relation to the previous prototype by the use of a database engine, allowing the data to be stored in a single file and giving easy access to it. This solution is in some aspects similar to the one used by McLaren, since it also uses a database. But the features of the database are very different. So with the presented solution it is not possible to have several databases synchronized, nor is it possible to have several users accessing it. The presented solution takes an approach more appropriate to the FST project, by enabling the share of the database by simply copying the database file. Furthermore a solution using Microsoft SQL Server would be much more expensive, forcing the users to install software to access it and having a computer to install the database. Data became more integrated since now all data is stored in the database. This includes CAN configuration, car configuration, track conditions and data from the sensors. Whereas the previous version only stored data from the sensors and in separated files. Is possible to import data from other sources(for example simulated data) similarly to McLaren solution, since the format of the imported files is known. And like both professional solutions it is possible to extract to files the data.

The base station interface was improved by allowing a session to have multiple plots. A widget to handle the transmition of audio and video was created. Also a widget to incorporate data regarding the steering angle, suspension displacement, tire temperature and brake and throttle usage was created to save space on the screen. It became possible to print the setup of the car.

7.2 Future work

As possible future enhancements of the system, it is proposed to further develop the offline session by making it easier to use. Also the integration of this system with the GPS, making it possible to analyse the graph having into account the location of the car. The graph tools provided by QT have some limitations. Explore new languages that provide more powerfull tools might be a better option.
Through the base station it could be possible to send specific CAN messages in order to solve possible errors in the CAN-bus. This option needs some caution to guarantee that no malfunction occurs. Although no serious danger can be done since the CAN-bus is not responsible to transmit data regarding the control of the car. Also a new CAN-bus transceiver could be built together with the sensor to detect that the mobile station is being powered by the auxiliary battery. This not only will decrease the size of this solution, but more important, should solve the problem of the existing transceiver that losts data when too many CAN packages are received.

Bibliography

[1] AIM Technologies. EVO4 User manual Release 1.03.

[2] AIM Technologies. RSA Main improvements ver. 2.30.00.

[3] Slvio Fernandes Calunda. Formula student racing championship: Implementation of the on-board audio and video acquisitionand encoding system. Master's thesis, IST, 2010.

[4] David Rua Copeto. Automotive data acquisition system - fst. Master's thesis, IST, 2009.

[5] Cosworth. http://www.cosworth.com/ last visited on 12/2009.

[6] Cosworth. Pi Toolbox User Guide.

[7] Rui Alexandre de Sousa Andrade. Sistemas electrnicos para o projectofst. Master's thesis, IST, 2009.

[8] Formula Student. http://www.formulastudent.com/ last visited on 12/2009.

[9] Robert Bosch GmbH. CAN Specication Version 2.0, 1991.

[10] The PostgreSQL Global Development Group. PostgreSQL 8.4.2 Documentation. The PostgreSQL Global Development Group.

[11] Sibsankar Haldar. Inside SQLite. O'Reilly Media, 2007.

[12] Egon Jhnk Harald Eisele. PCA82C250 / 251 CAN Transceiver, 1996.

[13] Stefan Csomor Kevin Hock. Cross-Platform GUI Programming with wxWidgets. Prentice Hall, 2005.

[14] Andrew Krause. Foundations of GTK+ Development. Apress, 2007.

[15] McLaren Electronic Systems. ATLAS, Advanced Telemetry Linked Acquisition System, 2009.

[16] McLaren Electronic Systems. EDR-400, Enhanced Data Recorder, 2009.

[17] McLaren Electronic Systems. HIGH SPEED DATA LOGGER, HSL-500, 2009.

[18] Microchip Technology Inc. Microchip Technology Inc. Application Note 713 - Controller Area Network (CAN) Basics, 2002.

[19] MySQL. MySQL 5.5 Reference Manual, 2009.

[20] Projecto FST. http://www.projectofst.com/ last visited on 12/2009.

[21] Qt Development Frameworks. Qt Reference Documentation. http://qt.nokia.com/doc/4.5/ last visited on 12/2009.

[22] Race Technology. http://www.race-technology.com/ last visited on 12/2009.

[23] SMC Networks. SMC EZ Connect G wireless USB 2.0, 2009.

[24] SMC Networks. SMCWUSB-G EZ Connect g 2.4GHz 802.11g Wireless USB 2.0 Adapter, 2009.

[25] VIA Technologies, Inc. VIA EPIA-P700 datasheet, 2009.

A

Acquired signals

Table A.1: Acquired Signals

ID	Description	Frequency [Hz]
0	Synchronization	-
32	Speed DD	10
33	Speed DE	10
34	Speed TD	10
35	Speed TE	10
36	LVDT DD	20
37	LVDT DE	20
38	LVDT TD	20
39	LVDT TE	20
41	RPM	10
42	TPS 1	20
43	TPS 2	20
44	BPS	20
45	Steering angle	20
51	IR DDi	5
52	IR DDe	5
53	IR DEi	5
54	IR DEe	5
55	IR TDi	5
56	IR TDe	5
57	IR TEi	5
58	IR TEe	5
61	Pressure DD / Temperature DD	1
62	Pressure DE / Temperature DE	1
63	Pressure TD / Temperature TD	1
64	Pressure TE / Temperature TE	1
71	Acceleration X	20
72	Acceleration Y	20
73	Acceleration Z	20
74	Orientation X	20
75	Orientation Y	20
76	Orientation Z	20
81	Pressure Break Line F	1
82	Pressure Break Line R	1
85	Temp Engine 1	1
86	Temp Engine 2	1
87	Engine Voltage	10
91	Temp HV Battery	1
92	HV Battery Voltage	10
93	HV Battery Current	10

B

Operating system installation

The installation and configuration of the mobile station operating system should be as follows.

1. Install Knoppix 6.0.1 to flash drive.

   This can be done by using the install to flash drive program.

2. Install the ssh server.

   The command is “apt-get apt-get install openssh-server”.

3. Install the at program.

   The command is “apt-get apt-get install at”.

4. Configure the runlevel.

   Change the line that has “id:5:initdefault:” to “id:3:initdefault:”.

5. Configure the wireless conection.

   Change the file “/etc/network/interfaces”, and at the end of the file, add:

   ```
   auto wlan0
   iface wlan0 inet static
   address 192.168.1.100
   netmask 255.255.255.0
   wireless-channel 1
   wireless-essid fst
   ```

```
wireless-mode ad-hoc
```

- create the file /etc/init.d/fstnet with:

  ```
  killall NetworkManager
  ifdown wlan0
  ifup wlan0
  ```

- make the /etc/init.d/fstnet file executable, “#chmod 755 /etc/init.d/fstnet”.

- at the end of the /etc/rc.local add “/etc/init.d/fstnet”

6. Start the reconnect script.

   Create the /etc/init.d/reconnect file with

   ```
   #!/bin/bash
   iwconfig wlan0 > /tmp/iwconfig_stat
   while :
   do
   grep -q ”00:13:F7:6D:24:20” /tmp/iwconfig_stat #change MAC address in case a new router
   is used
   if [ $? -eq 1 ] #grep ”Not-Associated” ”/tmp/iwconfig_stat”
   then
   ifdown wlan0
   ifup wlan0
   fi
   sleep 5
   iwconfig wlan0 > /tmp/iwconfig_stat
   done
   rm /tmp/iwconfig_stat
   echo 0
   ```

   make the /etc/init.d/reconnect file executable, “#chmod 755 /etc/init.d/reconnect”.

   In the /etc/rc.local file add: “at now +1 minutes /etc/init.d/reconnect&”

7. Start the mobile station program at startup.

Place the executable file in /root/carro and make it executable by running “#chmod 755 /root/carro”.

- Create the /etc/init.d/mobileStation file with:
- /root/carro&
- make the /etc/init.d/baseStation file executable, “#chmod 755 /etc/init.d/baseStation”.
- Add the to the /etc/rc.local the line “t now +1 minutes /etc/init.d/baseStation”.

C

Programmers manual

The diagram of the software arquitecture of the base station is presented in figure C.1.

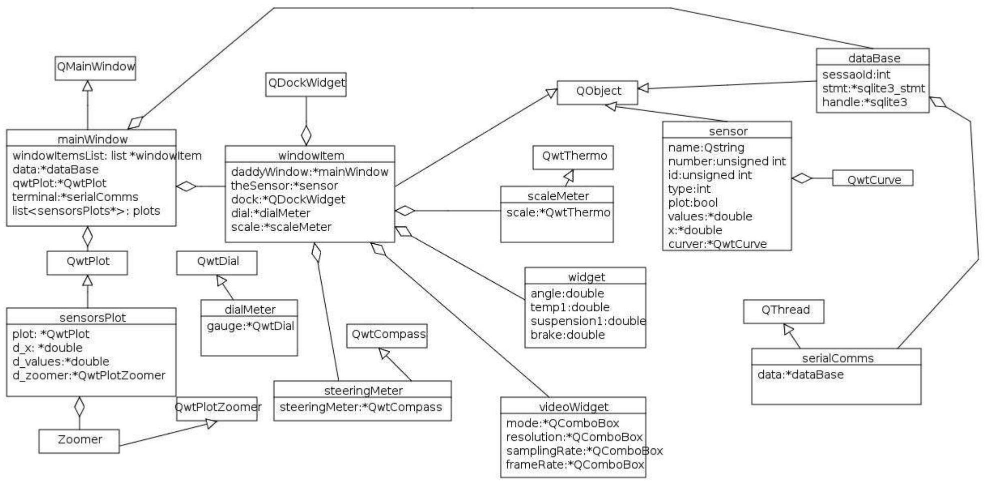


Figure C.1: Class diagram of the base station.

C.1 Sensor class

sensor(dataBase $*$db, const QString &sensorName, int sensorId, int sensorType)

Constructor of the sensor class.

db - the current database

sensorName - the name of the sensor

sensorId - the id of the sensor

signalLaunch()
signals that a new signal of this sensor has been received.

newValueSlot()
updates the values of the sensor.

C.1.0.A dataBase class

dataBase()
Constructor of the dataBase. Creates the tables for the sensors if they do not exist.

dataBase::CriaSessao(const QString fileName,const QString local,int tempLocal,const QString condClimaterica,const QString driver,int toe_fr,int camber_fr,int caster_fr,int KPI_fr,int weight_fr,int spring_rate_fr,int LSC_fr,int HSC_fr,int LSR_fr,int HSR_fr,const QString tyre_fr,int toe_fl,int camber_fl,int caster_fl,int KPI_fl,int weight_fl,int spring_rate_fl,int LSC_fl,int HSC_fl,int LSR_fl,int HSR_fl,const QString tyre_fl,int toe_br,int camber_br,int caster_br,int KPI_br,int weight_br,int spring_rate_br,int LSC_br,int HSC_br,int LSR_br,int HSR_br,const QString tyre_br,int toe_bl,int camber_bl,int caster_bl,int KPI_bl,int weight_bl,int spring_rate_bl,int LSC_bl,int HSC_bl,int LSR_bl,int HSR_bl,const QString tyre_bl)
Creates a new session with the given parameters.

C.2 Envio class

Envio()
Constructor of the class.

Envio()
Destructor of the class.

void Reenvio()
Resends the messages contained on the list.

lmsg Recebe()
Allows the user to received messages from wich all previous messages have been received.

Blocks if no message is available.

Returns the next message to be received.

void EnviaPacote(int nPacote)

Resends the specified message.

int EnviaMsgSemGuardar(unsigned short size, void∗ msg)

Sends a message without storing it in the circular list.

size - the size of the message.

msg - the message to send.

Return -1 on error and 0 on sucess.

void GuardaNaFila(unsigned short size, void∗ msg, unsigned short ID, queue¡msgEnviar¿ ∗filaInserir)

Stores a message in the specified queue.

size - size of the message.

ID - ID of the message.

filaInserior - queue in wich to store the message.

int EnviaMsg(unsigned short size, void∗ amsg, unsigned short ID)

Regular way of sending a message. This way the message is stored on the circular list if space is available or in the apropriate queue.

size - size of the message.

amsg - message to send.

ID - id of the message.

Return -1 on error and 0 on sucess.

int EnviaMsgDirecto(unsigned short size, void∗ msg, unsigned short ID)

Sends the message directly. The message is also stored in the circular list.

size - size of the message

msg - message to send.

ID - id of the message.

Return -1 on error and 0 on sucess.

void∗ CriaMsg(void∗ msg, unsigned short size, unsigned short tipo, unsigned short nMensagem)

Creates a message.

msg - data to send in the message.

size - size of the data to send.

tipo - id of the message.

nMensagem - number of the message. The number used to check that all messages are delivered in the right order.

Returns the created message.

unsigned char Checksum(void$*$ msg, unsigned short size)

Computes the Checksum.

msg - the message to calculate the checksum.

size - size of the message.

Returns the result of computing the checksum.

int Getsock()

Returns the current socket.

Receive(void $*$arg)

Thread that handles the reception of incoming messages. Here the integrety of the packages is checked. The order is guaranted by only making the ordered packages available to the Recebe() method.

C.3 Lista class

Lista(int tamanho)

Construtor of the class. Initiates the circular list and the queues.

tamanho - the size of the list.

Lista()

Destructor of the class.

int InsereMsg(unsigned short nMsg, void $*$msg, unsigned short size)

Inserts a message in the list.

nMsg - number of the message.

msg - message to send.

size - size of the message.

Returns -1 on error and 0 on sucess.

int RemoveMsg(unsigned short nMsg)

Removes the message it the specified number.

nMsg - number of the message to remove.

Returns -1 on error and 0 on sucess.

lista_msg* RetornaPacote(unsigned short nMsg)

Returns the message with the given number.

nMsg - number of the message to be retrieved.

Returns NULL on error or the message on success.

int GetTamanho()

Return the size of the list.

int Vazia()

If the list is empty return 0 else returns -1

C.4 mainWindow Class

mainWindow(QWidget * parent, Qt::WFlags f)

Constructor of the class.

parent - parent of this window.

f - flags to start this window.

void createMenus(bool notLoad)

Creates the menus.

notLoad - FALSE in case it is an offline session.

void addPlotMenu()

Shows the menu to add a new plot.

void addPlotMenu()

Shows the menu to print the data from a session.

void extractTableMenu()

Shows the menu to extract data from a table of the database.

void createActions()

Associates the buttons with the actions.

void timerSwitch(bool on)

Turns the refresh of the plot on or off.

on - True to start the periodic refresh of the plot.

void timerEvent(QTimerEvent *)

Timer activated function that refreshes the plot.

void mainWindowStart()

Creates the widgets according to the type of session selected and starts the communication with the mobile station in case an online session is selected.

windowItem* buildSensorListItem()

Creates the widget that display the available sensors to add to the plot.

windowItem* buildPrintItem()

Creates the widget responsible to print the session data.

windowItem* mainWindow::buildWriteItem()

Creates the widget that enables to select which data will be extracted from the database.

void print()

Prints the data about the selected session.

void buildLoadSessionWindow()

Builds the widgets for the offline session.

void updateCurveView(int state)

Enables or disables the view of the currently selected sensor on an offline session.

void updateCurveColor(int color)

Changes the color of the currently selected sensor on an offline session.

color - red level of the color.

void updateCurveSymbol(int style)

Changes the simbol used on the line of the currently selected sensor on an offline session.
style - style of the line to use.

void updateCurveMultiply(double constant)
Multiply the value of the currently selected sensor by constant.

void updateCurveOffset(int constant)
Increases the value of the currently selected sensor by constant.

void updateEditors()
Update the currently selected sensor on an offline session.

C.5 windowItem Class

void print()
Prints the values of the selected session.

void addToTable()
Adds a sensor in the selection of sensors to add to the plot. It is connected with the add button

void removeFromTable()
Adds a new sensor in the selection of sensors to add to the plot. It is connected with the remove button

void buildCarItem()
Builds the car widget.

void buildVideoItem()
Builds the video widget.

void buildSpeedItem(QString name, windowItem *item, bool edit, sensor *s)
Builds the speed widget.
name - name to give to the sensor.
item - widget to associate with.
edit - true to associate with a widget.
s - sensor to associate the widget with.

void buildRPMItem(QString name, windowItem *item, bool edit, sensor *s) Builds the speed widget.
name - name to give to the sensor.
item - widget to associate with.
edit - true to associate with a widget.
s - sensor to associate the widget with.

void savePlotImage()
Saves the current plot to an image. Associated with the save plot as image button.

sensorsPlot ∗buildPlotItem(const QString plot_name)
Builds a new plot.
plot_name - name to give to the plot.
Returns the new plot.

void addSensorToPlotItem(QString name, sensor ∗s, int h, sensorsPlot ∗plot)
Adds sensor to the plot.
name - name of the sensor.
s - sensor to add.
h - color.
plot - plot to the the sensor.

void destroyPlotItem(int s, sensorsPlot ∗plot)
Deletes a plot.
s - number of the plot curve to delete.
plot - plot to delete.

void help()
Associated with the help button. Shows the help.

void about()
Associated with the about button. Shows the about.

bool reallyOther()
On exit asks the user if he is sure he wants to exit.

C.6 serialComms class

serialComms(int argc, char ** argv, dataBase *d)

Constructor of the class.

d - the current database.

void logStateChanged(bool on)

Sends messages to the mobile station to stop or start sending data.

on - true to start sending data.

void run()

Thread that receives the messages from the mobile station. Translate the data and makes insert statement to insert in the database.

void on_commit (int signum)

Runs every second to insert data to the database.

void sair()

Checks that no messages are still to send.

C.7 VideoWidget class

VideoWidget(QWidget *parent)

Constructor of the class.

void changeMode(int index)

Changes between snapshot and video.

index - 0 for video, 1 for snapshot.

void changeStart()

Alternates between stopped and started.

void changeFrameRate(int index)

Changes the frame rate or periodic delay according to the selected mode.

void changeResolution(int index)

Changes to the selected resolution.

void changeSamplingRate(int index)

Changes the sampling rate of the audio.

void changeFrameRate()

Changes the frame rate or the periodic delay depending on the selected mode.

C.8 Video class

Video(QWidget *parent)

Constructor of the class.

void ChangeMode(int index)

Restarts mplayer to play the selected mode.

C.9 CarWidget class

CarWidget(QWidget *parent)

Constructor of the class.

void setAngle(double angle)

Sets the angle of the wheels.

void setTemp1(double temp)

Sets the temperature of the tyre. There is a function for each sensor. The numbering starts from the front right to the back left.

void setSuspension(double susp)

Sets the suspension displacement.There is a function for each sensor. The numbering starts from the front right to the back left.

void setBrake(double brake)

Sets the brake displacement.

void setAccelerator(double accelerator)

Sets the accelerator displacement.

void paintEvent(QPaintEvent * /* event */)

Responsible for drawing the car. Is called by the system.

void paintTyre1(QPainter &painter)

Paints the tyre. There is a function for each tyre. The numbering starts from the front right to the back left.

void paintSuspension1(QPainter &painter)

Paints the suspension. There is a function for each suspension. The numbering starts from the front right to the back left.

void paintAccelerator(QPainter &painter)

Paints the accelerator.

void paintBrake(QPainter &painter)

Paints the brake.

void paintCar(QPainter &painter)

Loads the image of the car.

QRect tyreRect()

Returns the shape of the tyres.