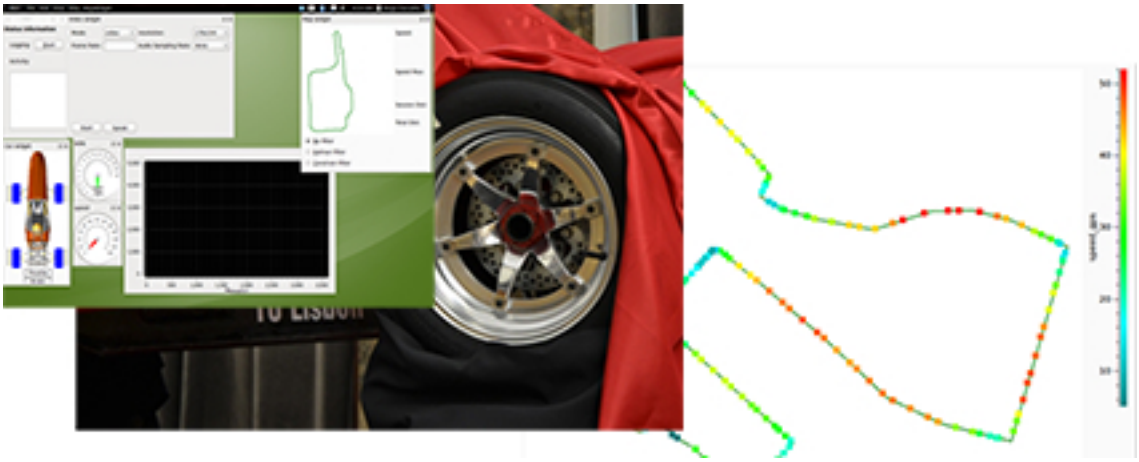
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



Formula Student Racing Championship: Design and implementation of an automatic localization and trajectory tracking system

Diogo Rafael Bento Carvalho

Dissertation submitted to obtain the Master Degree in

Information Systems and Computer Engineering

Jury

Chairman:	Prof. João Paulo Marques da Silva
Supervisor:	Prof. Nuno Filipe Valentim Roma
Co-Supervisor:	Prof. Moisés Simões Piedade
Member:	Prof. Alberto Ramos da Cunha

May 2012

Acknowledgments

Gostaria de agradecer a todas as pessoas que de forma directa ou indirecta contribuíram para este trabalho.

Aos elementos da equipa FST Técnico, e principalmente ao Pedro Oliveira pelos contributos através de requisitos propostos, pelo feedback que foram dando ao longo do desenvolvimento e pela ajuda durante os testes deste trabalho.

Ao Alexander Martins por me ter fornecido o trabalho desenvolvido por si juntamente com o Pedro Simplício, o Nuno Santos e professor José Sanguino, a quem também agradeço.

Ao Paulo Mendes pela disponibilidade para tirar dúvidas e para ajudar na integração com o seu trabalho.

Aos professores Nuno Roma e Moisés Piedade pela orientação e pela oportunidade que me deram de trabalhar nesta tese.

Ao Daniel Santana e à Inês Correia pelo apoio que me deram ao longo do trabalho.

À minha família que sempre me apoiou ao longo destes anos de faculdade.

Abstract

The aim of the IST Formula Student Project consists in designing and implementing a low budget car with limited resources, to compete with similar cars from other universities.

The objective of the presented project is to design and implement an automatic location and trajectory tracking system to equip the car and understand the relation between the position of the car and the values acquired from the other sensors. The position obtained by the Global Positioning System (GPS) will be adjusted to the track and it will be represented in the pit-stop screen.

The aim of the system is divided in two main targets: georeferencing the car's data that is: collected by several sensors, and tracking the car and represent its position in the circuit drawn in the pit-stop screen. Besides these two objectives, the system can be divided in two parts: one that consists in a GPS module connected to the communication system located at the car; and another one in the base station (at the pit-stops) where the data will be processed, analyzed and represented.

Keywords

Formula Student, Tracking, Georeferencing, Map inferencing, Map Constraining.

Resumo

O objectivo do projecto Formula Student Técnico consiste em desenhar e implementar um carro de competição, usando recursos limitados para competir com carros semelhantes de outras universidades. Este projecto tem como objectivo desenhar e implementar um sistema de localização e trajectória que irá equipar o carro de modo a perceber a relação entre a posição do carro e os valores adquiridos pelos sensores do veiculo. A posição obtida pelo GPS irá ser corrigida e ajustada ao mapa, e irá ser representada no ecrã do computador localizado nas boxes, por exemplo. Os propósitos do sistema estão divididos em dois objectivos: georeferenciar os dados dos sensores do carro, e representar em tempo real a posição do carro na pista em tempo real nos computadores das boxes.

Para além destes dois objectivos, o sistema pode ser dividido em duas partes físicas: uma consiste no módulo GPS localizado no carro que irá enviar a posição através do sistema de comunicação existente; outra que consiste na estação base (nas boxes), onde os dados podem se processados, analisados e representados.

Palavras Chave

Formula Student, Rastreamento, Georeferenciação, Desenho automático do Mapa, Restrição ao Mapa.

Contents

List of Figures

List of Tables

List of Acronyms

ACL	Advanced Central Logger
API	Application Programming Interface
C/A	Coarse/Acquisition
CAN	Controller Area Network
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DGPS	Differential Global Position System
ECEF	Earth-Centred Earth-Fixed
ECU	Electronic Control Unit
EGNOS	European Geostationary Navigation Overlay Service
ENU	East, North and Up
FST	Formula Student Team
GPS	Global Positioning System
INS	Inertial Navigation Systems
IMU	Inertial Measurement Unit
ISO	International Organization for Standardization
IST	Instituto Superior Técnico
KML	Keyhole Markup Language
LLA	Latitude, Longitude and Altitude
MCC	MATLAB C Compiler
PDF	Portable Document Format
RTK	Real Time Kinematics
SQL	Structured Query Language

TCP	Transmission Control Protocol
USB	Universal Serial Bus
UUKF	Unconstrained Unscented Kalman Filter
XML	eXtensible Markup Language
WAAS	Wide Area Augmentation System
WGS 84	World Geodetic System 84

1

Introduction

Contents

1.1 Context

The Formula Student Championship was created in 1998 in the United Kingdom and consists of a car competition very similar to Formula 1. The main difference is that this project is created and managed by student teams from many universities all over the world, with some strict restrictions like low budget and other limited resources [7].

Figure 1.1: 4th Formula Student Team Prototype developed at IST (FST-4e) [2]

The Formula Student Team (FST) is one of the existing teams in Portugal, composed by students from several departments of Instituto Superior Técnico (IST). In figure 1.1 it is represented the last of the 4 prototypes already built by the students, called FST 4e and the first car of an electrical era [2]. This car will be equipped with a full featured telemetry system, based in a mini-PC that collects data from several sensors such as: suspension displacement, engine RPM, tire pressure, battery level etc. This data is collected using a Controller Area Network (CAN) bus that interconnects the sensors and the car's station. Such data is then sent to a base station, installed at the pits, using the IEEE 802.11 (WiFi) protocol [6].

1.2 Motivation

In order to enhance the existing system and to support the processing of the acquired data it was decided to implement a georeference system that relates the information acquired by the car's sensors to the position, on the track, where that points were collected. Besides that, it was also required to implement an automatic tracking system that represents the actual and previous position of the car in the circuit map. This data will be represented at the team base station.

Hence, the presented work represents an integrated solution to implement the required georeference and tracking systems in the FST car and in its already existing platform. The solution implies the integration of specific processing blocks in both stations. In the car station, it was installed a Global Positioning System (GPS) device controlled by a software module that is responsible for receiving the location data and for transmitting it to the base station, by using the

existing communication platform. In the base station, the data acquired in the car sensors and in the GPS module is received and processed, in order to reduce eventual measurement errors and to relate it with other data; only then it will be stored in a database and represented in the team's screen.

This system provides a significant added-value to the data gathered by the car sensors, by relating this data to the car's position. This will allow the team developers to detect the origin of failures, by relating the data values with the track position, or to detect where those failures have started, facilitating the introduction of consequent improvements in the car using this information. As an example, it will be possible to know that the right front suspension starts to fail after a certain hairpin turn.

1.3 Objectives

This project represents the evolution of an already existing telemetry system in order to provide, in real-time, information about the car components state to the pit-stop team. Besides being important to the team, this information can even become more valuable if it is related with the position of the car in the track. By relating the values that are read by the car sensors with the position of the car, it allows the team to detect failures in the car components, detect in which track segment that failure occurred and improve the car performances based on the information acquired by the sensors.

As such, the main objectives of this project can be expressed as:

1. Create a real-time trajectory and tracking system, that provides the position of the car to the pit-stop team, with good accuracy.
2. Create a georeference system that relates the data from the car sensors with the place where the data were collected.
3. Integrate the information concerning to the position of the car in the base station screen, together with the existing widgets that represent the other sensors.
4. Integrate this solution with the existing telemetry and communication platforms, already existing in the car.

1.4 Main contributions

In this section it will be presented the main contributions of this work to the telemetry project, as well as the functionalities of this work that are important to the final user.

1.4.1 Create a Map

To represent the position of the vehicle, it is necessary to have a location context that allow the user to relate the information that he sees in the screen with the real world. In this case, the useful position information is not actually in which city or street the car is, but in which part of the

circuit a particular event occurred. Hence, the location context that is relevant in this project is the circuit.

To create this relation with the real world, the interface will represent a draw of the current circuit and mark the car position in this draw. Since the car can run in different circuits with different shapes, it is necessary to have (or to generate) those maps when needed. The following methods are considered in this solution:

1. Load from a Comma-Separated Values (CSV) or Keyhole Markup Language (KML) file
2. Generate in real time
3. Load the map from the database.

The first option, allows the user to load a CSV or KML file with a set of coordinates, representing the circuit map. The CSV corresponds to a comma separated values file, while the KML corresponds to a eXtensible Markup Language (XML) file from Google Earth application. This data contains a set of Latitude, Longitude and Altitude (LLA) coordinates, which will be processed through the application processing modules and the map will be represented in the interface.

Secondly, the map can be automatically generated from the coordinates received by the car. The car can run along the circuit, and the data received from the GPS will be processed to generate and draw the circuit in the screen, just like the first option.

Finally, after being processed all of this data can be stored in the database. Hence, the last option corresponds to load this data set, with already processed instead of raw data, skipping the generation process.

1.4.2 Live Session

The Live Session mode is enabled after the map is loaded. This mode should be used during the race and represents, in the computer dashboard, the real-time position of the car integrated with the representation of the sensor's gauges and graphs.

This mode represents in the corresponding widget the circuit draw, the car position (represented by a dot), and other information provided or calculated by the GPS. Along with the position information, this solution also has:

- instant speed
- maximum speed (all time/ session)
- distance (all time/ session

In order to improve the accuracy of the positioning, some coordinate filters are available to the user. Other information regarding the GPS speed and altitude were added to the set of sensors available in the vehicle.

1.4.3 Offline Session - Post Analysis

The Offline Session provides a set of tools to georeference the information from the sensors to the circuit map. This allows discovering some patterns between the car behavior and a section of the circuit.

The data will be presented in a set of circuit maps, corresponding to each lap, and the user will choose the sensor(s) that he wants to relate with the map. The application will apply a color scale corresponding to the sensor values, and will color the spots corresponding to the position with the respective color. This color-mapped circuit makes it easy to visually detect the relation between values and the car's position, and detect where some irregularity occurred.

In order to allow the analysis of the information out of this application, this solution allows to create a Portable Document Format (PDF) file with the information of the Offline session. The user will be able to select which sensors and laps he wants to analyze, and the application will create a PDF with this information for further print or distribution through email.

Besides the PDF report, the acquired positioning data can be exported to a KML file in order to be integrated in the Google Earth application, allowing the user to georeference the acquired data using satellite images.

1.5 Dissertation outline

The content of these documents is briefly described in the following:

- **Chapter 2 - Related work:** In this chapter will be presented other both academical and industrial works which are in some way related to the present work.
- **Chapter 3 - Proposed System Architecture:** The high level description of the proposed solution is described in this section.
- **Chapter 4 - Supporting Platforms:** In this chapter are present the description of the already existing hardware, software frameworks and algorithms which are integrated in this work.
- **Chapter 5 - System Implementation:** This chapter refers to the description of the implementation of all components present in this solution.
- **Chapter 6 - Results:** Presents the development steps, tests and results done in order to guarantee that the objectives are accomplished.
- **Chapter 7 - Conclusions and Future Work:** It will be analyzed in this chapter the results of the test and will be proposed future works in order to improve the telemetry system.

2

Related Work

Contents

2.1 Summary

Besides providing some insight on the supporting georeferencing technologies, this chapter presents a set of academic and commercial prototypes that are related with this project in different ways. The most similar work described here is the Racelogic VBOX [5]. This system provides data-log and GPS tracking with great precision. Other presented product with similar behavior is used for anti-carjacking and fleet management and also allows to track vehicles at distance with a reasonable accuracy. Finally, there are also other works related with improving the precision of the GPS and about the representation of the position in a road map.

2.2 Supporting Technologies

In this project, it will be used a low-cost GPS module that will be connected to the car's embedded board. The description of the GPS technology and its enhancement systems will be presented in this section.

2.2.1 Coordinate Systems

Latitude, Longitude and Altitude The standard representation of the coordinates used in the GPS devices is the LLA coordinate system. This coordinates are composed by latitude, longitude and altitude. The latitude and longitude are represented in degrees, where the latitude is the angle which ranges from 0^o at the equator to +/-90^o in the north/south poles and the longitude referrers to the angle which ranges from 0^o at the Greenwich meridian to +180 eastward and 180 westward. The altitude is measured as the distance above the level of the water and is measured in meters [8].

Earth-Centered, Earth-Fixed The Earth-Centered, Earth-Fixed, consists in a Cartesian coordinate system and it represents positions as an X, Y, and Z coordinates. The point (0,0,0) is defined as the center of mass of the Earth, hence the name Earth-Centered, being the distance to this origin measured in meters.

The z-axis is pointing to the North but it does not coincide exactly with the instantaneous Earth rotational axis. The x-axis intersects the origin of the previously addressed LLA system, at 0'0'. [8].

East, North and Up The East, North and Up (ENU) coordinates represents a local Cartesian coordinate system. The difference from the Earth-Centred Earth-Fixed (ECEF) is concerning about the origin of the referential which in the case of the ECEF is located at the center of the Earth. In the ENU coordinate system, the origin of the referential is locally placed, providing relative positioning of the coordinates instead of an absolute positioning on the earth. The origin of the map could be located for instance in the center of one map. In order to convert the WGS84 Datum (LLA) to ENU, the ECEF Cartesian coordinate system can be used as an intermediate step. [8].

2.2.2 Global Positioning System (GPS)

GPS it a global navigation satellite system established in 1973 by the United States Government for military purposes. The system consists in 24 satellites orbiting the Earth, several control and monitoring stations on Earth, and the receivers owned by the users. Its main aim is to provide timing and location services.

The system became fully operational in 1995, by offering two signals for different purposes: the military signal, consisting on an encrypted signal to be used by the US army with the best accuracy and reliability; and the civil signal, originally with an additional error induced by the US government to decrease the precision to 100 meters [9].

Currently, this system is widely used in military purposes, such as: controlling intercontinental missiles, orientation, tracking troops; and civil purposes as well, such as: car navigation systems, agriculture, wild life tracking, aeronautics, etc.

Nowadays, the whole system is composed by 24 satellites, one master control station and an alternate control station, four dedicated antennas and six dedicated monitor stations. These monitoring and control stations maintain the satellites in their proper orbits, and adjust the satellite clocks.

The timing service is implemented by incorporating in each GPS satellite a high accuracy atomic clock. The satellites permanently broadcast their own time (at the speed of light) to the receiver, so they can synchronize themselves with high precision.

Besides the information about the time of each satellite, the satellites also broadcast their current position.

With the information about of which time the message was sent and the speed (speed of light), it is possible for the receiver calculate the distance between him and the satellites. Knowing the position of the satellites, which is sent in the message, and calculating the distance between the receiver and the satellite, it is possible for the receiver to calculate his own position. The figure 2.1 is exemplified the triangulation of the satellites.

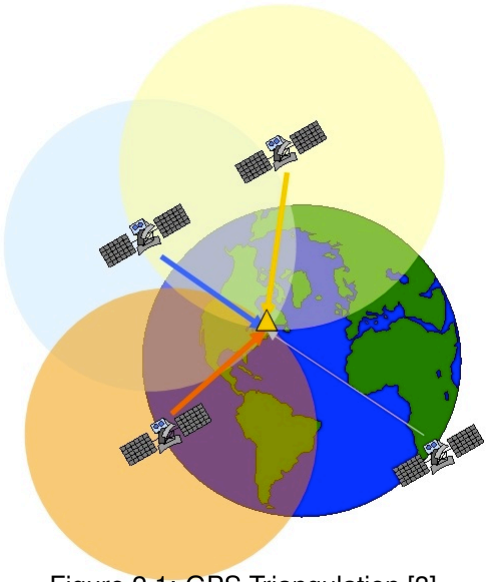


Figure 2.1: GPS Triangulation [3].

To provide accurate location data in tree dimensions (latitude, longitude, altitude), this system requires the simultaneous clear vision operation with a minimum of 4 satellites [10].

Until May, 2000 the US government added an additional error called *Selective Availability,* in order to decrease the precision of the civil GPS receivers to 100 meters. Today this service is turned off, but there are still other external aspects that impose a decrease of the GPS signal precision, such as [10] [11]:

- **Multipath** - Consisting on reflections of the GPS signal in a building, for instance. It is very common in city centers, leading to a determination of a fake position.

- **Atmospheric delays** - Dry air, water vapor, hydrometeors and other particles induce delays in the GPS signal, which decrease the accuracy of the location estimation, since the calculation of the position is based on the time that the signal takes to travel from satellite to the receiver.

In sections 2.2.3 and 2.2.4, it will be described two current techniques to solve these problems, by using two GPS modules to detect and correct the errors. Other systems to improve the accuracy of the GPS, are the North American Wide Area Augmentation System (WAAS) and European European Geostationary Navigation Overlay Service (EGNOS) system. This systems consisted in a set of ground stations which monitors and measures the GPS signal, in order to detect deviations from the correct positioning. This signal error information will be send to the geostationary satellites it order to broadcast the correction message to all the GPS receivers compatible with this technology. The receivers compatible with this technology can provide accuracy between 2 and 5 meters [12].

2.2.3 Differential GPS (DGPS)

Differential Global Position System (DGPS) is a more precise solution based on two GPS terminals, one high-quality GPS receiver with an antenna at a known location, and a roving receiver, located for instance in a car or boat (see figure 2.2). It also assumes a communication medium between the stations. These stations need to have at least 4 common satellites in view, and must be within a distance of 100 km.

The reference station estimates the error by comparing the location information received from the GPS satellites and its real and known position, and assuming that the errors due to atmospheric effects (e.g. ionosphere, troposphere, etc.) are similar for both stations. These corrections are then transmitted to the rover receiver, which will apply it to its own estimate. This correction can decrease the error to 2-8 meters. Nevertheless, this techniques does not provide any solution for the multipath effect [13] [14].

2.2.4 Real Time Kinematic (RTK)

Real Time Kinematics (RTK) technique is similar to the DGPS, in the sense that it also uses both receivers and a communication medium. However besides the Coarse/Acquisition (C/A) code, it also uses the carrier phase for position calculation. The user antenna needs to be within

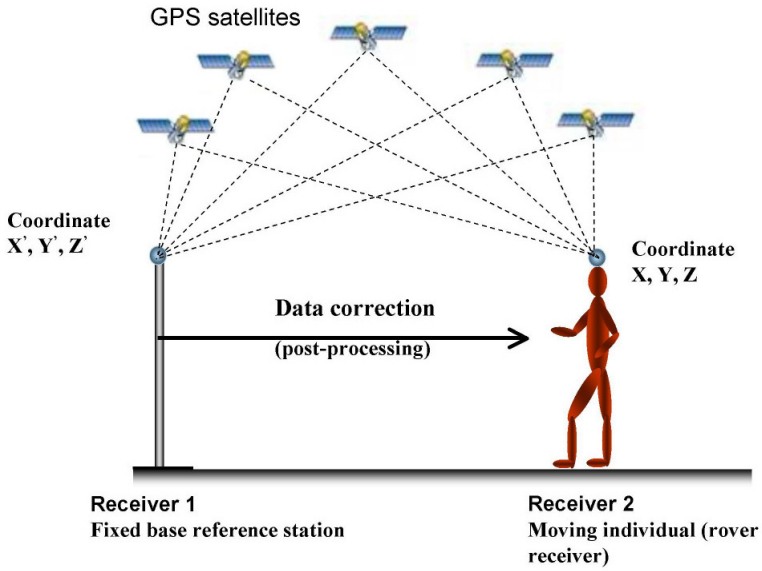


Figure 2.2: DGPS [4].

a distance of 10 km to the base station, and must have a real time radio link in order to transmit the correction of the actual position.

RTK offers two types of solutions: float and fixed. The first one needs at least 4 common satellites (like the DGPS) and has an accuracy between 20 cm and 1m. The RTK fixed solution needs one more satellite than the float solution, and offers accuracy within 2cm. Both solutions need about one minute of initialization time to give their maximum precision [14] [5].

2.3 Academic Georeference Prototypes

In this subsection it is presented two academic works that have the purpose of filtering the data acquired from the GPS and reduce the error using other information gathered from external sources, such as Digital Map and/or other sensors.

2.3.1 Uncoupled GPS Road Constrained Positioning Based On Constrained Kalman Filtering

As it was previously referred, the data points received from the GPS receiver may be less accurate than what would be desired. As it was also referred there are some external conditions that may introduce some error in the determination of the position, such as atmospheric and multipath effects that induce extra delays in the GPS.

To reduce that error and to determine the most probable position of the receiver on a given road, this algorithm makes use of a Kalman Filter (to reduce the noisy points) and of a previously generated map to constrain the points to a predefined track [11]. This algorithm is particularly addressed to low cost receivers and to platforms with reduced computational capabilities.

The Kalman Filter takes as input the noisy coordinates from the GPS. Then, it predicts the location based on past positions and corrects the estimated value based on the most recent location [15]. The resulting values have less noise and tend to be closer to the real location. Then,

the constrain function takes that points and restricts them to the nearest point of a predefined map. The final step will project the obtained coordinates in the road map, thus reducing the initial error margin. The map is generated from a linear or quadratic regression, obtained from a set of very accurate and previously traced points [11].

2.3.2 Constrained Unscented Kalman Filter Based Fusion of GPS/INS/Digital Map for Vehicle Localization

This alternative proposal presents a solution to multipath and signal loss (e.g. in a tunnel) problems with GPS [16]. Besides using the data obtained from a DGPS device, it also uses the data from Inertial Navigation Systems (INS) and road geometry from Digital Map. To increase the accuracy and to combine these different sources of data, the solution makes use of a Constrained Unscented Kalman Filter, to constrain the estimates to a predefined road that can be obtained from a Digital Map Database.

The INS contains an Inertial Measurement Unit (IMU), which is composed by a gyroscope and an accelerometer. It provides the values of the yaw rate and forward acceleration, which will be used to create the Dynamic INS equations. These equations will be the state equations of the Unconstrained Unscented Kalman Filter (UUKF) algorithm, when the coordinate estimations from DGPS will be part of the measurement equations. The result of the algorithm, applied to this loosely coupled DGPS/INS system estimates, will be finally projected into the state constraints provided by a Digital Map database [16].

2.4 Industrial/Commercial products

Nowadays, the potential of GPS is being explored by several companies in different areas. Some of these commercial products are presented in this section and are somehow related with the developed work. The VBox [5] and Anti-carjacking [17] systems are related by consisting in a tracking and trajectory system, while the GPS Navigators [18] [19] projects the position of the car in a existing map.

2.4.1 RaceLogic VBox

Racelogic is a UK company founded in 1992 that develops electronic control and measurement systems to the automotive testing and motor-sports markets. One of their popular products is VBox [5]. In its simplest version, it consists of a GPS and a CAN data logger, while the high-end version provides a multi-camera video and a real time graphic overlay. The VBOX GPS data logging system, represented in figure 2.3, is placed in the car and can acquire data at 100Hz and send it, in real-time, to a USB or bluetooth device (like a PC) or even send it through the CAN bus, just like other car sensors. It can also be combined with an IMU that can provide better measurements in poor visibility conditions, and with a RTK [1] station to provide more accuracy.

The IMU consists in three accelerometers and three gyroscopes, that can measure accelerations and rotations in the x, y, z axis. The objective is to provide better measurements in conditions

[1] Described in section 2.2.4

Figure 2.3: RaceLogic VBox [5].

with low or no visibility to the satellites (like in a tunnel or under a bridge), using a real-time Kalman Filter. As it was referred in section 2.2.4, the RTK unit is a station composed by a second GPS receiver with a very known location. This auxiliary station gets its GPS coordinate, compares with its real location and corrects the position acquired by the car's GPS module. With this technique, smoother velocity traces are achieved and the accuracy of the car position increases to 2cm [5].

2.4.2 Anti-Carjacking System/ Fleet Management

These systems are often used by fleet managers to know, in real-time, where a particular car is. This application is also used by car owners to recover the car after a robbery, by using this service to obtain its position.

The system provides the location at distance, i.e. it is possible to be at home, or in the fleet management office, and watch in the computer map where a particular car is at any time. It is also possible to track where the car had been before, relate with its speed, engine sensors and other information.

The technology behind this system is a GPS module, that locates the car in the globe, and a communications unit based on a GPRS module (or similar technology), located in the same device, that sends the GPS coordinates and other data related to car components to a data center. The user can then receive and see the data through the internet or the mobile phone (via SMS), either in the form of GPS coordinates, or through a computer map/satellite image, where it is represented the car's location and other related information, such as the car sensors. History data can also be remotely obtained [17].

2.4.3 GPS Navigators

In the last decade, several navigation companies, like TomTom or Garmin, have become increasingly popular for providing GPS Navigation solutions. Besides wide-open and rural areas, these products may also be used in cities, which have a large number of buildings that increase the multipath error.

Since these products are highly exposed to the risks of multipath effect and other distortions that decrease the quality of the GPS signal and the precision of the computed estimative of the current position, the device software makes use of digital maps not only to represent the roads, but also to constrain the car's position to the nearest road, thus avoiding the representation of the car in wrong places [18] [19].

2.4.4 MoTeC

MoTeC is a US company founded in 1987, dedicated to the manufacture of data acquisition and engine management systems. It is a leading company, with clients in the military, commercial and competition industry, with a great set of those clients participating in competitions like Le Mans, NASCAR, Dakar, World SuperBikes, among others [20]. In the context of this thesis, the most relevant MoTeC product's are: the i2 and i2 Pro data analysis tool, the MoTeC telemetry system, the GPS modules and specially the components of these systems that provide the car location service and relation between the car's sensor information and the cars position, where that data was acquired.

i2 and i2 Pro software tool for data analysis

This software framework is divided in two levels, the i2 free software and the i2 Pro version. The i2 Standard is a software tool freely available to the customers with data logged from the MoTeC Data Logger product or a Electronic Control Unit (ECU) from the same manufacturer. The i2 Pro is a more advanced software tool that provides mathematics, multiple overlays and a paid license that allows the software to analyze data from other types of source and file formats.

Figure 2.4: MoTeC i2 graphical interface.

Some commercial video-games or simulators (like GTR-2 [21]) can also save data in the same file format as MoTeC Data Logger for subsequent analysis, and with an other extra license it is possible to use the MoTeC's Application Programming Interface (API) to create a file format compatible with the software. This allows the user to import data from other type of files and sources to i2 Pro for data analysis [20].

The i2 Pro software allows the representation of the data logged in a file, gathered from a multiple kind of sensors present in the car, in a way that can be organized, interpreted and manipulated by the user. To achieve this goal, i2 Pro allows the user to manage the data using math plugins and an equation editor, as well as to convert data, and choose one or more representations to any sensor data from a set of components that can include:

- Time/Distance Graphs;
- Scatter Plots;
- Histograms;
- Suspension Histograms;
- Frequency (FFT) Plots;

- Mixture Map (Lambda);
- Track Reports;
- Channel Reports;
- Section Time Reports;
- Synchronized Video;
- A variety of telemetry style gauges.

The software tool can be used to perform calculations and to organize and chart data in order to identify trends, problems and improvements. This might involve reviewing overlaid data, creating track maps and comparing graphs and gauges [20].

MoTeC Telemetry Monitor

The Telemetry Monitor system is a similar software tool that presents to the user located in the pit stop (or other fixed location) real time information about the vehicle through a set of widgets. Although most of this software is not concerned with data logging, it also allows the user to record the data gathered by the car sensors. The data is collected by the data acquisition system located in the car and connected by a CAN bus to an Advanced Central Logger (ACL) module, that sends the data to a remote computer through a Ethernet connection. This allows the data engineer to monitor the engine and chassis data in real time, while the vehicle is still on track [20].

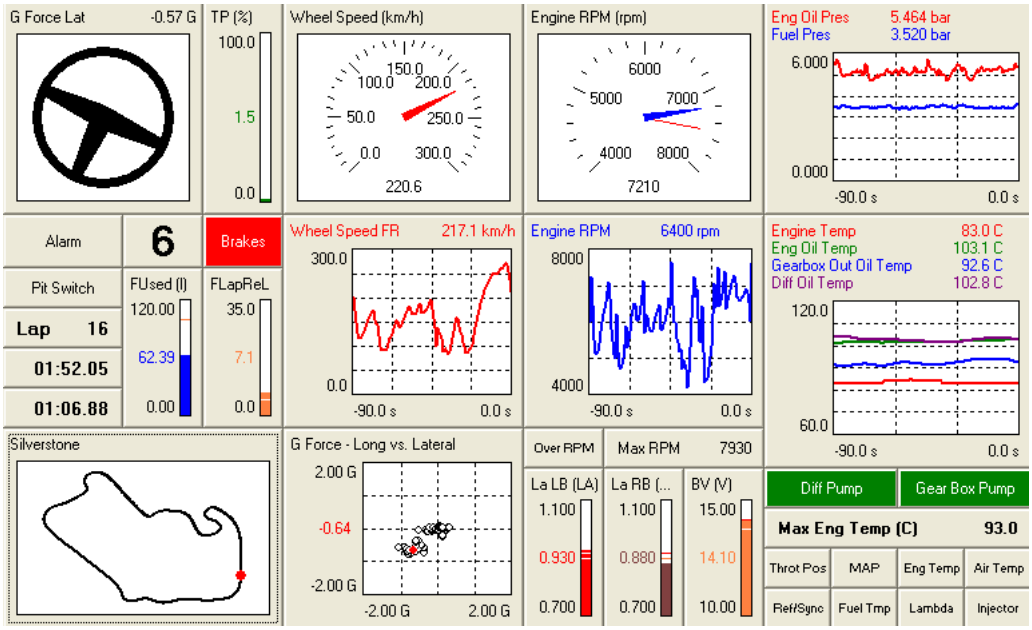


Figure 2.5: MoTeC Telemetry Monitor.

The data received by the Telemetry Monitor software is presented in real time by dial bar graphs, virtual steering wheels, track maps and several warnings in a similar way as in the i2 software (see figure 2.5). This allows the engineer to detect early signs of problems and warn the driver or prepare for a set up adjustments or repairs during the next pit stop.

MoTeC Track Map

One of the MoTeC's components in the former i2 Software (the MoTeC interpreter) is the Track Map, that consists in a graphical representation of a circuit, upon which position data recorded during the race is displayed in a easy way (see figure 2.6). This track Map also allows to determine what is happening with the car in every point on the circuit, by providing a set of reports for different analysis of the data collected in a session [22].

An example of the Track Map rainbow report is illustrated in fig 2.6 . This example corresponds to a lap of Bathurst circuit with several gathered signals, such as throttle usage, gear change points, maximum and minimum speeds and battery voltage displayed. The values of the sensors are represented by a gradient of colors [22].

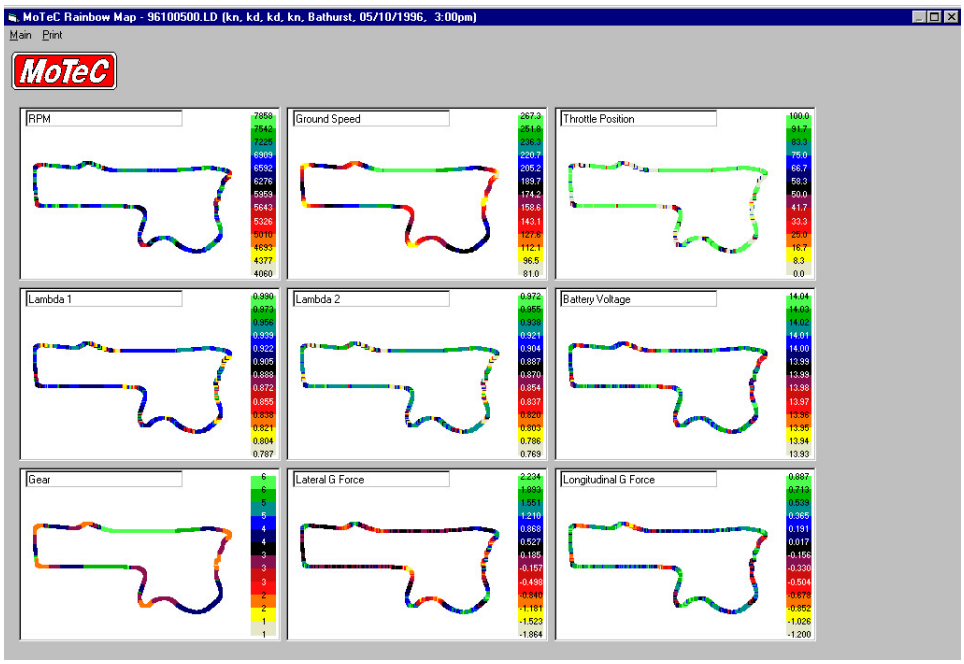


Figure 2.6: MoTeC Interpreter Report screen.

The plotted track map is not the exact shape of the real circuit, but a representation of the line that was driven around the track. The map can be generated either with the data gathered at the car while driving, or from an existing Track Map. If an existing track map is used to display new data, to represent the position, the only requirement is that the vehicle speed exists in the log file [22].

The information used to generate the map using the car sensor date, consists in Speed, Lateral G force and Lap Beacon to detect the laps. The accuracy of the generated map depends of the quality of the collected data by the car, and the calibration of the Lateral G force or Speed sensors.

To improve the accuracy of the calculations, a Longitudinal G force sensor can be used to correct the speed by eliminating the effect of wheel locking and/or lifting. If Longitudinal G data is not available, it is important to choose a lap with minimum to generate the track map.

3

Proposed System Architecture

Contents

3.1 System Architecture

Despite the great accuracy offered by the DGPS and RTK, they also have a high cost associated with the price of the equipment and with their implementation. To workaround these costs, the solution that will be now proposed will use a low-cost GPS, with an accuracy of 5-10 meters. Furthermore, to provide a more precise result to this solution, the techniques described in section 2.3.1 and some features like GPS speedometer and odometer will be also integrated.

This chapter presents the architecture of the proposed solution, composed of the two units represented in figure 3.1 : the mobile station, installed in the car; and a base station, installed at the pits.

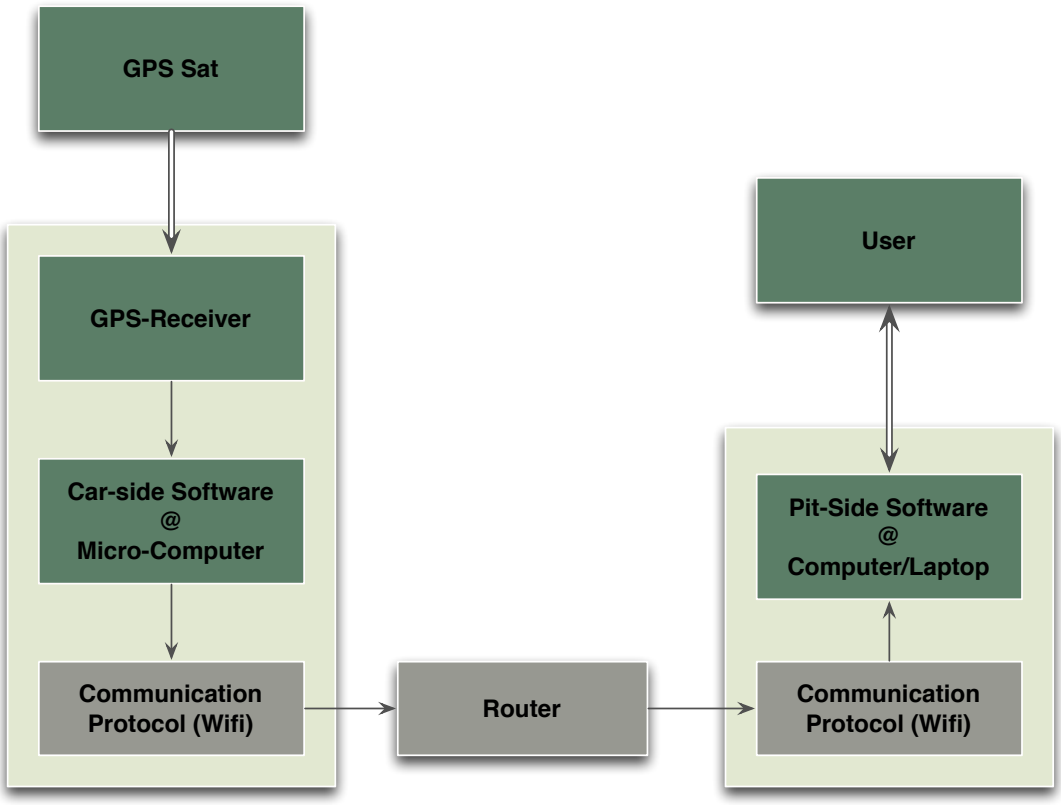


Figure 3.1: Simplified block diagram of the proposed architecture: Mobile, Base Station and Communication infrastructure.

The mobile station consists in a small computer placed in the car, connected to a GPS receiver, a wireless antenna, and to a CAN BUS module that receives sensor data from the car's CAN Bus and sends it to the computer Universal Serial Bus (USB) interface [6]. This mobile station runs with a Linux based operating system, a data gathering application, a WiFi communication framework and a local data logger.

The data gathered by the mobile station is sent by WiFi to the base station, using the data communication protocol implemented in [6]. When the sensor data is received by the base station, it is stored in the database for further analysis and represented in real time in the user interface through widgets like gauges, plots and maps.

This base station consists in a common PC or Laptop with Linux operating system, a database and the corresponding data logger, and an interface application.

3.2 Mobile Station

The mobile sub-system will be integrated with an already existent prototype [6] (described in section 4.2) and with the GPS module receiver, whose characteristics will be described in section 4.3.1. The acquired GPS coordinates and other data obtained by the GPS module, like time and speed, will be sent to the pit-stop by using the existing communication platform. This data will be properly timestamped in order to be correctly processed by the base station, which guarantees the temporal order of all the received data and makes a corresponding correlation with the other sensors data.

The solution should be as light and cheap as possible, since the car's station is limited in terms of computational resources and space, and the whole project is limited in terms of costs.

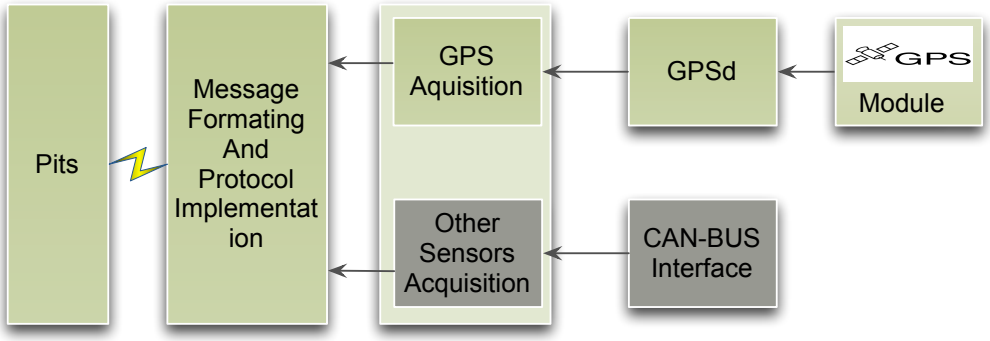


Figure 3.2: Mobile Station Architecture.

From the mobile station point of view, whose architecture is depicted in fig. 3.2, the system is composed by the following elements:

- **Pits** - Corresponds to the Base Station, where the data gathered in the car will be processed.
- **Message Formatting and Protocol Implementation** - The used platform to send and receive data messages to/from the Base Station.
- **GPS Acquisition** - Module that communicates with the GPS daemon and formats the messages that will be sent.
- **GPSd** - Daemon that makes the interface between the application and the GPS hardware.
- **GPS Module** - Hardware which assures the communication with the GPS Satellites, calculating the actual absolute position in the Earth.
- **CAN-BUS Interface** - Allows other sensors and hardware installed in the car to communicate with the mini-computer, by adapting the data received in the CAN Bus to an USB Interface, in order to be connected to the computer. Some examples of these sensors can be a speed or temperature sensor, an accelerometer, a battery level sensor or other hardware plugged to the CAN Bus.
- **Other Sensors Acquisition** - It is the software that receives the sensor data from the CAN

Bus interface, timestamps it and sends it to the base station through the communication protocol.

3.3 Base Station

The Base Station is the module that is responsible for receiving, processing, storing and representing the data collected at the car. In this case, the base station will receive the coordinates (and all other useful information like time and speed) from the car's GPS, log this data in the database for further offline analysis and represent it in real-time as a widget to the user. The coordinates received from the GPS may be presented directly, processed using the Kalman filter or constrained to a digital map using the algorithm described in section 2.3.1, according to the user option. The offline analysis can be made using a developed application used to navigate through the plot, which represents the circuit map and the values of the sensors, in order to visualize the evolution of the sensors values according to the position of the car. Other alternative way to analyze the information is by generating and printing a PDF report, with all the data gathered in a given race session.

To implement this solution, it was designed the following architecture represented in figure 3.3. It consists in seven new modules:

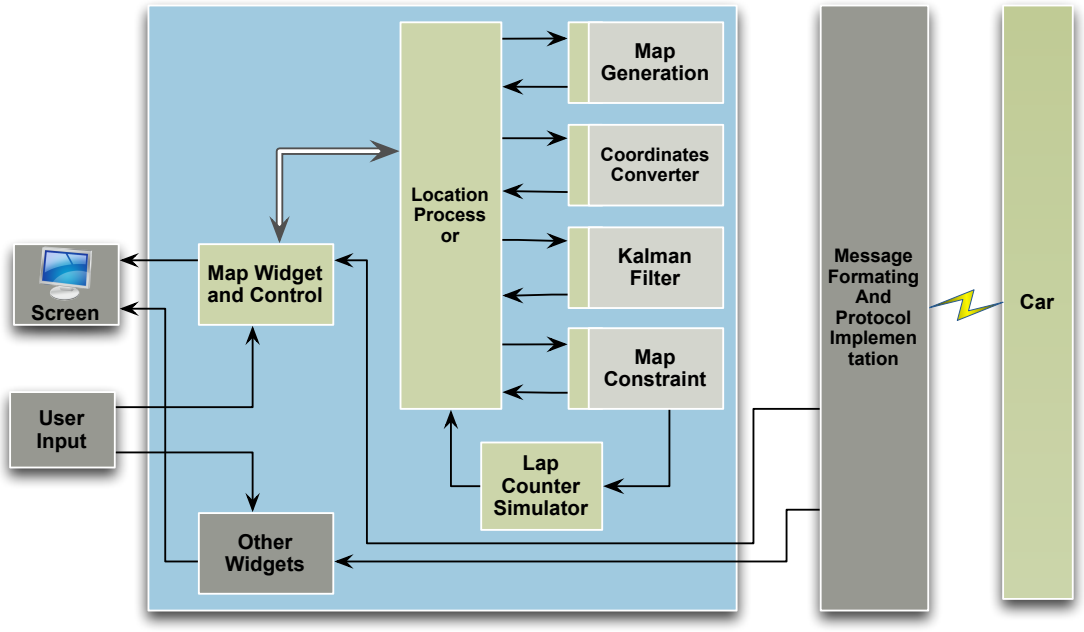


Figure 3.3: Base Station Architecture.

This architecture is composed by a group of modules responsible for processing the coordinates data, a module which simulates a lap detector and a counter sensor and a module responsible for all the interaction with the user (visual interface, user input, reports, etc.). A detailed description of each of these modules is provided in the following paragraphs:

- **Coordinates Conversion Module:** Since the coordinates received by the GPS are in LLA

units and the computer screen uses local screen coordinates, the coordinates values under processing must be converted.

This module converts the gathered global LLA coordinates to a local coordinates representation using meters ENU. Since the coordinates are in meters, this will allow the introduction of some features like a speedometer and a odometer, based on this GPS information.

The conversion from ENU coordinates to screen coordinates takes place in the Map Widget and Control Module.

The inverse operation is also supported, so that data can be exported and opened in an external application.

- **Map Constrain Module:** The location of the vehicle is represented at the draw of the current circuit by a circle corresponding to the car.

  Since the circuit map can be obtained either by an external tool or by the coordinates provided by the car, the final representation of these two layers (circuit and circle) may not coincide. A small error margin between the tool and the GPS car coordinates, or a different passage of the car during the map generation procedure, can cause the car to be represented outside the circuit.

  The aim of this module is to prevent this situation, by constraining the car position to the circuit, thus contributing to a better visual representation and understanding. If the current location of the car is in a place with a bad GPS reception (in the middle of buildings or mountains), the constraining of the coordinates to a previous defined map can reduce the error margin that occurs in these situations.

  This module will use information provided by the Kalman Filter Module and by the Map Generation Module.

- **Kalman Filter Module:** This filter is used to reduce the noise of the GPS samples, in order to provide better estimates. In this application, the filter is especially useful to design a smoother map of the track using the car GPS sensors, by removing some irregularities.

  Then, this module projects the estimated points into the track that was modeled by the Map Generation Module.

- **Map Generation Module:** This module is responsible to model the map of the circuit where the car will run by using a set of arithmetic functions.

  The module receives as input the circuit coordinates, and transforms then it into a set or arithmetic functions modeling the Map. These functions correspond to the digital map which will be used by the map constrain module to more accuracy represent the car location.

- **Map Widget and Control Module:** This is the interface module between the application and the user. This module draws and manages the user interface, receives input from the user and generates the PDF report.

  To support these functionalities, this module makes use of several widgets to represent the circuit map, the speed, the traveled distance and the current position. For the offline analysis

mode of the application, it also represents the sensors values for each coordinate allowing a correlation between position and car behavior.

This module will work in parallel with all other similar widgets, adding more information to the user application.

The Control part of this module provides the data management support, by receiving the coordinates data from the communication platform and dispatching it to the other modules responsible for the coordinate processing. It is also this module that is responsible for the conversion between the ENU coordinates and the screen coordinates, the managing of the window events and the user input. It is and also responsible for updating and reading data values from the database.

- **Location Processor Module:** This is one of the main processing modules. It is this module that contains the functions to transform raw coordinates into local filtered coordinates and maps.

  These functions are made available to the representation module by a set of interfaces. To implement those functionalities, this module uses and manages the converter module, the Kalman filter, the map generation and the constrain module.

  Depending on the user input gathered by the representation module, the Location Processor Module chooses the corresponding data flow between the modules: generate map, tracking car's position, etc. These dataflows are represented in the next pages, in figures: 3.6, 3.8, 3.9 and 3.7.

- **Lap Detector Module:**

  Since neither the car nor the circuits that are used to test the prototype have a Lap Detector device, and the information about the current lap is not only useful but often necessary to the application and to its users, the functionality of this module is to receive the GPS position and check, using the history data, if the car had already passed in the same place and if it corresponds to a new lap or not.

  With this information, the collected data can be categorized by session and lap, helping the organization of all the data and gives more information to the user. This is also helpful to the map generation module, which generates a unique lap corresponding to each circuit from a set of coordinates corresponding to distinct laps.

In figure 3.4 it is represented the interaction between the application modules and the database, corresponding the ”W” and ”R” values to the write and read operations:

The message protocol processes all the information received from the car-side system, information including the GPS data and the CAN Bus data. All the values which are processed by this message protocol are stored in the database in the proper table corresponding to their sensors.

The Map Widget and Control Module is the module that interacts more with the database. In a real-time session, this module can load a previous saved map from the database or store a recently generated map. It is also stored in the database, the processed coordinates along with the information of the current session (map used, distance travelled, etc). This information will be

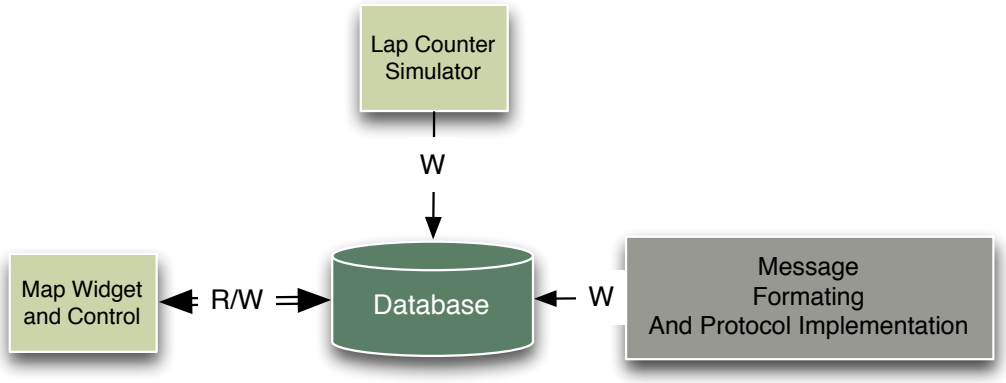


Figure 3.4: Application modules and Database Interaction.

subsequently be loaded in the Offline mode along with the GPS speed and other values gathered in the other sensors.

Also the result of the analysis made by the Lap Counter Simulator, that concludes at which time the vehicle start a new lap, will be stored in the database as a CAN Bus sensor.

3.3.1 Flow Description

Having briefly described the several elements that compose the proposed architecture, this section presents the main functionalities of the base station, as well as the data-flow that supports each functionality.

As presented before, this telemetry system allows the user to remotely observe the information originating from the car components in his computer in real-time. The gathered data is stored in the user's computer database, to be further analyzed with the help of other functionalities present in this system. Figure 3.5 represents the interaction between the user and the system, considering the two and the main operation modes: live session with real-time map generation and data-processing mode, and offline session responsible for the analysis and report generation.

In the beginning of the application, the user can chose if he wants to log data from the car (live session) or if he wants to analyze previously stored data (offline session).

For the Live session, the user first needs to choose how he wants to generate a map. Three choices are available:

- Generate as the car moves, using the coordinates gathered in real-time by the GPS;
- Import from a CSV or a KML file, containing a set of coordinates;
- Load a previous generated map from the Database.

After the map creation, the application will run the live session mode, presenting the acquired information in the widget in real-time. The Filter Mode allows the user to choose if the coordinates present in the screen are subject to any filtration or constraining.

If the offline Session mode was chosen, the user also needs to choose which session he wants to load from the database to analyze. A screen showing the map and each position of the car for each lap will appear, and the user has the opportunity to select, from a set of available sensors,

which one he wants to correlate with the map. In the following paragraphs, it will be described with mode detail the implementation of some of the main functionalities offered by the base station.

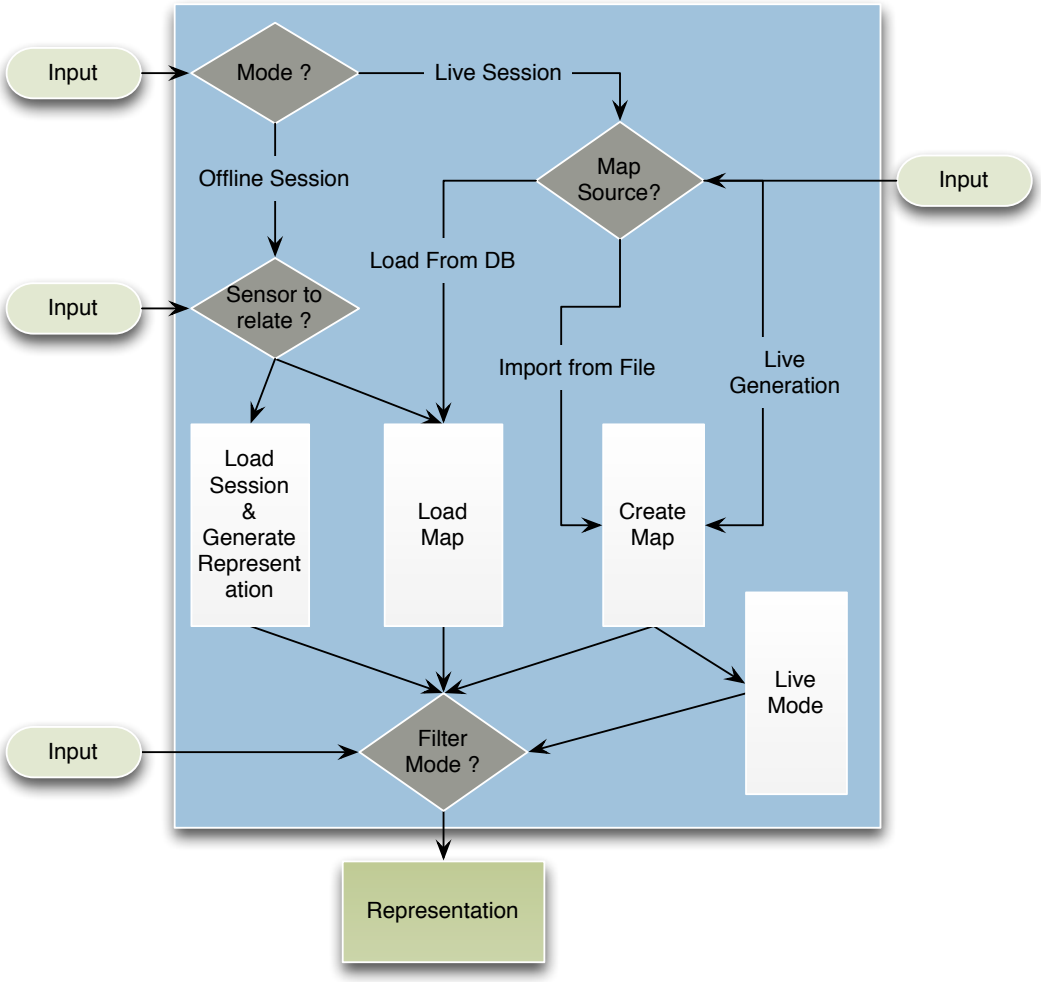


Figure 3.5: Main Processing Dataflow.

Create Map Mode The generation of the map is an non-trivial process that starts with the reception of the data by the application, and finishes in the user screen and in the database. To achieve such objective, the coordinates data must be processed, filtered and subjected to several other manipulations. The data-flow of the map generation is present in figure 3.6.

The map is generated from a set of points that composes the circuit. These points can be obtained from a previously defined map in a CSV or KML file, or directly modeled from the car trajectory.

The first case is more simple and accurate to implement. According to figure 3.6, the user simply chooses the file that contains the points (1). The Location Processor module after receiving the points from (2), opens the file and sends the points to the coordinate converter module (3). After the conversion (4), the coordinates are sent to the Kalman filter module (5) to be filtered. Then, the filtered data is sent to the Lap Detector (7) to check the coordinates and detect if the car

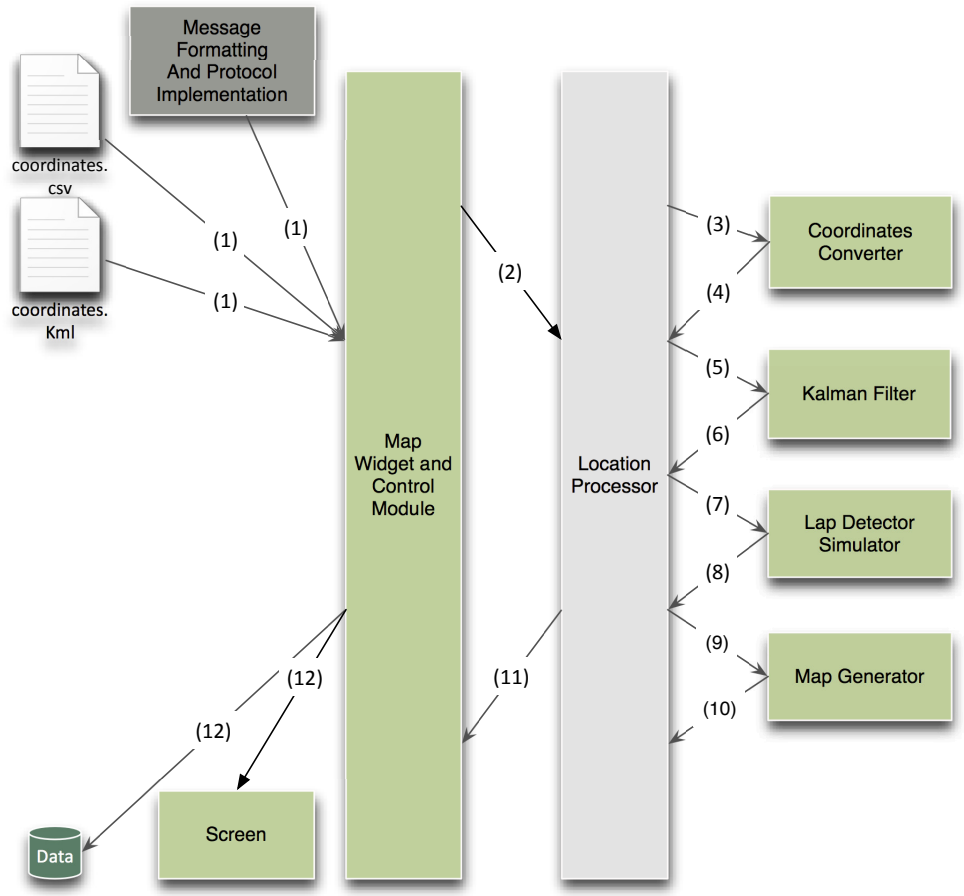


Figure 3.6: Map Generation Dataflow.

has crossed a fictional finish line. The filtered coordinates along with other Kalman Filter output, will be sent to the Map Generator module (9), which will return the map structure (10) that will stay in memory in the Location Processor Module. The processed map coordinates will be sent to the Widget and Control Module (11) that will store it in the database (12) and presented it in the screen.

In the second case, corresponding to the one where the map is directly modeled from the car trajectory in real-time, the user gives the start command and the Map Widget and Control Module starts receiving data from the Message Protocol (1). Once the reception of the coordinates is finished by users option, it starts generating the Map and the process will be the same as above.

Load Map After the map creation, the ENU coordinates corresponding to the map are stored in the database. These coordinates correspond to the raw key coordinates that delimits the map, by storing the ENU coordinates instead of LLA will allow to skip the conversion process and improve the performance. Saving the coordinates instead of the actual map structure reduces the performance, once that the map structure will be generated again, but on the other hand the original data is saved, because after the data has been filtered and processed it is not possible to

reverse this operations and get the original data back.

Figure 3.7 represents dataflow of this procedure. Firstly, the coordinates are loaded from the Database (1) and transferred to the Location Processor (2). The processor will then filter (3) and generate the map structure (5), just like in the creation step, returning to the Map Widget and Control Module to be presented in the screen (8).

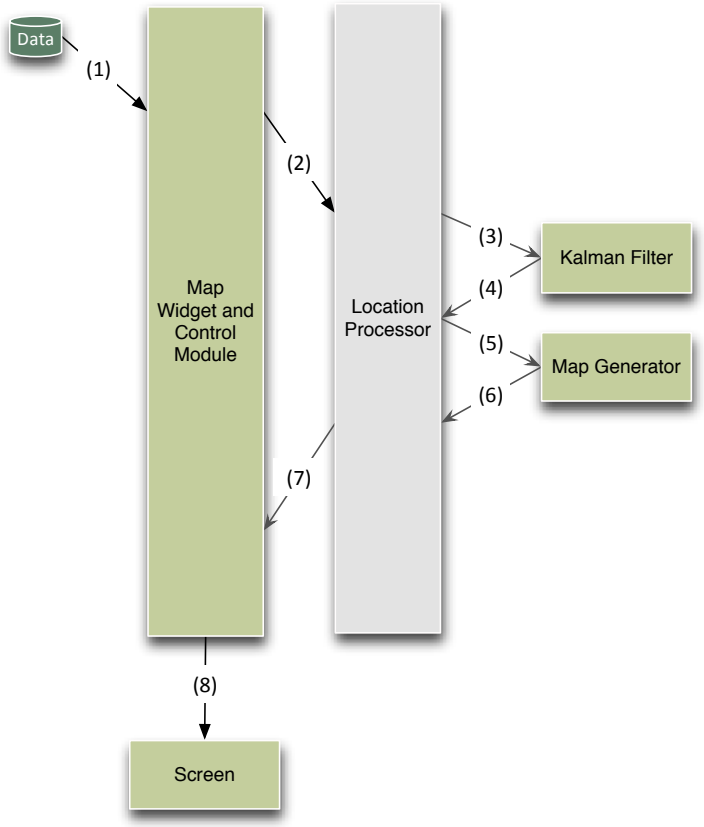


Figure 3.7: Load Map Dataflow.

Live Mode After the map generation, or map loading procedures, the live mode is activated and the system is ready to acquire coordinates. The processing of the data related to the live session is represented in figure 3.8 the main differences with the figure 3.6 is concerning about the last step of the Location Processor.

Firstly, the raw LLA coordinates, together with the speed and time measurements from the GPS are received from the Message Formatting and Protocol Implementation Module, to be persisted in the database (1), as is shown in figure 3.8. After being stored, the data corresponding to the location will be sent to the Map Control Module, while the other data will be sent to the respective processing modules (2). Then, the LLA Coordinates are sent to the Location Processor (3), which will orchestrates the coordinates processing by first sending them to be converted into ENU coordinates in the Coordinates Converter module (4). After the conversion, the ENU raw coordinates (and the previous position and velocity estimates) will be sent to Kalman Filter (6),

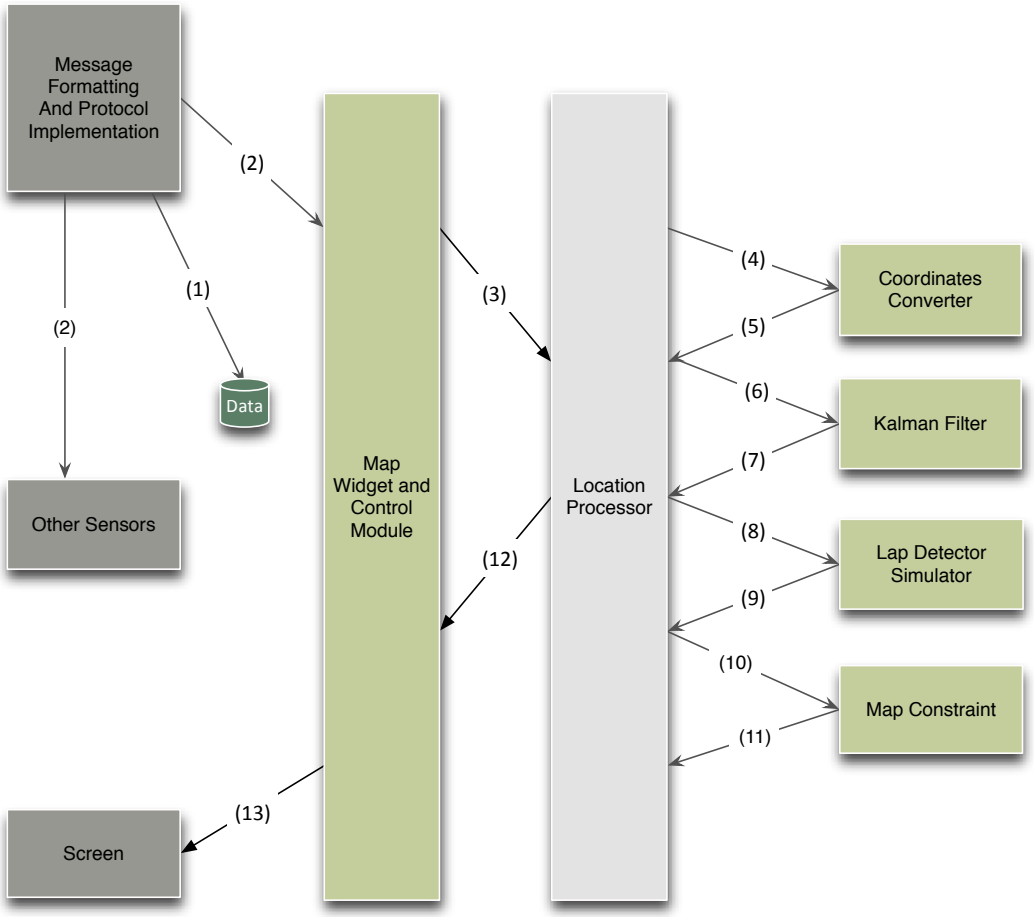


Figure 3.8: Live Mode Dataflow.

from where it returns the actual estimate position, speed and error.

The data received from the Kalman filter (7) will be sent to the Lap Detector Simulator (8) and then to the Map Constrain Module (10), together with the map structure generated in the Map Generation Mode. This module constrains the received points to the tracks of the map, and returns the corrected coordinates, already projected in the map (10).

The raw, filtered and projected coordinates will be sent to the Map Widget and Control Mode (12) where according to the user option, will be presented in the screen (13).

Load Session & Generate Representation Mode After having all data logged, the user may want to analyze all the stored information. The data acquired by the car in one session corresponds to a significant set of different sources (sensors) with each one supplying a great amount of data values. While the version of the processing prototype only provided the possibility to show this information using a graph plot, with this new prototype that is now proposed it is possible to relate this information with the position of the car in the track.

In the figure 3.9 is identified the steps to accomplish this task:

The gathered LLA coordinates are loaded from the database and sent to the Map Control

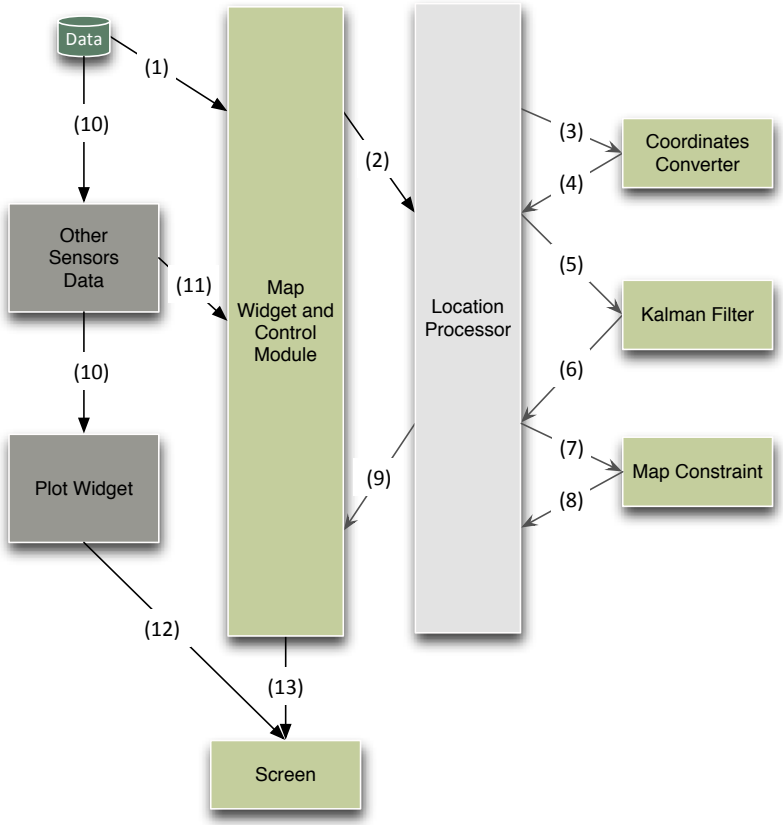


Figure 3.9: Load Session Dataflow.

Module (1). There they are converted (3), filtered (4) and constrained, just like in the previous modes. In parallel, it takes place the loading and representation of the other sensors (8). These are first loaded from the database and represented in a plot (10), and sent to the screen (12).

The correlation between the sensors information and the position of the car takes place when the coordinates are already converted (9) and the values of the sensors loaded from the database (11).

The Map Widget and Control module will relate the projected coordinates with the other sensors data (11) finding the mutual (or approximate) timestamps, and will present the map and the position corresponding to each sensor state in the screen (13).

4

Supporting Platforms

Contents

4.1 Summary

In this chapter, it will be presented some platforms and processing modules that will constitute the implemented system. These platforms consists in the telemetry hardware and software, used in the car and in the pits, the GPS Device and the several software utilities to communicate with this, and finally some mathematical algorithms that are implemented in the base station related to the filtering, conversion and constraining of the positioning data.

4.2 Telemetry Platform

Because of the team's need to detect failures in the car, make improvements in its components, and establish voice communication between the car and the pit-stop, a telemetry system, along with a communication platform, has been implemented in the FST 4e prototype [6].

This system is composed by two stations: one minicomputer installed in the car, and a general purpose computer located in the pit-stop. The communications between these two stations are ensured by an IEEE 802.11 WiFi transceiver. A brief description of both stations and of the protocol implementation is described in the next subsections.

4.2.1 Embedded Car Platform

The minicomputer that is installed in the car is connected to the CAN bus that transfers the data acquired at the several sensors, such as: speed sensors, battery voltage, engine RPM, engine temperature, etc.

The car's station is illustrated in figure 4.1 and its technical specifications are depicted in table 4.1.

Figure 4.1: VIA EPIA-P700 Board equipping the car's station [6].

This minicomputer runs Knoppix 6.2 Linux operating system, which is installed in the pen drive.

Processor:	VIA C7 x86 @1GHz
Board	VIA EPIA-P700 compact board with 1GB of DDR2 RAM memory
Chipset	VIA VX700 Unified Digital Media IGP
Graphics	VIA Unichrome Pro II IGP
Storage	Pen Drive USB 4GB
Interfaces	1xIDE 1xS-ATA 4xUSB 1xSerial RS232 1xVGA 1xGigabit Ethernet

Table 4.1: Car Station Specification.

4.2.2 Pit-Stop Platform

The pit-stop station is a general-purpose computer, with the following installed software:

1. SQLite database management system, that allows the storage of the data acquired by the car's sensors;
2. User's interface composed by a set of widgets programmed in a Qt platform;
3. Operating system based on Linux.

As it was referred before, the communications between the two stations are ensured by IEEE 802.11 protocol (WiFi) transceivers. In the car's station, such means are assured by an on-board USB WiFi adapter. To guarantee the reliability of the communications, stable reconnection mechanisms were also implemented in order to automatically re-establish the communication link in the event of a disconnection, when the system goes out of the communication range.

4.2.3 Formatting and Transmission Protocol

The message formatting and protocol implementation consists in a interface module already existing in the car, that is responsible for transmitting and receiving the data from the car/pitstop, convert it to the appropriate format used in the communication protocol (represented in the table 4.2) and send it to the other station using the WiFi protocol.

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

Table 4.2: Package structure of the communication protocol.

4.3 GPS Module

The tracking feature corresponding to this work will be supported by a GPS device, as described in the following subsections. This device will be installed in the car, and connected to the mobile station using USB or RS-232. The communication between the GPS and the software layer running in the mobile station is presented in 4.3.2.

4.3.1 GPS Device

Several GPS hardware devices were considered to provide the coordinates on Earth. Once the device will be connected to the USB/serial port of the mini-pc described in section 4.2, some limitations and restrictions have to be met. The most important requisites that were considered in the selection of this particular device were:

1. Low Cost;
2. Reliable;
3. RS-232 or USB Compatible;
4. Hardware ready to connect to a serial/USB port;
5. Compatible with GPSd linux device driver and NMEA 0183 protocol;
6. Compatible with EGNOS Augmentation System to provide good accuracy.

Although the refresh rate was assumed as an important requisite, only GPS devices with 1Hz supply frequency were considered. In fact, devices greater refresh rate would be more interesting for this work, since they would allow to collect more data, increasing the real-time and post-analysis precision. However, those devices have a substantial cost in terms of price, when compared with standard 1Hz modules.

Among the several available hardware devices, the modules depicted in tables 4.3 4.3 represent the considered subset that meets most of the described conditions.

Module	Best Cost	RS 232	NMEA	Easy to connect	EGNOS	Extern Antenna	Chip
LeadTek LR9552LP	1st	Y	Y	Y	Y	N	SiRFstar III
Gsat ET-332	2nd	N	Y	Y	Y	Y	SiRFstar III
Garmin GPS 15	3rd	Y	Y	Y	N	Y	SiRFstar III

Table 4.3: GPS RS-232 Module Comparison.

Module	Low Cost	USB	NMEA	Easy to connect	EGNOS	Extern Antenna	Chip
GlobalSat ND-100S	1st	Y	Y	Y	Y	N	SiRFstar III
Navistick LE GPS	2nd	Y	Y	Y	Y	N	SiRFstar III
Canmore GT-730F	3rd	Y	Y	Y	Y	N	SKYTREQ
GiSTEQ GR-110	4th	Y	Y	Y	Y	N	SKYTREQ

Table 4.4: GPS USB Module Comparison.

For the RS-232 interface, the chosen module was the LeadTek, because it meets most of the defined requirements. Although it does not support an external antenna, this issue will be easily solved, by installing the GPS module on the top of the car, making it to work as it were an antenna.

Since the RS-232 GPS modules needed at least 2 connections: the data serial connection and also the power connection, the chosen module was the USB GlobalSat 4.2. Since this module does not require any additional power source, it will be easier to install to the mobile station. With this, the car station will only need one power source to connect the mini-computer.

4.3.2 Interface Software with the GPS device

The protocol that is used by the majority of the GPS modules to communicate with other devices is the NMEA 0183, created by the National Marine Electronics Association. This protocol

Figure 4.2: GlobalSat ND-100S GPS Dongle.

is a character-oriented protocol with a bit-rate of 4800bits/second. It is also used in sonars, autopilots, echo sounders, and other devices [23] [24].

An example of an NMEA 0183 message is the following:

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
```

This message has the following structure:

GGA		Global Positioning System Fix Data
123519		Fix taken at 12:35:19 UTC
4807.038,N		Latitude 48 deg 07.038' N
01131.000,E		Longitude 11 deg 31.000' E
1	Fix quality:	0 = invalid
		1 = GPS fix (SPS)
		2 = DGPS fix
		3 = PPS fix
		4 = Real Time Kinematic
		5 = Float RTK
		6 = estimated (dead reckoning) (2.3 feature)
		7 = Manual input mode
		8 = Simulation mode
08		Number of satellites being tracked
0.9		Horizontal dilution of position
545.4,M		Altitude (meters), above mean sea level
46.9,M		Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field)		Time in seconds since last DGPS update
(empty field)		DGPS station ID number
*47		The checksum data, always begins with *

The GPSd is a linux daemon that communicates with GPS devices through NMEA messages and converts that messages to C or Phyton data structures, as represented in figure 4.3. It also creates an interface that can be acceded by a client application through a TCP/IP socket in the 2847 port [25].

The Gpsd-client contains gpxlogger, an application program that uses the GPSd daemon to receive the coordinates (along with other information) from the GPS device, returning it to the

stdout[1] in XML format.

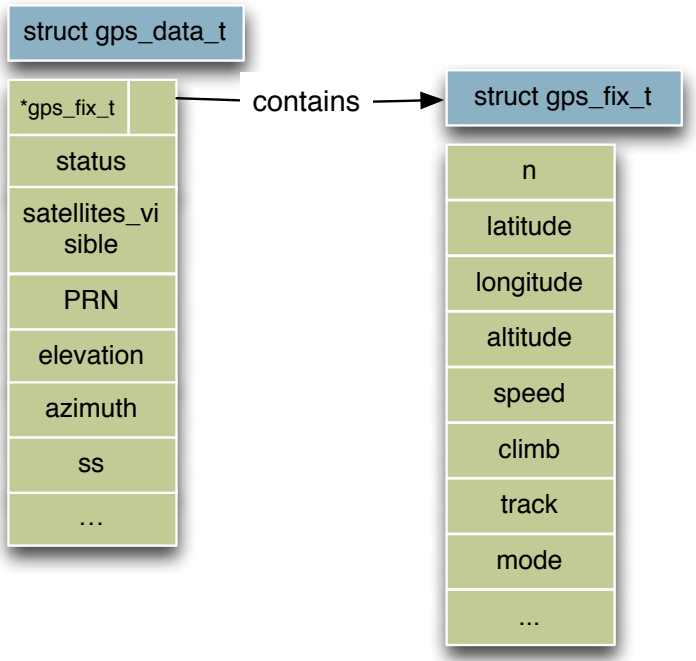


Figure 4.3: GPS data structure returned by GPSd.

This tool transforms NMEA 0183 interface into a C data structure accessible by a socket, is the reason to install this daemon in the car station. Since the car station application is implemented in C/C++ running in a Linux based system, the use of this daemon makes the integration of the module with the system easier and more reliable.

4.4 Supporting Algorithms

In this section will be presented the formalization of the mathematical algorithms used to convert, filter and constrain the GPS coordinates. These algorithms are part of an already existing work ”Uncoupled GPS Road Constrained Positioning Based On Constrained Kalman Filtering” [11], and they were integrated in the base station as-is, by using the MATLAB C Compiler (MCC) tool.

4.4.1 Coordinate Converter

The mathematical expressions corresponding to the algorithm used to convert from the (ϕ, λ, h) LLA coordinates to the XYZ ECEF coordinates were extracted from NIMA [**?**] and are presented below, where x,y and z are defined as:

[1] Standard Output

$$x = (N+h)cos\phi\cos\lambda \tag{4.1}$$
$$y = (N+h)cos\phi\sin\lambda \tag{4.2}$$
$$z = ((1-f)^2N+h)sin\phi \tag{4.3}$$

Where N is defined as:

$$N = \frac{a}{\sqrt{1-f(2-f)sin^2\phi}} \tag{4.4}$$

With:

ϕ : represents the latitude;
λ : represents the longitude;
h : represents the altitude (above the ellipsoid);
a : ellipsoid semi-major axis;
f : ellipsoidal flattening;
N : radius of curvature in the prime vertical.

And where a and f represents the World Geodetic System 84 (WGS 84) values [**?**]:

a = 6 378 137.0 m

$f = \frac{1}{298.257223563}$

With the XYZ coordinates calculated before, the ENU coordinates are obtain from:

$$[enu]_{ENU} = [xyz]_{ECEF}Rot_Z(\lambda+\frac{\pi}{2})Rot_X(\frac{\pi}{2}-\phi) \tag{4.5}$$

The values obtained at the end of the constrain algorithm can be confirmed with an external tool, like Google Earth. In such a case, it is necessary to do the inverse operation, i.e., convert from the ENU system to the LLA. First, from the ENU to ECEF:

$$[xyz]_{ECEF} = [enu]_{ENU}Rot_X(\phi-\frac{\pi}{2})Rot_Z(-\lambda-\frac{\pi}{2}) \tag{4.6}$$

And then, from ECEF to LLA, where the latitude(ϕ), the longitude(λ) and the altitude(h) are defined as:

$$\lambda = 2\arctan(\frac{(\sqrt{x^2+y^2}-x)}{y}) \tag{4.7}$$
$$h = U(1-\frac{b^2}{aV}) \tag{4.8}$$
$$\phi = \arctan(\frac{(z+e'^2z_0)}{r}) \tag{4.9}$$

The U, V and z_0 values can be obtained by using the following auxiliary functions:

$$
\begin{aligned}
F &= b^2 z^2 && (4.10)\\
G &= r^x + (1-e^2)z^2 - e^2(a^2-b^2) && (4.11)\\
c &= \frac{e^4 F r^2}{G^3} && (4.12)\\
s &= \sqrt[3]{1 + c + \sqrt{c^2+2c}} && (4.13)\\
P &= \frac{F}{3(s+\frac{1}{s}+1)^2 G^2} && (4.14)\\
Q &= \sqrt{1+2e^4 P} && (4.15)\\
r_0 &= -\frac{Pe^2 r}{1+Q} + \sqrt{\frac{a^2}{2}(1+\frac{1}{Q}) - \frac{P(1-e^2)z^2}{Q(1+Q)} - \frac{Pr^2}{2}} && (4.16)\\
U &= \sqrt{(r-e^2 r_0)^2 + z^2} && (4.17)\\
z_0 &= \frac{b^2 z}{aV} && (4.18)
\end{aligned}
$$

4.4.2 Kalman Filter

The Kalman Filter [15] is a mathematical algorithm that is commonly used to estimate the past, present or future states using noisy and inaccurate observed values. It outputs resulting values that tend to the real values. The algorithm that will be used in a first phase will be the simple Kalman Filter that will estimate the actual speed and position, based on the previous set of values. This algorithm was chosen because it is easy to implement and to tune.

This Kalman Filter algorithm is split in 2 steps: *Prediction* and *Filtering*

In the Prediction Step, the Kalman filter estimates the current car position and velocity based on its previous values. It also estimate the covariance of these signals [11].

The estimated state is given by the following vector:

$$\vec{\hat{x}} = [\hat{x} \;\; \hat{\dot{x}} \;\; \hat{y} \;\; \hat{\dot{y}}]^T \tag{4.19}$$

In this equation, $\hat{x}$ and $\hat{y}$ are the estimated position in the x and y axis respectively, while $\hat{\dot{x}}$ and $\hat{\dot{y}}$ are the estimated velocity in the x and y axis.

To predict the state vector values, it is used the transition matrix given by:

$$A = \begin{bmatrix} 1 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.20}$$

where t represents the time between the samples (in seconds).

Finally the recursive equations to predict the vector state and covariance estimate are:

$$
\begin{aligned}
\vec{\hat{x}}_{k+1} &= A\vec{\hat{x}}_k && (4.21)\\
\bar{P}_{k+1} &= AP_kA^T + Q && (4.22)
\end{aligned}
$$

where:
A - is the State Transition Matrix
$\vec{\bar{x}}$ - is the Prediction State Vector
$\vec{\hat{x}}$ - is the Estimated State Vector
P - is the Error Covariance Matrix of the Filtering
$\bar{P}$ - is the Error Covariance Matrix of the Prediction
Q - is the Noise Covariance Matrix of the Dynamics
$\vec{y}$ - is the Observations Vector

Since the initial position of the car can change every time the car runs, it will be assumed for the initial conditions that the predicted position and speed (vector $\vec{\hat{x}}$) are the values gathered by the GPS module. The initial Error Covariance Matrix of the Filtration step (P Matrix) adopts the same values given by [11]:

$$P = \begin{bmatrix} 10^2 & 0 & 0 & 0 \\ 0 & 5^2 & 0 & 0 \\ 0 & 0 & 10^2 & 0 \\ 0 & 0 & 0 & 5^2 \end{bmatrix} \tag{4.23}$$

The state and the covariance estimates will be upgraded with the weighted values gathered from the GPS and the prediction step:

$$K_{k+1} = \bar{P}_{k+1} C^T (C P_k C T^T + R)^{-1} \tag{4.24}$$

$$\vec{\hat{x}}_{k+1} = \bar{x}_{k+1} + K_{k+1}(\vec{y}_{k+1} - C\bar{x}_{k+1}) \tag{4.25}$$

$$P_{k+1} = (I_4 - K_{k+1}C)\bar{P}_{k+1} \tag{4.26}$$

where:
K - is the Kalman Gain
C - is the Observation Matrix
R - is the Noise Covariance Matrix of the observation

The Observation Matrix (C matrix) represents the absolute position provided by the GPS device, while the R represents the error associated with the same device. By considering the table present in Appendix A, in the worst case in the implemented module assumes a position accuracy of 10 meters, so $R = 10^2 I$.

Other more advanced algorithms like the Extended Kalman Filter [26] and the Unscented Kalman Filter [16], may still be considered in future evolutions of the prototype in this prediction step to maximize the accuracy of the estimation. This will be particularly useful when other information concerning the movement of the car, obtained from gyroscopes and accelerometers will be available.

4.4.3 Map Constrain

Besides the estimated coordinates obtained from the Kalman Filter algorithm, this module also uses the P Matrix to represent the Error Covariance of Filtering. This P matrix to be used to calculate the weighting matrix ($W = P^{-1}$), which is used to give different weights according to the estimation error in the constraint function.

The mathematical formalization of the constraining algorithm, which adjusts the acquired points to a pre-defined map is stated by function g 2.3.1:

$$g(\lambda) = (\hat{x}^T W - \lambda m^T)(W + \lambda M)^{-1}[M(W + \lambda M)^{-1}(W\hat{x} - \lambda m) + 2m] + m_0 = 0 \tag{4.27}$$

with:

$$M = \begin{bmatrix} C_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.28}$$

$$m = \frac{1}{2}\begin{bmatrix} C_2 & 0 & -1 & 0 \end{bmatrix}^T \tag{4.29}$$

$$m_0 = C_3 \tag{4.30}$$

and where the C_3, C_2 and C_1 parameterize one track segment, contained in the map structure, represented by the following quadratic function:

$$y = C_1 x^2 + C_2 x + C_3 \tag{4.31}$$

In this project, only linear functions were used to represent the tracks. Consequently, the value C_1 will be always 0.

Derivation of *g* mention to determine the roots λ using Newton-Raphson:

$$\dot{g}(\lambda) = -2[m^T + (\hat{x}^T W - \lambda m^T)(W + \lambda M)^{-1} M](W + \lambda M)^{-1}[M(W + \lambda M)^{-1}(W\hat{x} - \lambda m) + m] \tag{4.32}$$

Where the Newton-Raphson method is stated as:

$$\lambda_{k+1} = \lambda_k - \frac{g(\lambda_k)}{\dot{g}(\lambda_k)} \tag{4.33}$$

5

System Implementation

Contents

In this chapter will be presented with a more technical detail, the implementation of the architecture, the used technologies and a technical description about each module.

This solution will be focused in two physical systems, one located in the car and the other in the pit-box:

- **The car system** corresponds to an application implemented in C/C++ that receives, among other sensors data, the GPS coordinates data structure gathered by means of a daemon running in the system. The application transforms this data structure in a formatted message, adds the timestamp and sends to the pit-box through the communication platform.
- **The pit-box system** is divided in logic modules, where each module has a specific functional task that will be described with more detail in the next sub-sections. The different modules were implemented in different technologies: C/C++, Qt and Matlab. In figure 5.2 is represented a more detailed view of the architecture.

5.1 Car-Side Subsystem

The GPSd daemon, described in section 4.3.2 , was installed in the existing mini-computer. This daemon is responsible for creating an interface to communicate with the GPS module. One of the already existing GPSd-clients is the gpxlogger, which uses the GPSd interface to get data from the GPS device like latitude, longitude and altitude. However, such information is provided to the stdout in non convenient XML format.

Hence, instead of using the gpxlogger together with an adapter that converts from XML to other simpler format (like CSV) it was decided to create a similar client based on this. This client communicates with the GPSd through a socket, and then it sends the output (latitude, longitude, altitude, speed and time) directly in the CSV format. This solution was considered preferable, not only because the gpxlogger does not present the information about speed, but also because it is not efficient to convert the file from XML (instead of directly writing it in the desired format).

The overall architecture of this solution was already represented in figure 3.2, by using arrows to represent. The information transmitted from the GPS module to the Base Station consist in latitude, longitude, altitude, speed and time.

When the mini computer is turned on, one script initializes the GPS daemon and starts the application, as represented in Fig. 5.1. Once the application is started it will wait for a start message that is sent by the Pit-Box computer. Meanwhile all the received coordinates will be stored in a local file. Only when the start message has been received, will the coordinates be sent to the pit-box computer.

This allows the mobile station logs all the gathered data in a file for posterior analysis, without having the full telemetry system operational (the base station). When the base station is operational, it will send a message to the mobile station indicating the current session id. With this message, the mobile station will start to send the data to the base station, and in a case of an interrupted communication all the gathered data will stay in memory until the connection between both systems is restored.

The reason for the existence of this start messaging is for the car system to know if it is in an online mode, which consists in real time communication with the base station; or an offline execution mode, which consists in logging the date to one file. Knowing the existence or not of the base station, will avoid to keep all the logged data in memory, once that the mobile station will wait for a reconnection to send all the data kept in memory, which in an offline mode will not occur.

The execution of this application consists in a thread which checks if there is any data structure from the daemon in the Transmission Control Protocol (TCP) socket, whenever it receives the data structure with the coordinates, time, speed and other data, this information will be formatted into a string message to be stored in a CSV file and sent to the pit box for further treatment and presentation, by using the ”Envio” communication class.

The information that is present in this message is:

- **LLA Coordinates:** altitude, longitude, latitude
- **Time-stamp:** number representing the relative instant of the acquisition in order to sensors;
- **Date and time:** the absolute date and time when the coordinates where gathered.

This information is represented by a comma-separated string by adopting the following format:

- `"Latitude;Longitude;Altitude;Speed;SensorTimestamp;Date;Hour"`

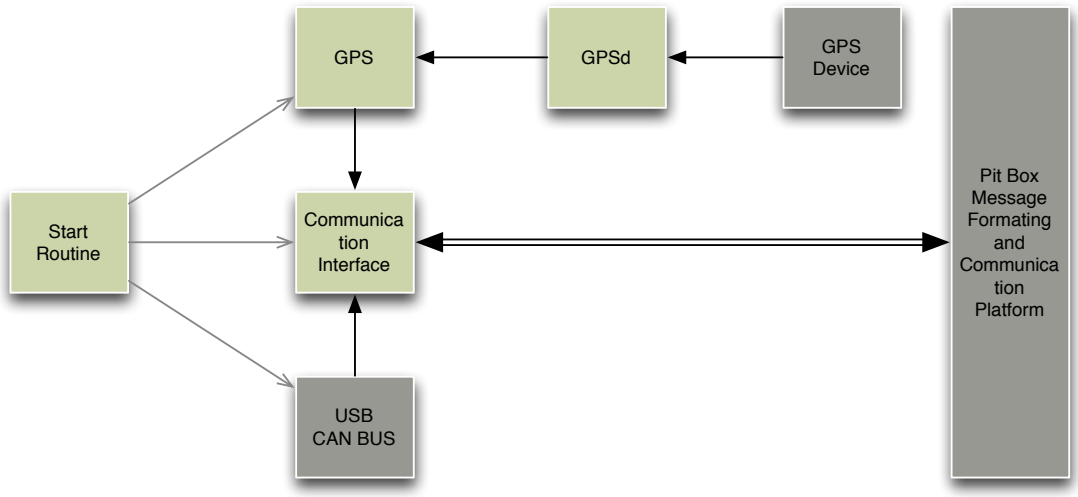


Figure 5.1: Car Side Subsystem.

The time-stamp consists in an incremental number given to each data coming from all sensors. With this number, it is possible to know the sequence of the events and calculate the time of when each event occurs. When a coordinate arrives from the daemon to the application, it will assigned the time-stamp corresponding to the last sensor data, relating in this way all car positions with the sensors data. With such procedure, it will be possible to know which events occur in that particular place. Since the refresh rate of the GPS device is lower than the refresh rate of the other sensors

(1 Hz vs. 10-100 Hz), each coordinate will be related to a refreshed sensor. Some sensor values will not have the same timestamp as the coordinates due to the refresh rate differences, but since the timestamp is ordered it is possible to know that the sensor valued was acquired between two coordinates with lower and upper timestamp.

5.2 Pit-Box Subsystem

In this section it will be presented the Pit-Box side subsystem. This system is the responsible for receiving data from the vehicle, store, process and present it to the user. Here it will be presented the several modules in a more detailed approach. The integration with the current telemetry system and the interconnection between the different platforms will be also described. In the following picture (5.2) is represented the architecture of this subsystem, the modules corresponding to this architecture are briefly described in the following lines:

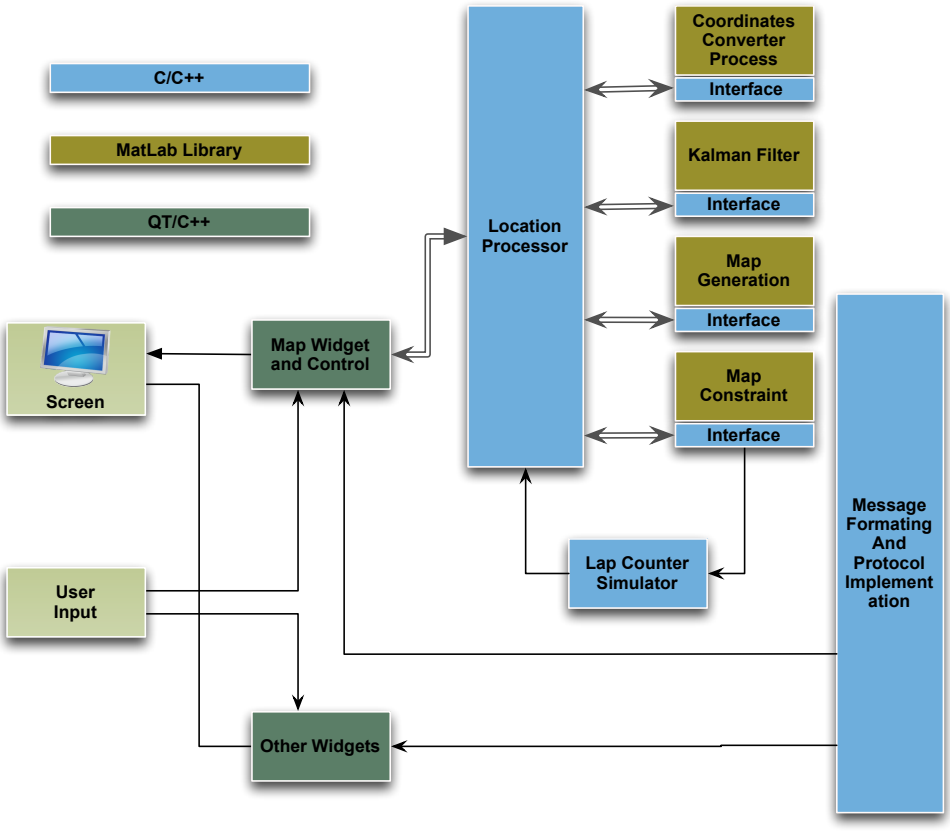


Figure 5.2: Pit-box subsystem.

- **Map Widget and Control Module:** Is the main module that controls all the input/output corresponding to the user interface.
- **Location Processor Module:** The module that orchestrates the information flow corresponding to the position (i.e. coordinates) with the other processing modules.

- **Map Generation:** Converts the points corresponding to the circuit coordinates into a map function format.
- **Coordinate Converter:** Converts the gathered coordinates, from LLA into ECEF and ENU format (and vice-versa).
- **Map Constrain:** Represents the coordinate points inside a pre-defined map.
- **Kalman Filter:** Filters the noise component in the coordinate signal, and provides data (estimates) that will be used in the Constraining Algorithm.

The C/C++ object oriented programming language, is the base of all telemetry system. This language was chosen mainly due to offered facilities to integrate with the previous developed telemetry modules of this application, also implemented in this language. Moreover, since this is a widely used programming language, there are lots of frameworks compatible with this language, like those described in the next paragraphs.

The user interface of the existing telemetry system was implemented using Qt framework. This is a multi-platform framework for C/C++ providing an intuitive API for C++ and Cascading Style Sheets (CSS)/JavaScript-like programming for user interface creation. Since this is a multi-platform framework, it is compatible with Linux/Windows/Mac OS and portable platforms to ensure full compatibility with the existing platforms[27]. This framework was also used to create the user interface of the location system, where the map, current location, controls and other kind of information will be presented.

With the system separated in functional modules, and with this architecture the modules will have low cohesion, being less dependent from each other. This means that each module can be replaced and improved independently, maintaining the functionality and the design of the application.

5.2.1 Matlab Integration

Some of the functionalities developed in the scope of this work required some mathematical specific representations and operations. One example is the work described in Section 2.3.1, which was integrated in this prototype, consisting in the core of the Kalman Filter, Map Constrain and Generator modules. These modules implement the mathematical functions described in Section 2.3.1, consisting in matrix operations, linear regression among others.

Hence, instead of implementing the algorithms in native C code, by creating the framework to support that kind of operations, it was decided to re-factor the prototype implementation into functional and logic modules in Matlab, using the native Matlab libraries, which were already tested and optimized, and convert then to C/C++ shared libraries to be used by this prototype implementations.

Since those converted libraries use different data types, it was applied one adapter that converts the Matlab C/C++ libraries data types into usual C/C++ types, and encapsulates the calls to the libraries, allowing an easy and transparent development of the application. With this approach, the specific Matlab types are hidden from the rest of the code, (implemented in C/C++) providing a smooth integration (see fig 5.3).

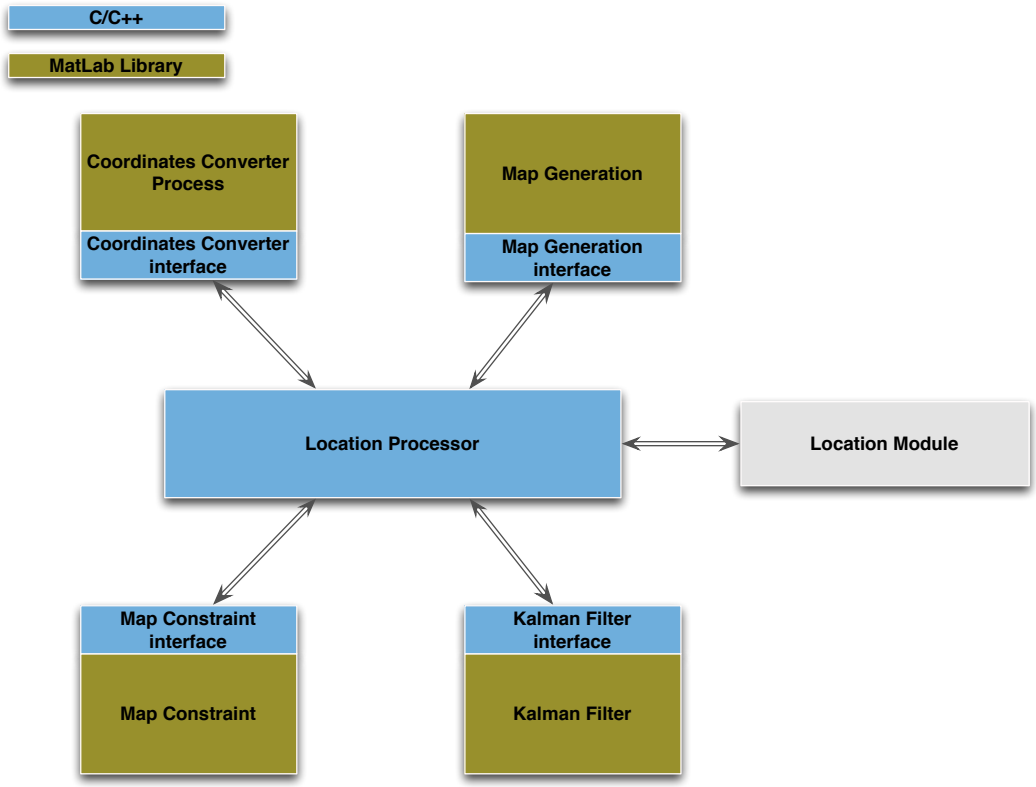


Figure 5.3: Location Module and Matlab/C++ adapters.

The Matlab code was compiled using the “deploytool” present in the Matlab application [28]. Each function generates two files: function.so and function.h. The first one corresponds to the C library containing the Matlab function implementation. This file object will be linked with the remaining project code. The function.h file, provides the interface to the C code, where it is possible to see the functions made available and their signatures: name, arguments and return types.

Below, is an example of the specific C-Matlab structure that is present in the adaptors:

- `mxArray` - Data type that is equivalent to a C array.
- `mxDestroyArray(array)` - Frees the memory allocated to the array variable.
- `mxGetPr(var)` - Returns the memory pointer of the variable var; this is used to read and write from and to this variable using the ”memcpy” C function.
- `allocDoubleMatrix(x,y)` - Allocates space for a matrix with x*y elements of the type Double.
- `mxCreateDoubleMatrix(x,y,mxREAL)` - Allocates space for a matrix with x*y elements of the type mxReal, corresponding to real numbers.

Besides the items above, there is the need to initialize the library correspondent to the Matlab function before the function call. Similarly, it is also necessary to release its resources before ter-

mination. As an example, the methods to initialize and terminate the Kalman Filter implementation are the following:

- `libKalmanFilterInitialize();`
- `libKalmanFilterTerminate();`

The corresponding function call is:

- `mlfKalmanFilter(4, &pPred, &xPred, &xFiltOut, &pFiltOut, coordinate, num, xFilt, pFilt);`

The first argument (4) represents the number of return elements. These are the next 4 arguments that begin with the memory address symbol &. When the function is called, these 4 arguments will be release, so they must be set to NULL value to avoid segmentation faults. The data types of all input and output arguments are the defined in the Matlab library (mx- types).

Any application that makes use of one or more Matlab functions must initialize the Matlab runtime application before the initialization of the libraries. It should also terminate it when the libraries are terminated by using the following two methods:

- `mclInitializeApplication(NULL,0);`
- `mclTerminateApplication();`

In this work, these two methods are used in the Location Processor module, which is the module responsible to orchestrate the data flow between the other modules. These are the only two non C/C++ methods that are used in the main application, outside the adaptors.

Finally the application structure will be something like this:

- `mclInitializeApplication(NULL,0);`
- `libKalmanFilterInitialize();`
- `mlfKalmanFilter(4, &pPred, &xPred, &xFiltOut, &pFiltOut, coordinate, num, xFilt, pFilt);`
- `libKalmanFilterTerminate();`
- `mclTerminateApplication();`

5.2.2 Integration with the existing database

All the data received from the car-side subsystem, is stored for further analysis. Similarly, the data provided by the GPS device will also be stored in the same database that is already present in the telemetry system, located in the pit-box, and based on the *SQLlite* framework. By means of the *sqlite-dev* library, the application will be able so send queries in the string format, like creates, selects and inserts to the database.

Each time the application runs, it will read the CAN_BUS configuration table to identify the type of data that was received. Table 5.1 illustrates an excerpt of such table, corresponding to the GPS device data, where each row corresponds to the sensor's name, its identity and type. For the coordinate's sensor it was chosen the id 777 and for the gps_speed the id 778. All data from GPS has the type: 70.

name	ID	Type
Coordinate_O	777	70
speed_gps	778	70

Table 5.1: CAN_CONF table.

After reading this CAN_BUS configurations table, the application will attempt to create one table for each sensor. In case this table does not exist yet, this table will consist in three columns with the sensor value, timestamp and session id. The unique identifier of each table is the combination of the timestamp with the session id.

However, for this specific application it was decided to aggregate other relevant data in this table, since the GPS provides more relevant information that can be used to add value to the data. Hence, associated with the tree coordinate positions, it was added the timestamp of an event gathered by a sensor which occurs at the same time, the corresponding session identification and the GPS date and hour, allowing to specify and relate the time when the data was acquired. The resulting table 5.2 is represented below:

latitude	longitude	altitude	timestamp	date	hour	type	session_id
38.736557	-9.138795	100.651	0	2012-02-25	11:42:00	2	400
38.736557	-9.138795	100.651	1	2012-02-25	11:42:01	2	400
38.736557	-9.138795	100.651	2	2012-02-25	11:42:02	2	400

Table 5.2: Coordinates table.

For the speed acquired by GPS, it was decided to store the gathered values in a individual table (table 5.3), also with the date, time, timestamp and session id. It was added also the type of the speed value, since it could be a instant speed, lap average speed, max lap speed, maximum lifetime speed etc.

An example of these tables is presented below:

speed	timestamp	type	date	hour	session_id
0.0	50	1	2012-02-25	11:42:43	400
10.399	51	1	2012-02-25	11:42:44	400
35.403	53	1	2012-02-25	11:42:45	400

Table 5.3: GPS speed ”sensor” table.

Where the type id is mapped as following described:

- 0 - ENU Map

- 2 - LLA gathered data
- 3 - ENU gathered data
- 4 - Kalman gathered data
- 4 - Constrained gathered data

As soon as the pit-box application receives the data from the car GPS, it first stores the received data in the database and only then it processes it. The inserted data corresponds to the raw data received from the GPS.

All the created and imported maps are persisted in the database. The information corresponding to the map description and the coordinates that compose the map are stored in two tables. The first corresponds to the already presented coordinates table (table 5.2) which store the coordinates that composes the map with a slight difference: these coordinates will be stored using type = 0, indicating that the coordinates corresponds to a map, where the type = 2 means that it is a raw coordinate and corresponds to a log session.

The second table is exemplified in table 5.4 and contains the id, name of the map, the session where the map was generated, and the coordinates corresponding to the origin of the referential used to generate the map.

id	name	session	origin_latitude	origin_longitude	origin_altitude
1	Palmela	2	38.736557	-9.138795	100.651

Table 5.4: Map Description table.

Each running session occurs in a particular circuit. By using this system, it is possible to generate a map for that circuit once and store it in the database for further use. In this way, only one map needs to be generated and used in several sessions. To load this session and analyze in the offline mode, it is only necessary to choose which map corresponds to that session. This is made automatically when a new session is created and the relation stored in the database. The table that persists this relation is the Map_Session table and is represented in table 5.5 below:

map_id	session_id
1	3
1	4
2	6

Table 5.5: Map - Session relation.

The GPS configuration table contains one raw that will store some data which is common to the application, like the total traveled distance(in meters), maximum speed (km/h) in order to be present to the user, and some other settings and configurations that the user can set (e.g: the kalman filter settings). This table is represented in table 5.6 below:

total_dist	total_max speed	kalman_conf rate	kalman_conf sigma_acel	kalman_conf sigma_ura
2300.23	60.01	1	2	10

Table 5.6: GPS configuration table.

Considering that the whole database management system is based in sqlite3, the reads and updates had to be made using Structured Query Language (SQL) and sqlite3 instructions to connect, disconnect, send SQL instructions and receive the responses. Good practices say that these calls must be encapsulated in a distinct class, instead of being mixed with the remaining project code.

In this telemetry system, the class that is responsible for managing and encapsulating the database functions is the dataBase class. In particular, it incorporates several functions to read and update the tables above. These functions are simply described below:

- **insertMapCoordinates** - Function to insert the coordinates corresponding to the map in the database.
- **insertNewMap** - Function to insert the map details like id, name, session where the coordinates where gathered and origin coordinates.
- **checkMapName** - Check if there is a map stored in the database with a given name; this function is particularly used upon a new insertion of a map, i.e., before saving a new map in the database, it will be checked if it already has one with the same name.
- **insertNewMapSessionRelation** - Insert a new relation between a Map and a stored Session; this is made when a new session is started and a map is loaded or generated for that session.
- **getMapNames** - Get all the names of the maps available in the database; this is used at the beginning of a new session to allow the user to load an existing map.
- **loadCoordinates** - Load the coordinates from a given session, is used in offline session to load the positions of the car in that given session.
- **getTotalMaxSpeed** - Load the total maximum speed value stored in the database.
- **setTotalMaxSpeed** - Update the total maximum speed in the database, if the actual speed record is hit, then this new record will be stored in the database.
- **getTotalDist** - Load from the database the value corresponding to the total travelled distance value.
- **setTotalDist** - Update the total traveled distance, by adding the distance traveled in the current session.
- **loadMapForSession** - Load the map coordinates corresponding to a given session id; this uses the map-session relation table described above.
- **loadMap** - Load the map by giving the map id; this function is used in the start of a new session, to load an existing map.

5.2.3 Integration with the existing communication infrastructure

The transmission of the GPS data from the mobile station to the base station makes uses of an already existing communication infrastructure. This communication platform uses the WiFi protocol to transfer information between the two bases.

The communication framework was implemented by means of the ”Envio” class, which is present in both sides of the application and contains the methods to send and receive messages. The prototypes of those methods are:

- `EnviaMsg(size, mensagem, id);`
- `msg = Recebe();`

In these functions the 'size' argument corresponds to the size of the message, 'message' is the transferred data and 'id' to the identification of the corresponding sensor. The second method returns one message, which contains the three components: size, id and the message data.

In the base station, the platform has a thread always running, looking for new messages. When it receives one message from the mobile station, it will store it in the database (in the table correspondent to the sensorId received within the message), fills the sensor data structure and sends a signal to the correspondent widget, to notify that there is a new value in the structure. The class that is responsible for this is the Serial class, while the data structure is the Sensor class, which contains:

- sensorId
- timestamp array
- value array
- number of values
- name of the sensor

The GPS module within the car-side subsystem has some slightly differences when compared with the usual sensors, since that generates more than one value at time: longitude, latitude, altitude, speed, time, id and time stamp. This system makes use of the "Envio" class and its respective methods to send the message from the mobile station to the base station. The message to be sent has the following format:

`"Latitude;Longitude;Altitude;Speed;SensorTimestamp;Date;Hour"`

This message will be sent by using the "EnviaMsg" method, with the parameter id set to 777 and the size corresponding to the length of the string.

Some modifications needed to be performed in the thread that receives the messages, since in this case it is sent more than one value per message. Since this module was designed to receive one single value per sensor, one way possible to workaround would be to split the message into several messages, each containing one value from the GPS. This solution was not chosen due to the consequent increase of transmitted information and due to the complexity to joins the messages that would result. The adopter approach was to modify the tread, by adding a condition relating to the Id of the sensor, wherever the received message was from the GPS, then it will be processed by other function.

Hence, the function "processGPSData" is responsible for: parse the message into a vector, with fixed indexes corresponding to each value (latitude, longitude, etc.), store the information in both tables of the database (Coordinate_O and speed_gps), and generate a vector to the Sensor data structure which will be read by the Map Widget and Control modules. In order to know when a coordinate arrives to the system, the "processGPSData" function will send a signal to the Map Widget module.

In the Sensor data structure it was added a new field to include the coordinates since had been structure was designed to only contain one value per timestamp and sensor.

5.2.4 Implementation of the location and trajectory tracking modules

Coordinate Data Object

Since the core of this work is based on the position of the car, this object is the data structure that keeps in memory the value of the gathered coordinate through the several modules inside the application. This coordinate object and his attributes are represented in figure 5.4 and is described in the following paragraphs.

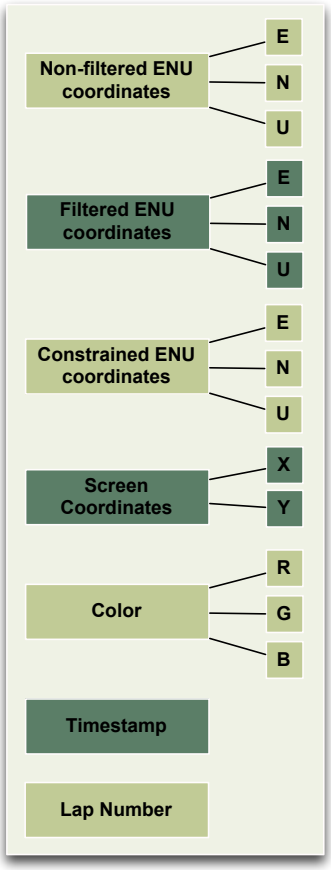


Figure 5.4: Coordinate data object.

Upon the reception and conversion of the coordinates into the ENU coordinate system, the values corresponding to the position are stored in this object to be subsequently processed by other modules (Kalman, Constrain, etc.). At the end, they are converted to screen coordinates to be represented in the user interface. Since the user has the option to choose, in real-time, between the raw coordinates, the coordinates already filtered by the Kalman filter and the constrained coordinates, this object will permanently store these data sets in order to improve the performance, and to avoid the processing waiting time when switching between modes.

In the interface the user can choose between exporting the data into the LLA format or watching the map and car position on the screen. This data structure was designed to support both cases. For the first case, with the ENU coordinates (processed or not) present in this object, it is possible to convert to the original format, i.e., LLA. In the second case, since the computer screen has a different referential (in pixels) the coordinates that are present in a metric scale need to be readjusted to a scale corresponding to the windows size, by using cross-multiplication.

These screen coordinates are also stored in this data object, and are then read by the map drawing module, who draws it on the screen. The other processed coordinates sets (ENU, Kalman Filtered, Constrained) recorded in this data object are only used to generate this screen coordinates. The reason to occupy such memory space with this double information is also due to performance issues; with this, the operations of resizing the widget window, that consists in re-calculating the screen coordinates from the ENU or Kalman filtered coordinates, occurs smoothly and without any delay, thus providing a better using experience.

The screen coordinates are calculated in the widget module, since it is an interface issue and is responsible for drawing the map and the car position on the screen. Each time the user resizes the window, is necessary to re-calculate the screen coordinates from the ENU or Kalman filter data again. Since resizing a window is a common act for the user, this architecture enables fast and smooth changes for the user, since it doesn't imply the initial complex conversions.

Furthermore, the color used to represent each coordinate will be also stored in the data-structure. This information is combined with the screen coordinates and provides information about which color should be used to represent this coordinate. This information is particular relevant in the offline mode, since the color used to represent each coordinate corresponds to the average of the selected sensor values in that position. This means that the color can be mapped with any of the considered (for instance, the speed).

Other pieces of information are also stored here, such as the timestamp of the actual coordinate and the correspondent lap number. The timestamp represents the relative time when this coordinate was gathered. This information is synchronized with all the sensors in the car and allows relating which events occurred at the same time that this coordinate was gathered in the vehicle. The lap information corresponds to the number of times that the vehicle has passed in that part of the track. This information will be used to split the coordinates by track, in order to analyze the values lap-by-lap.

Coordinates Converter Module

The coordinates that are received from the GPS device are represented in the LLA coordinate system. This system makes sense when it is necessary to know the absolute position on Earth. Since the problem here is to locate the car in a more confined area (the circuit), it makes more sense to use a local coordinate system, such as a Cartesian Coordinates System.

Hence, this module of the architecture converts the data from the LLA geodetic coordinates into ECEF coordinates and then to the local ENU coordinates.

There are lots of coordinate types in this implementation, the coordinates received by the GPS are in LLA, the coordinates used are in ENU, and they are then converted to screen coordinates.

The GPS is a system that works in Latitude, Longitude and altitude coordinates. That kind of coordinates is useful to locate an object in the globe, but to locate in a confined and small region, is more indicated a local referential than a global one. With a local referential and with a International Organization for Standardization (ISO) system metrics like meters and seconds we can calculate the distance and speed using the coordinate data, and represent the information

to the user using metrics that he will understand like Km/h , meters for speed and distances respectively.

This module was implemented in Matlab, and subsequently compiled to a C library, to be integrated in this work. It was created a specific interface that transforms Matlab data types and Matlab function invocations to usual C data types and functions.

The interfaces provided by this module are :

- `double * convertENUtoLLA(double *origin, double *coordinate);`
- `double * convertLLAtoENU(double *origin, double *coordinate);`

The ”origin” argument represents the origin of the local reference of the coordinate system. After converting the coordinates from LLA to ENU, there is still an intermediate step where the coordinates are converted to meters, by using the center of the Earth as the origin of the referential (ECEF). Once this is transformed to a local coordinate system, the origin coordinate should be conserved and be used to every invocation of the coordinate converter module, in order to keep consistency.

The coordinate input argument corresponds to the coordinate that will be converted (in LLA format in the first case, and in ENU in the second).

The result will be also a vector corresponding to the converted coordinate.

Kalman Filter Module

To implement the Kalman Filter, the Map Generation and the Map Constrain Modules, it was decided to use the set of algorithms presented in section 2.3.1 and described in section 4.4.2. Were implemented using MATLAB programming language and the MATLAB C Compiler, to convert to a C/C++ library in to be linked with the developed C/C++ program.

Fig 5.5 illustrates the generated trajectory with and without the application of the Kalman filter. As it can be shown, the results after the application of the Kalman filter tend to be smoother i.e., with less noise than before.
The importance of the application of this filtering stage is the following:

- Reduces the noise, to help reducing the error before constraining the trajectory to the map.
- The generation of a map using the Kalman filter and the data acquired from the car GPS, leads to a smoother and more accurate map, since the points have less noise.
- Some values generated by the Kalman Filter (such as P - Error Covariance Matrix of the Filtering) will also be used in the constrain function.

The prototype of the Matlab function that implements the kalman filter presented in section 4.4.2 has the following arguments [11]:

```
mlfKalmanFilter(4, &P_pred, &x_pred, &x_filt_out, &P_filt_out, rate, sigma_acel,
sigma_ura, coordinate ,xFilt, pFilt);
```

- **P_pred** - (output) - Predicted Error Covariance Matrix. Is not used in this system.
- **x_pred** - (output) - Predicted Position Matrix. Is not used in this system.

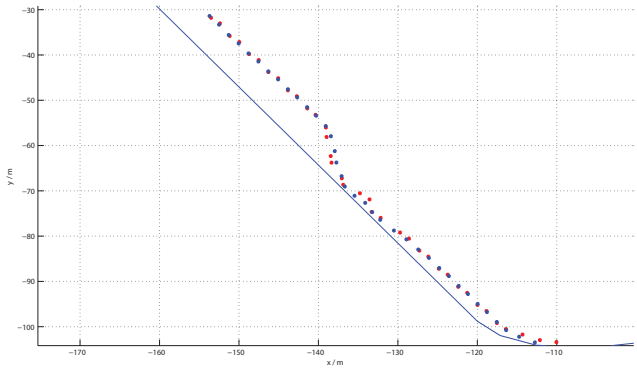


Figure 5.5: Application of the Kalman filter to the gather GPS coordinates; Red: Before filter; Blue: After filter.

- **x_filt_out** - (output) - These are the ENU coordinates after they have been processed by the filter. And it will be as xFilt input, in the next iteration of the filter.
- **P_filt_out** - (output) - Error Covariance Matrix. It will used as pFilt input in the next iteration of the filter, and also in the map constrain module.
- **rate** - (input) - Corresponds to the refresh rate of the samples; in this case, since the GPS hardware has a refresh rate of 1Hz the value that will be used here corresponds to 1.
- **sigma_acel** - (input) - Corresponds to the acceleration of the vehicle.
- **sigma_ura** - (input) - Corresponds to the acceleration error margin.
- **coordinate** - (input) - Is the coordinate that will be filtered
- **xFilt** - (input) - Previous output from kalman filter x_filt_out
- **pFilt** - (input) - Previous output from kalman filter P_filt_out

Actually these functions and values are internal to this module from its output point of view. The functions used to interact with this module (which encapsulate the Matlab invocations and structures) are the following:

- `double * Libkalmanfiltera::FilterCoordinate(double *coordenada)` - receives one coordinate and returns the filtered coordinate.
- `double ** Libkalmanfiltera::getPFilt()` - returns the pFilt - Error Covariance Matrix
- `double ** Libkalmanfiltera::getXFilt()` - returns the text positioning vector.
- `void Libkalmanfiltera::setRate(double n)` - sets the refresh rate of the coordinates
- `void Libkalmanfiltera::setSigmaAcel(double n)` - sets the aceleration.
- `void Libkalmanfiltera::setSigmaUra(double n)` - sets the aceleration error margin.

Map Generation Module

To allow an intuitive identification of the car position, a map representing the circuit was drawn in the screen to let the user to understand the relative position of the vehicle. The Map Generation Module is the responsible for picking up a set of coordinates from the car (or from an external file) and convert it to a set of functions that can be used to constrain the position and to draw the circuit in the user screen.

Hence, the purpose of this module is to generate a data structure with the coordinates of the

map tracks. This structure contains a set of arithmetic functions generated from a set of points by linear or quadratic regressions [11]. There are several approaches to achieve this objective.

The first option consists in recording the car coordinates running around the track, by using the live session mode. Then, the application filters, interpolates and generates the map, by using this gathered data. This is a more inaccurate option, but faster and easier.

The secondary option consists in using an external tool, like Google Earth, marking the points corresponding to the track and then exporting to a CSV or KML data format. This can be a more accurate and sophisticate way of producing a map, but requires specific tools and has to be manually done by the user.

The implemented approach allows the generation of the map by using information from more than one consecutive lap, since the number of points per lap is directly related to the speed of the vehicle. In fact, when the car speed is high it decreases the precision and quality of the circuit representation.

One way to solve this problem is by interpolating the information of a set of laps, thus increasing the number of points and then the quality of the map. However, with too many coordinates the generated map consists in a great number of unnecessary linear functions, since the map is represented by a set of straights that will be used in the constrain mode. With a large number of straights to calculate, the application will become heavy and the performance will decrease. To avoid this, before and after the interpolation, the coordinates are filtered, by removing all the coordinates in a distance less than X meters. In the Palmela circuit, it was used X = 5 to test with good quality and performance.

The linear regression model was adopted because of its simpler procedure to create the map. Furthermore, its output can be used to draw the points of the map in an external tool, like Google Earth or other software: in fact, the second order equations used by quadratic regression models would be more accurate in the curves. However, such accuracy decreases in the straight lines, transforming some straights into parabolas (illustrated in figure 5.7) and turning the algorithm less efficient.

In a hypothetical compromise, the solution could be an hybrid approach, by using first order equations to represent the straights and second order equations to model the curves. However, this approach was not considered because it would require more interactivity with to the user, to choose which points are part of the curves and which are part of the straight lines, thus leading to a much expensive solution.

The points that are used to generate the map can be acquired from two ways:

- Through the introduction of a KML file created by the user, using for instance Google Earth.
- Through a set of points, gathered by one or more laps with the car.

The first solution has the advantage of containing fewer points. This makes the map generation and the map constraint modules more efficient, since it only uses the necessary tracks segments, with clear and non-redundant points, and reduces the number of calculations. On the other hand, it has the disadvantage of requiring an additional effort by the user to draw one new map for each circuit where the car runs.

The advantage of the second solution is related with its usability, since the circuit its automat-

ically generated without the users intervention. To draw a map, the user only has to select the drawing mode and wait while the car is running in the circuit. The gathered points will be used to automatically draw the map. One of the biggest disadvantages is when the points gathered with the GPS are inaccurate. This situation occurs more frequently when the receptor does not have an upper sight to the sky (such as in city centers), leading to a bad generation of the map. Once the map is generated from inaccurate points, this means that the map will also contain errors. As a consequence, when the constrain function projects the coordinates received from the car, these coordinates will be adjusted to an inaccurate map. Hence instead of decreasing the error, this solution can even increase it, mainly if the GPS does not have a clear view to the sky, without any reflections.

In order to add value to the prototype, the final solution will contain both options. The first one to get more precision, and the second one to be used by the team when they do not have time to draw the circuit.

In figure 5.6 it is represented a map generated from a linear regression made of 139 marked points. In figure 5.7 it is represented the same track with a quadratic regression model obtained by using the same points.

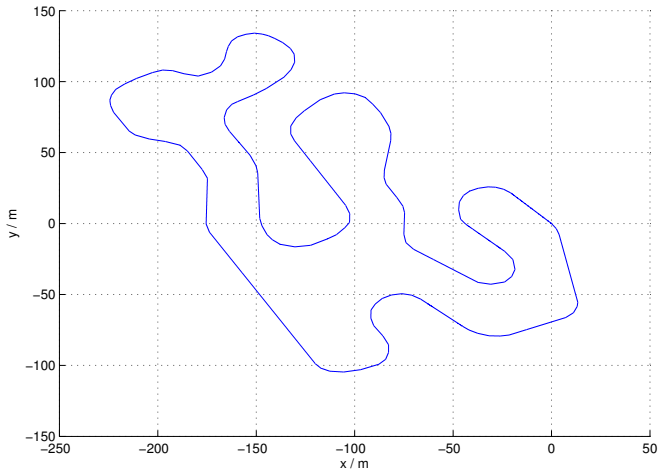


Figure 5.6: Palmela Track Map - Linear regression model.

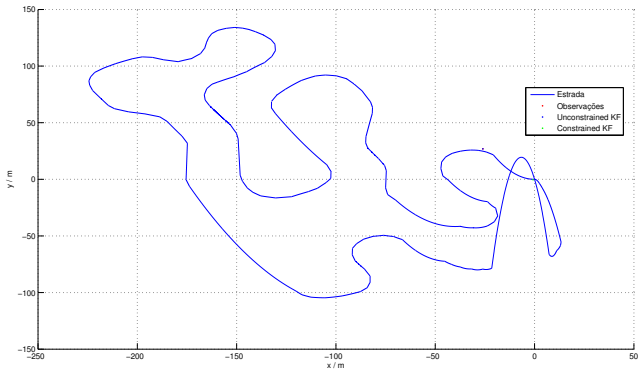


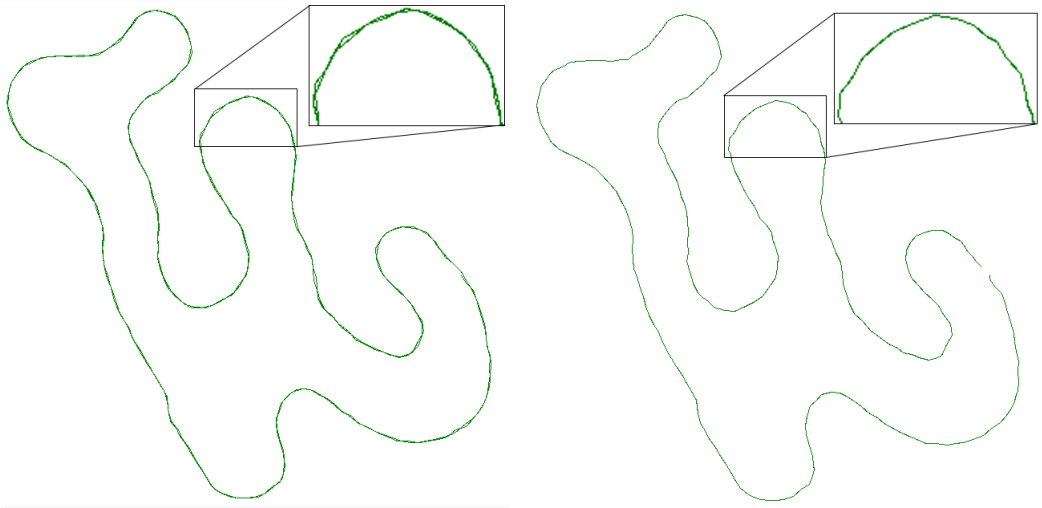
Figure 5.7: Palmela Map - Quadratic Regression.

The interface of the functions that implements the map generation and are available by this module are:

- struct myMap * Libmapgeneratora::generateMap(list<Coordinate*> coordinates) - Returns a structure which contains one matrix corresponding to the tracks represented as functions. The second matrix corresponds to the relation between tracks, where one track is mapped with the previous and the subsequent. This function corresponds to the linear regression implemented in Matlab.

- static list<Coordinate*> removeClosePoints(list<Coordinate*> coordinates) - This function removes duplicate coordinates and coordinates that corresponds to very close points (in an area of 5 meters for example). Removing coordinates closer than a certain distance maintains the precision of the map drawing and increases the performance since, it will reduce the number of track segments, and functions to be constrained.

- static list<Coordinate*> joinLaps(list<Coordinate*> coordinates) - The map is represented by an ordered set of connected coordinates. Considering that the map may be generated from more than one lap, it means that the coordinates will not be connected by the correct order, creating multiple circuits instead of one, as represented in figure 5.8(a). This module detects the overlaps and returns the coordinates in the order that they should be connected, as represented in 5.8(b).



(a) Map generated before joining laps.

(b) Generated map after the points corresponding to consecutive laps have been joinned.

Figure 5.8: Comparation between the map before and after the join maps algorithm.

This is done by iterating each point until another point in its proximity is detected. If this point is inside a radius of x meters (in the performed test it was chosen 6 meters) then it will be the next point to connect. Figure 5.9 represents this process where the black circles represent the first lap and the blue circles the second lap. In this picture, is possible to see the green line that corresponds to the resulting tracks normalized by the algorithm, and the black lines that corresponds to the tracks that are still waiting to be processed.

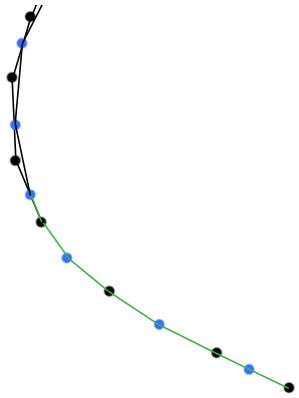


Figure 5.9: Processing of the join laps algorithm

Map Constrain Module

The representation of the car in the user interface consists in a diagram of the circuit, drawing by connected lines, and a circle representing the current position of the vehicle. This representation could be simply archived by a direct mapping of the geographic coordinates to the screen coordinates. However, since the GPS has an error margin and the car may not repeat the same exact trajectory, the current position may not match the track, i.e., the car may be represented outside the line that represents the track (see figure 5.10). This kind of issues may be even worse in areas with sharp curves and counter-curves, since it would be more difficult to know in which part of the circuit the car was, decreasing the quality of the representation.

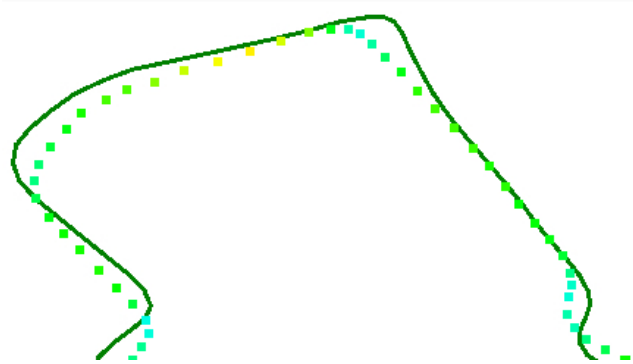


Figure 5.10: Partial map with non constrained coordinates.

The Map Constrain Module is the responsible for restricting the points given by the Kalman Filter in the circuit map previously generated, increasing this way the quality of the representation. This module allows to improve the representation of the circuit and of the car's position, and to decrease the estimation error of its position.

Nevertheless, according to the observations that were obtained with preliminary implementations of the constraining algorithm described in section 4.4.3, some problems still arise in this technique. In fact, the map is composed by linear functions, i.e., straights or parabolas (curves). Hence, whenever straights or curves intersect, the points perpendicular to this intersection may

be not properly constrained to the track, making this algorithm to fail in this specific case (as it is illustrated in figure 5.11). This can happens when the distance between the points and both tracks is the same. To address this particular problem, two possible approaches can be considered:

1. Ignore the outlier point;
2. Associate the point with the next track segment.

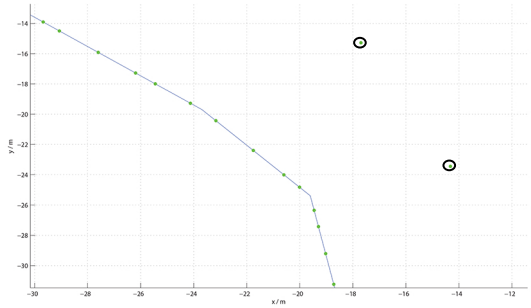


Figure 5.11: Descontinuity problem.

The first alternative was discarded, since the GPS coordinates will be related to other sensor values. If one GPS coordinate was removed, it would mean that a set of values from the other sensors would not be related to the acquired position, or would be related with a wrong position. The second solution induces a small error in the acquired coordinate, until the following obtained samples leave the ”critical-point” of the algorithm.

Other problem related with this constraining is concerned with the generation of the map and with its representation using a structured set of linear functions. In particular, this limitations observed in tracks which are completely vertical i.e., track sections where each edge has the same x value (in this case east value, since it uses ENU coordinates). In this situation, there will be more than one y values for the same x, which is impossible to represent in a linear function. To workaround this problem, it was decided to add a small error (of about 0.001 meters) to the East coordinate corresponding to x coordinates. This way, the track will no longer be completely vertical and a linear regression for this track will be made possible.

The implementation of this module adopts the constrain algorithm described in section 4.4.3 and was carried out as a Matlab C library, like some other modules in this prototype. Its interface is the following:

```
mlfTrack_identify(2, &x_constr, &track, road_map_tracks, road_map_adjacent, x_filt, P_filt, enu_data, track_in1, safety_coef);
```

Below is a brief description of in what each argument represents:

- **x_constr:** (output) - the coordinate already constrained to the map;
- **track:** (output) - the number of the track function for which the coordinate has been constrained;
- **road_map_tracks:** (input) - Matrix which contains the digital map represented by linear functions;

- **road_map_adjacent:** (input) - Matrix which contains the relation between tracks;
- **x_filt:** (input) - Coordinate filtered by the Kalman filter;
- **P_filt:** (input) - Covariance matrix, corresponding to the output of the last Kalman filter iteration;
- **enu_data:** (input) - Non-filtered coordinate;
- **track_in1:** (input) - Current track, usually the last track which was constrained in the previous iteration;
- **safety_coef:** (input) - Error margin (in this prototype, the default value is 0.1).

Since this module also represents an interface between the Matlab C library and the remaining C/C++ code, the interface available for the remaining project is the following:

- `Coordinate * Libmapconstraina::contrainPoint(double** pFilt, double ** xFilt, Coordinate * coord)`

Lap Detector

During a race, a significant set of data is acquired in this system. In order to divide this information in small analyzable blocks it was developed this Lap Detector algorithm. This allows dividing the gathered information by laps, by detecting the beginning and the end of each lap, thus providing a more organized way for the user to analyze the information.

Since no Lap Beacon sensor is installed at the car, or at the circuit it was used the Map constrain algorithm output to calculate the lap changes. With the information provided by the number of tracks generated by Map Generator module, it is possible to divide the map in two areas containing the same number of tracks (see fig. 5.12).

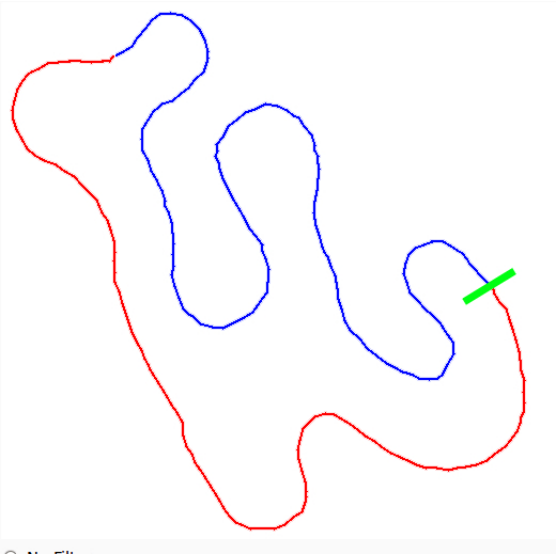


Figure 5.12: Lap detector.

When a coordinate is acquired by the system and processed by the constrain module, this module will also returns the id of the track to where the coordinate was constrained. By maintain-

ing a history of the order of in which area the map was constrained, it is possible to determine when the vehicle has made a complete lap.

By looking at the figure 5.12, it is possible to observe that the map was divided in red and blue areas, the green line represents a virtual finish line. By receiving coordinates in the same order which they were acquired, and if the first 10 coordinates (for instance) were constrained to the red zone, and the following coordinates were constrained to the blue zone, the next time that one coordinate will be constrained to the red zone it will mean that the vehicle crossed the green line completing a lap.

Map Widget and Control Module

The Map Widget and Control module is the logical part of the application which manages both the user interface and the modules responsible for processing / storing the information. The architecture of this module can be observed in figure 5.13. It contains three sub-modules: Widget, Map Draw and Logic. The Widget and Map Draw sub-modules corresponds to the Representation part of this prototype, while the Logic Module is the responsible for providing the information to be provided by the other modules as support (Location Module, Database management, etc.).

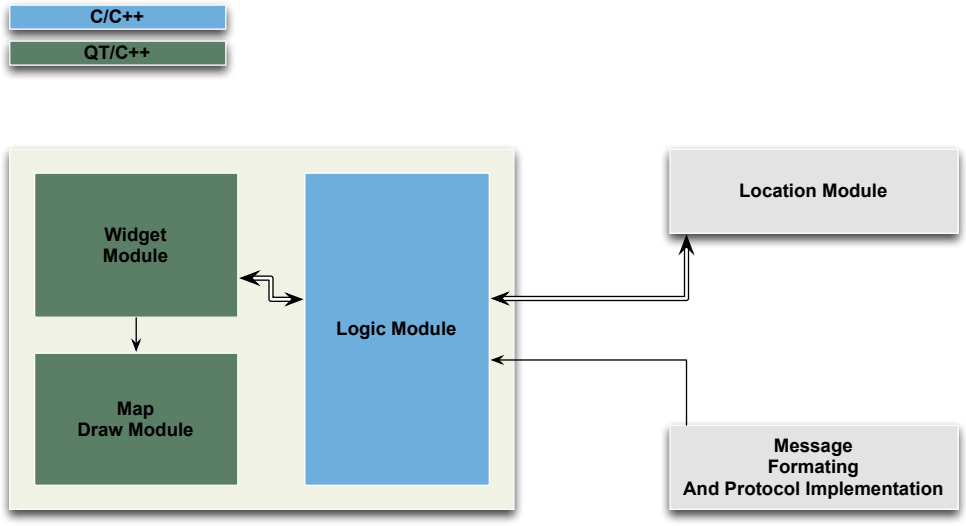


Figure 5.13: Qt widget architecture.

Representation The graphical representation of the whole application consists in a set of widgets developed in Qt framework. Each widget provides an intuitive representation of the state of the car, transmitting that information to the user in the shape of graphs, gauges, and other tools. For this specific work, the representation will consist in one widget that is integrated among others in a unique application corresponding to the telemetry application. In a mode called ”Live Session” this widget will represent the real-time position of the car while in ”Offline Mode” it presents the correlation of the data, by relating the position with other events. To accomplish this, it is necessary an interactive process with the user, as well as a set of available options to improve

the representation of the data according to the user needs. In the following paragraphs it will be explained the role of this module and its iteration with the user.

Logic Module The control part is implemented in the Logic sub-module, which manages the information that is going to be present in the user screen. The management of this information consists in using other modules to process the information, to store in the database or using internal functions to process the data.

This is the component that makes the bridge between the representation, processing, and data receiving (see fig. 5.13). The Logic component receives the data from the communication platform, sends it to the Location Processor Module, according to the user options and current operation Mode, and finally sends the processed information to the Map Widget component, to be presented in the screen. This module also converts the ENU coordinates into the screen coordinates, and calculates the travelled distance, the average speed and the actual speed by using the ENU coordinates (in meters) provided by the location module.

Map and Distance Calculator By using only the data acquired by the GPS it is possible to extract more information besides the one already provided by the sensors. In fact, by using the information corresponding to the car position and the refresh rate of the device, it is possible to calculate the distance that the car has run, its speed and also the average distance and other derivable information.

This information could be obtained by the car sensors (like the speedometer and the odometer), however, they imply that the car is equipped with these sensors, properly installed and connected to the system by CAN-BUS. Although, this approach (based on the GPS) is less precise, it was very useful in the test and debug phases of the project wherever the car did not have those sensors installed, or was simply not available. Since this approach is easy to install and is not "car dependent", it only required the usage of a computer (netbook for instance) and a GPS pen in the car thus avoiding technical installations.

Screen Coordinates Converter The coordinates from the car-side subsystem are in Latitude, Longitude and Altitude format. These coordinates are then sent to the Processor Module, where they are converted, filtered and returned in ENU format (in meters). Since the screen units are in pixels, these coordinates must be converted to screen coordinates in order to be represented in the screen and scaled in order to fit the window.

The screen coordinates functions are implemented in this module. The conversion and scale are done using the cross-multiplication method, receiving as inputs the LLA coordinate, window size and the maximum distance between two current LLA points. Wherever a new coordinate is received or each time the window is resized, these operations are applied to all coordinates being presented.

MapWidget The Map Widget sub-module corresponds to the part of the prototype responsible for the implementation of the user interface, the management of the screen menus and events related with the objects inside the widget. In this widget, along with the MapDraw Module are the

visible parts of the application. The communication with the user makes uses of buttons, lines, points, labels, boxes, menus, and other objects present in Qt framework.

To represent the information desired by the user some auxiliary steps may be necessary. In this case, the desired information could be either the real time analyses of the car location, or an offline presentation, with more attention being devoted to other functionalities, such as the map generation/loading, file load/save, some configurations and extra functionalities. To support all of these functionalities the application is designed in a wizard like style allowing the user to change between screens and configurations until it reaches the desired configuration. The structure of the wizard menus is displayed in figure 5.14, and a brief explanation of contents of each screen is present below:

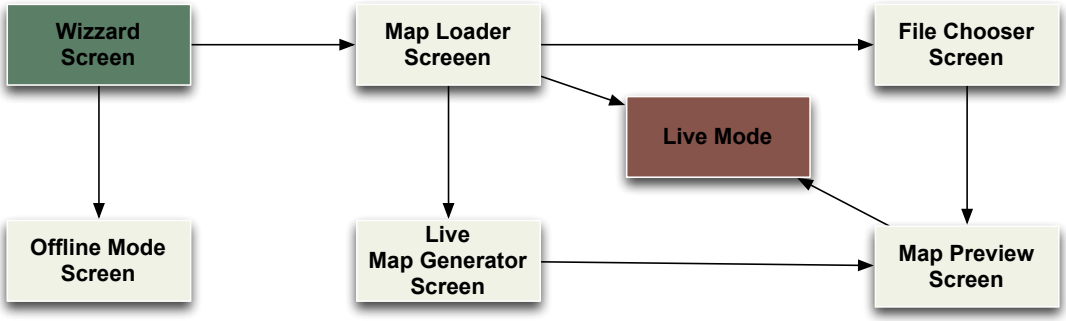


Figure 5.14: Map Widget Screen Flow.

- **Wizard Screen:** The application begins with a wizard that allows the user to choose between logging a new session or loading a previous logged one. According to the user choice, the Offline Mode Screen or the Map Loader screen will be presented.

- **Offline Mode Screen:** In Offline Mode of operation, it is presented the map with all the detected positions of the vehicle as well as, controls to choose the filtering mode and the sensor to relate in the screen. The sensors values are mapped with a color bar and a scale, so the color of the coordinate will change with the absolute value of the sensor in that specific position. It is also presented a set of buttons to change the presented lap, to navigate through the coordinate, and to generate a report correspondent to the selected sensor.

- **Map Loader Screen:** In this screen the user has the option to select from the maps available in the database through a combo-box containing the id and name of each map. It also contains three buttons corresponding to Load Map from Database, Generate from a File and Generate from Live Session operations. According to the pressed button, the Live Map Generator, Live Mode or File Chooser Screen would be chosen.

- **File Chooser Screen:** A simple screen where the user can open a new file, by navigating through the file system. After choosing the file, the Live Map Generator Screen will appear.

- **Live Map Generator Screen:** In this screen, the coordinates collected in real-time by the car are presented, as they are arriving to the system. As a consequence, the shape of a map will start appearing. Since the vehicle is already moving, the speed and travelled distance are also represented, as text, in this screen. To control the generation of the map there will are several controls: Start the map generation, re-start, stop and finally to generate the map. After pressing the last button the Map Preview Screen appears.

- **Map Preview Screen:** After the points are collected and drawn in the screen, the map will be generated and represented as set of aligned lines. In this screen the user may want to enter the name of the map and save it in the database through a text box and a button.

- **Live Mode:**

  Just like the Map Generator Screen and the Offline mode, in the Live Mode the map and the filter controls are also present. However, it also includes an area that shows the values concerning to the speed (maximum/average/etc) and travelled distance.

For a better organization of the visual space of the widget, the screen is divided in several areas. In the Live mode, it is present the Draw Area (where the map is displayed), a Control Area (where the control buttons are placed) and an information Area (where data related to the speed, altitude and distance are displayed). In the Offline session mode interface, besides including a Draw Area and a Control Area, like in the first mode, the information area will consists of a Color Bar and a scale widget, which map the sensor values into colors. A brief description of these areas is described below:

- **Draw area:** The draw area is where the map is drawn and the acquired coordinates are represented. The map is represented as a green line, while the points corresponding to the location of the car are colored according with the value of the chosen sensor. The Map Draw module is the module responsible for this part.

- **Control area:** It is the area where the control buttons are present, i.e. the filter modes and the sensors. In this area there is a button to Print a report, to hide the map and show the plot, a combo-box with all the sensors and radio buttons to choose which filter should be applied in the map (no filter, kalman filter, constrain filter).

- **Information area:** Displays text information to the user, containing a list of real-time updated information according to speed (actual, maximum, average), distance (total, partial) and altitude.

- **Color bar and scale**. Each coordinate will be drawn with a specific color, according to the chosen sensor value, by using QwtLinearColorMap method of qwtlibrary [29]. With this library it is possible to choose which colors to present according to the values of the sensors. After setting the interval of values in this object, giving it a value it gives back the color correspondent in the color scale. This color will be used to paint the position of the car corresponding to where those values were acquired.

To relate the colors with the values in the right side of the applications it is presented a Scale and a ColorBar which varies between the minimum and maximum values of the sensor data in the current session, and relates the values with the colors, to allow the user to understand which value corresponds to a given color.

MapDraw This component consists in a sub-part of the Map widget corresponding to the area where the map/circuit is drawn. The information corresponding to the actual location of the car is represented by a circle. All the involved operations to draw the track, represent any point alter the color values, etc., are implemented in this component, so the actual map state containing the current coordinates and loaded map are stored here in data vectors.

Sensor - Position relation With the coordinates and sensors already loaded, this module adjusts eventually different sampling rates used by the sensors and matches the corresponding timestamps.

In practice the refresh rate of the sensors is usually higher than the refresh rate of the coordinates. Hence, the value represented in that coordinate is given by the average of the values near to that timestamp. As and example, if the GPS works at 1Hz and the temperature sensor samples at 5Hz, each coordinate will represent the value of an average of 5 reads of the temperature sensor. By applying this it, guarantees that no value is lost or ignored.

Report Generator In this module, it is output a PDF map report from the collected data. The implementation presented in this document corresponds to screen that is presented in the offline mode. The only differences consists in drawing several laps in the same document at the same time, and additional information corresponding to the number of laps, name of session, pilot, whether, name of the current sensor among other auxiliary information.

6

Results

Contents

6.1 Summary

Since the actual prototyping vehicle has an difficult and exclusive access, a significant part of the conducted tests need to be performed using other alternative (but still reliable) means like: personal computers, GPS simulators, an usual vehicle to install the mobile station, and software tools to analyze the data. In this chapter, it will be described the development test tools, the evaluation methodology, the conducted tests and their results.

6.2 Evaluation Tools

In this section, it will be presented some software tools used to test and evaluate the implemented modules, as well as other tools used in the development of the solution.

6.2.1 GPSfake

To simulate the GPS behavior, it was used the GPSfake tool [30]. This tool takes as input an NMEA[1] file with the coordinates to simulate and it communicates with the GPSd just as a real GPS device, using the coordinates pre-defined by the user (see figure 6.1).

This allows to test the solution with previously acquired data, instead of using a real GPS device which needs a perfect and clear vision to the sky in order to receive data.

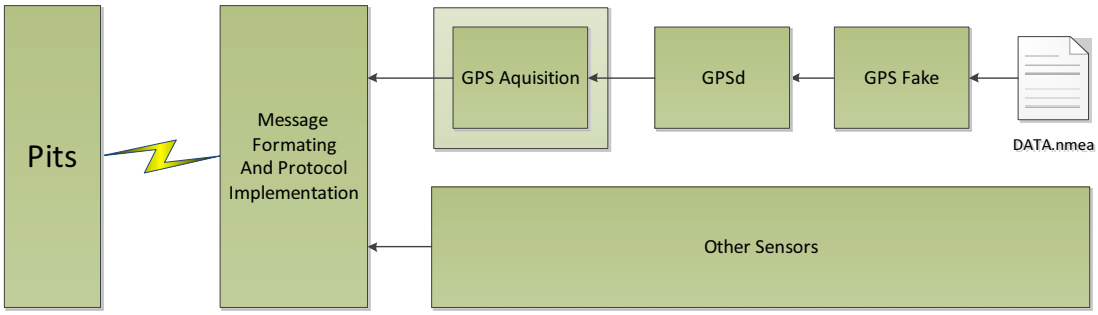


Figure 6.1: Mobile Station Architecture with GPSfake.

6.2.2 Google Earth

The Google Earth program allows to visualize in the globe the data obtained by the GPS and stored in a KML file. This allows to confirm if the coordinates acquired by the device or from a software module correspond to the truth positions.

This tool also allows the user to mark check points, draw the circuit's map and save it in a KML file. This file can be used as the input file to the pit-stop's Map Generation Module.

Since the Google Earth only deals with KML files, it is necessary to use a converter that provides the translation from the file gotten by the test solution (in CSV) to KML, and the conversion between the map drawn in the Google Earth (in KML) to the same format that is read by the map generation module (CSV). During the preliminary steps of the development, the used converter

[1] Described in section 4.3.2

was the KMLCSV Converter [31]. Currently, the developed solution is also able to convert both formats.

Another tool used also to confirm, represent and convert the coordinates is the GPS Visualizer, available in the following website: `http://www.gpsvisualizer.com`

6.2.3 MATLAB

The MATLAB programming language is a very convenient and efficient way to implement the mathematical operations involved this project. It provides very useful mathematical functions, like matrix operations and polyfit, which allows to easily and efficiently implement linear and quadratic regressions used during the map generation. With MATLAB it is also possible to graphically represent the data in the debugging phase, thus preventing the need of the whole representation module during the developing phase.

In order to integrate the MATLAB source code with the remaining modules, was used the MCC tool that allows to convert the algorithms written in MATLAB programming language to a C/C++ code or library, that can be easily used by other modules [32].

6.2.4 SQLite Browser

This application requires several and frequent read/update actions in the database. The values gathered and generated by the prototype are stored in a SQLite database and then are loaded to be presented in the screen when the offline mode is used. To guarantee that the values are correctly read and written in the database, as well as to add test values directly in the database, to be subsequently processed by the application, it was used the SQLite Browser application. This tool allows to do the following actions in the database [33]:

- Create, define, modify and delete tables;
- Browse, edit, add and delete records;
- Search records;
- Import and export records as text;
- Import and export tables from/to CSV files;
- Import and export databases from/to SQL dump files;
- Issue SQL queries and inspect the results;
- Examine a log of all SQL commands issued by the application.

6.3 Development and Testing Scenarios

In this section it will be briefly described the development and tests stages, starting with the acquisition of data, the processing, representation and integration. Most of the results corresponding to these tests will be presented on section 6.4.

Mobile Station The mobile station application was the first subsystem to be developed, and it was used to analyze the viability of the GPS signal in terms of reception and precision.

To accomplish this, a complete data-set was acquired at Kartódromo Internacional de Palmela track. The data was acquired using a laptop computer running Ubuntu 10.10 operating system connected to the GPS Bluetooth module [2] described in Appendice A. This particular track was chosen because it is the circuit where the FST prototypes have been tested. To collect a considerable set of samples, the developing team walked two laps around the circuit with the laptop running the acquisition software and the GPS module.

A first collection was extracted by following the most probable trajectory in the track [3]. It took 12m33s and it were collected 754 points with a sample rate of 1 LLA coordinate per second.

In the second lap, the collection was extracted by following the center of the track, in order to get the coordinates that allow modeling the real circuit with more precision. This lap took 14m43s and 880 samples were collected with the same bit-rate.

Since the collections were extracted by walking along the track, a great quantity of points was obtained. To simulate the car's movement with these data-sets, it is only necessary to sub-sample the intermediate points.

To analyze these results, it was used the Google Earth software to present the acquired data in the Earth. By using satellite images, it was possible to relate the gathered information and observe its precision.

With the obtained data, it was possible to revert the data from CSV to NMEA protocol, in order to be used by GPSFake. This tool allowed testing the output of the application, which consisted in a string message containing the coordinates, speed and timestamp data. Later, this tool was used to test the integration with the communication platform.

Base Station The base station is the component of the system where the data is processed, stored and represented. The first modules to be developed and tested were related with the coordinates processing: Coordinate Converter, Map Generator, Coordinate Filtering and Constraining. After correctly processing the data, it was developed the set of modules responsible for the user interface and management of this part of the system: Map Widget and Control. The Lap Detector and the Report Generator were the last modules to be developed.

The algorithms corresponding to the coordinates processing modules were tested in Matlab by using the source code provided by the investigators 2.3.1 of the work described in 2.3.1. These tests had the aim to check the impact of these algorithms in this project, corresponding to the quality and precision of the output coordinates data. The quality and precision was subsequently checked using the Google Earth tool to observe the difference between the non-processed and processed coordinates in the satellite images.

With the development of the Map Widget and Control modules, it was tested the representation part (generated map and actual location) by using one of the developed tools, which reads an CSV file (containing the coordinates used in the previous test) at a 1Hz rate. The output of this representation module must be a map and several coordinate points, similar to the ones obtained in Matlab and Google Earth in the first tests.

The Offline mode was the last to be developed, since at the time of the development the

[2] GlobalSat BT-359W.
[3] By following the tires marks.

system was integrated with the telemetry application, and more data could be collected, in order to test the lap detection and the relation with the sensor data. This data collection also allowed testing the system since the acquisition of coordinates until the representation in the screen.

The GPS module was then connected to a laptop, which served simultaneously as base and mobile station, by running both systems. This laptop was placed in a personal car to simulate a dynamic much closer to the FST prototype than the previous data, which had been acquired by walking. It was performed a lap in a mid-sized village to generate the map, and then three other laps in order to acquire more points and to test if the application could detect the three laps. Besides the coordinate points themselves, it was also possible to acquire the speed. The speed information and the altitude information corresponding to the 3rd coordinate of the GPS were used as external information (in the same way as the other sensors values) and marked with the corresponding time-stamp.

The length of the considered circuit is about 1.8 km, not much more than the previous one (1.2km). Including the generation of the map, it was gathered 1202 coordinates in a 20 min session.

By issuing an offline session with the gathered data, it was possible to observe in the developed widget the drawn map, the coordinates gathered splitted by the correct laps and the coordinates correctly colored according to the sensor or altitude ”sensors”. Once again, the Google Earth was only used to confirm the precision and the relation between the place and altitude or speed.

Communication Platform and Integration Tests The communication platform is the part of the system that allows the car-station to communicate with the base station. The first test with this framework was done by trying to establish a communication link between the mobile and base application in the same computer, by using localhost address. Then, it was used the telemetry router and the mobile station hardware (with the GPS connected), in order to send the coordinates to the base station through WiFi.

6.4 Final Tests

The final tests were supposed to have been conducted by using the FST 4e prototype under construction. However, due to the scarce availability of the car to perform those tests, and due to the fact that many of the sensors were not installed yet in the car, it was not possible to acquire the necessary data in the race context. To work around, other alternative, but fully conclusive types of tests were done with the help of the FST team. These tests included an usual vehicle (instead of the prototype), and some extra data acquired by the GPS (such as speed and altitude), instead of other sensors included in the vehicle.

These tests were performed by using the final prototype equipment, which will be definitively installed in the FST-04e prototype: the mobile station hardware, the telemetry platform router and the FST team’s laptop.

6.4.1 Data Acquisition

The test consisted in a simulated circuit located at IST Alameda Campus, where the telemetry router was installed in a place near the first track (see fig. 6.2) and the team's base station laptop was located near the router. In order to gather the data, it was used the FST 4e prototype mobile station. The mobile station was installed in a usual automobile and it was powered by the car cigarette lighter (by using an adapter developed by the team).

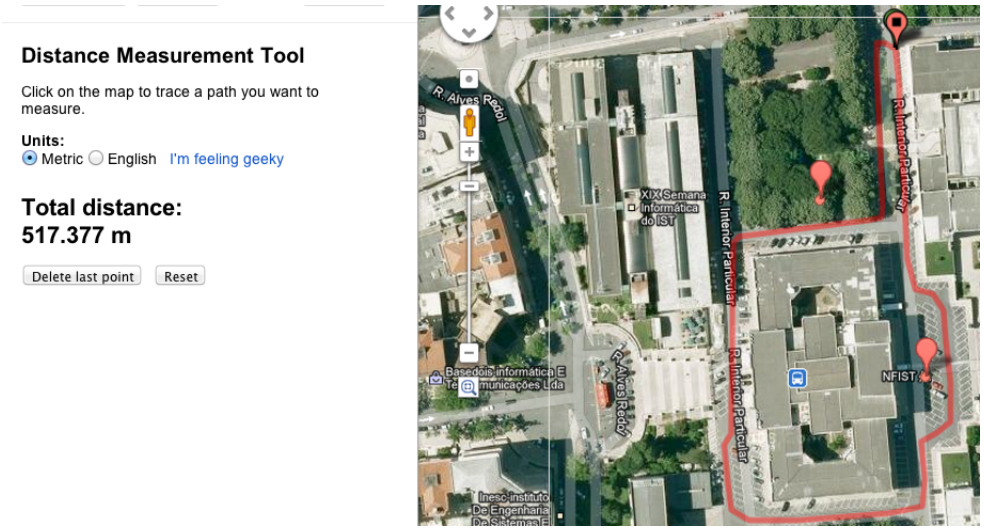


Figure 6.2: IST Test Circuit.

The test consisted in performing three laps in the circuit, where the first lap was used to generate the map. From this logging session, 447 coordinates were obtained from a 5 minutes and 37 seconds of data logging.

The raw data was saved both in a CSV file at the mobile station, by using the logging functionality, and in the base station's database using the telemetry communication platform. Both logging results proved that the mobile station was properly acquiring, logging and sending the GPS information. In the following figures (fig. 6.3(a) and 6.3(b)) it was represented the data collected in the mobile station. The first figure (fig. 6.3(a)) represents only the data used to generate the map, while in the second figure (fig. 6.3(b)) presents all the data to be analyzed.

The location data, acquired by the mobile station, was also sent to the base station. In this test, only a partial area was covered with WiFi network. This area is represented in figure 6.3(b) by blue delimiters, and corresponds to the area where the information is presented in real-time. Besides these limitations, the data concerning to the coordinates were properly transmitted to the base station as soon as the vehicle returned to the blue area. By using the SQLite database tool, and by analyzing the timestamp information it was possible to observe that no data has been lost.

In figure 6.4 it is represented a screenshot of the application, just before the map has been generated. Along with the coordinates, it is also present the GPS speed and distance information.

Although this test has been made in an urban environment, it is possible to conclude that the most part of the coordinates corresponds to locations with an error margin smaller than 3 meters (measured in Google Earth). In a real track context (like in Kartódromo de Palmela) the conditions concerning the visibility of the sky are significantly better (since there are no buildings around) and

(a) Coordinates gathered during the live generation of the map.

(b) Complete set of gathered coordinates during the test session

Figure 6.3: Coordinates gathered during map generation and live session.

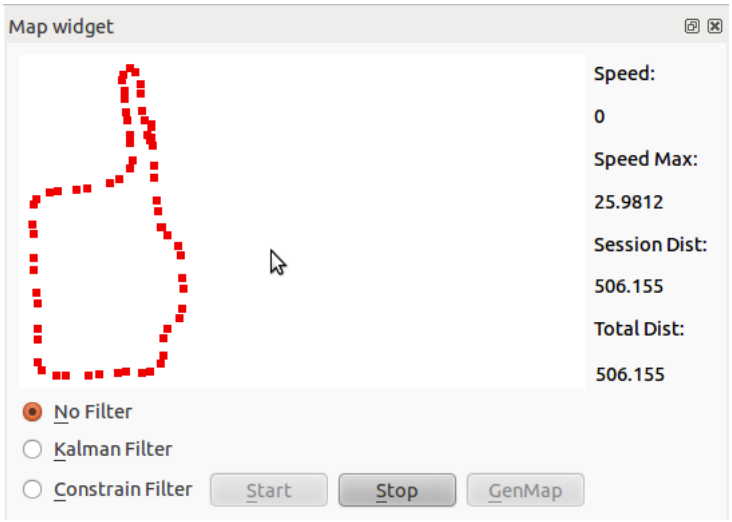


Figure 6.4: Coordinate points gathered for the map generation.

the width of the track is about 11 meters (almost the double of these test tracks).

6.4.2 Map Generation and Interface Integration

In order to verify the proper generation of the map, two tests were conducted: the first test corresponds to the live mode generation, while the second test was done by importing an KML file from a circuit designed with Google Earth.

The following application screenshots (figs 6.5 and 6.6) represented the maps generated with these two options. In these screenshots, it is possible to observe the main differences between them, where the imported map is slightly more accurate. In these figures, it is also represented the

two forms of integration of this interface with the other widgets present in the completed telemetry system. The first option (fig. 6.5) represents the widgets in separated windows (default), while in the second 6.6 screenshot it is possible to observe the widgets docked in the main widget.

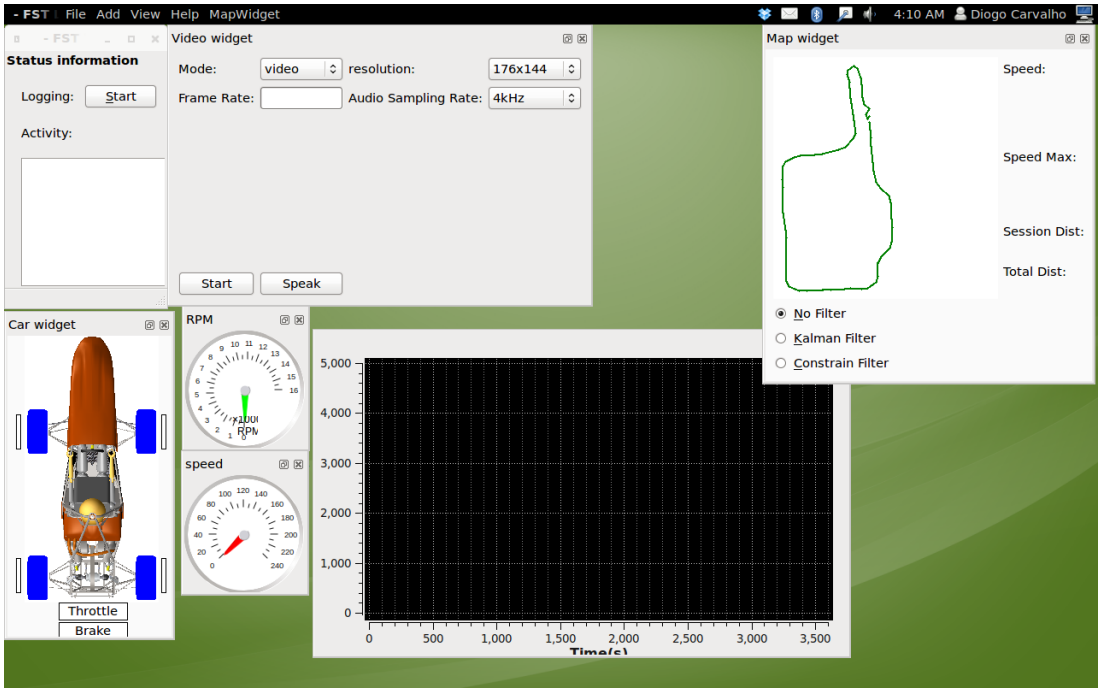


Figure 6.5: Map Generated from a live session.

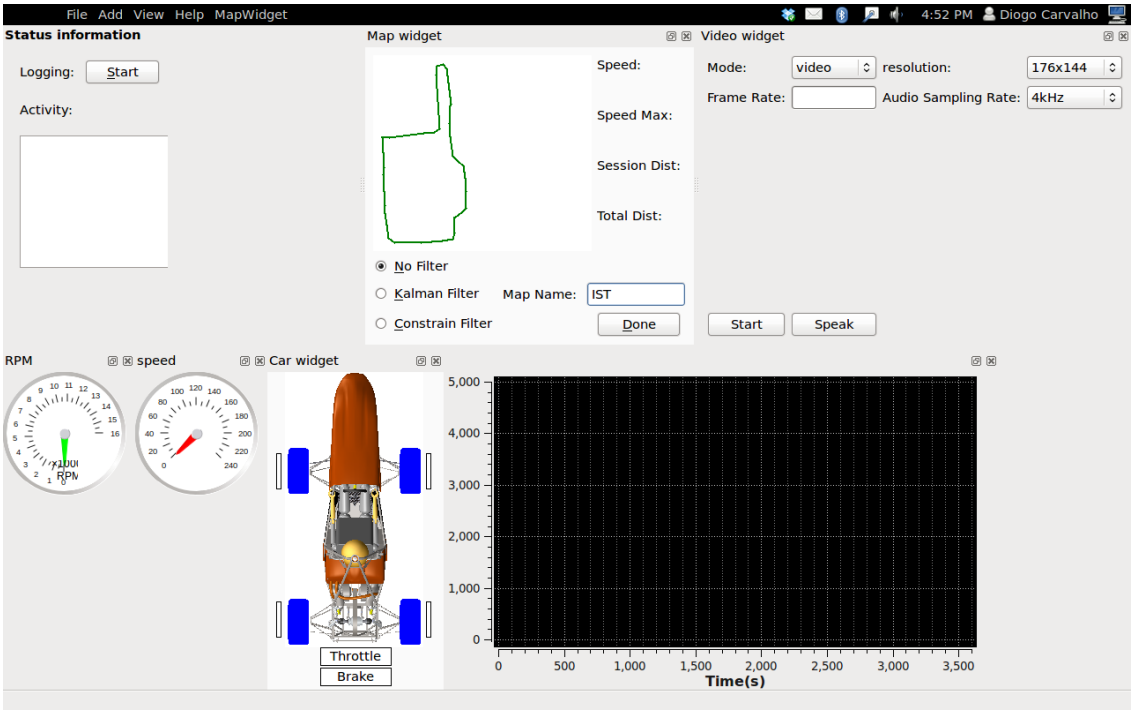


Figure 6.6: Map generated from the KML file.

After saving the map into the database, the application continued receiving in real time the positioning information. In figure 6.7 is represented the position of the car relating to the map

generated during a live session. Besides the three filtering modes that are available in this live session, the results concerning to this information will be presented later in the Offline Mode subsection.

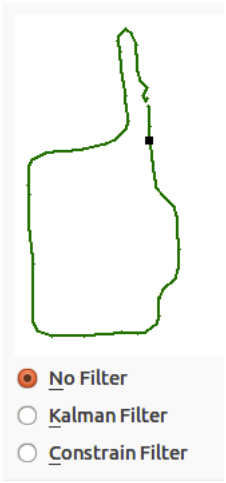


Figure 6.7: Live positioning of the vehicle, represented on the map.

Once the Live Session was finished, it was possible to start the offline analyses of the logged data. In this mode, the values from the gathered sensors were presented in a unique plot, where the GPS speed and altitude were also integrated. In this plot, it is possible to analyze the relation of the speed/altitude according to the time of the acquisition. The screenshot of this widget is represented at figure 6.8.

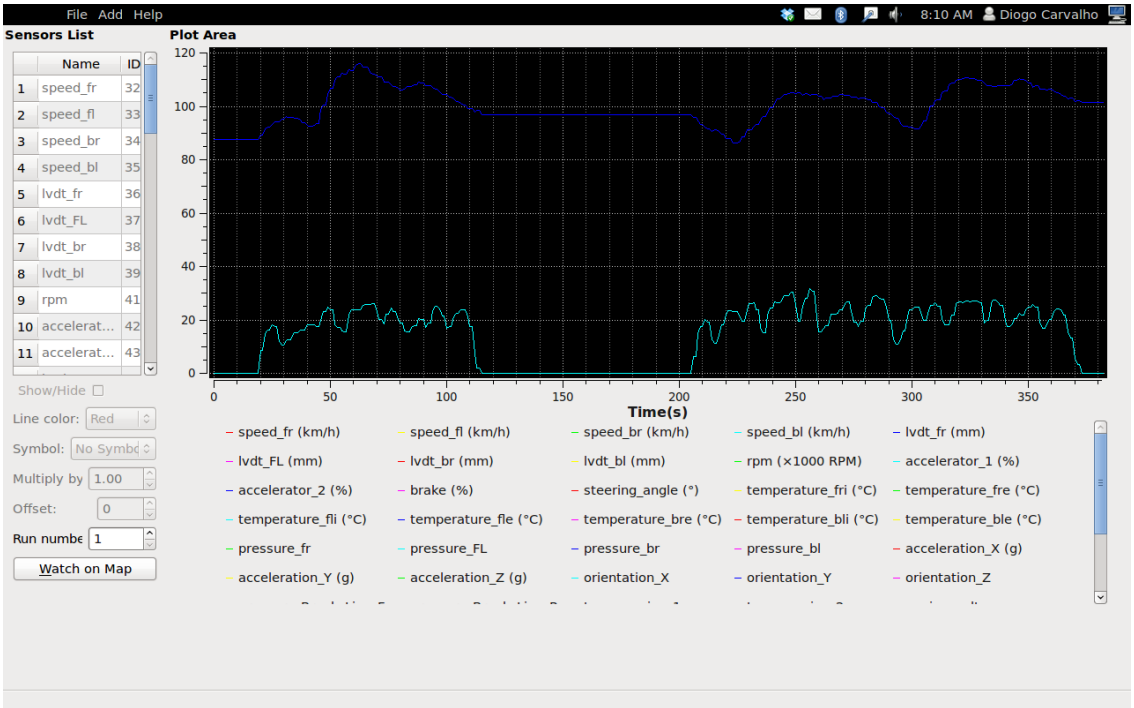


Figure 6.8: Plot representing the GPS altitude and speed.

6.4.3 Data Analysis and Filtering

To relate the values represented in the plot with the corresponding position, it was pressed the button ”Watch on Map” on the screen illustrated in fig. 6.8. The images corresponding to the offline mode are presented in the figures 6.9, 6.10 and 6.11. The represented images correspond to the same coordinates and speed values. They only differ in the chosen filtering algorithms: fig. 6.9 to the constrain filter mode, fig. 6.10 corresponds to the raw coordinates, and fig. 6.11 to the kalman filter mode.

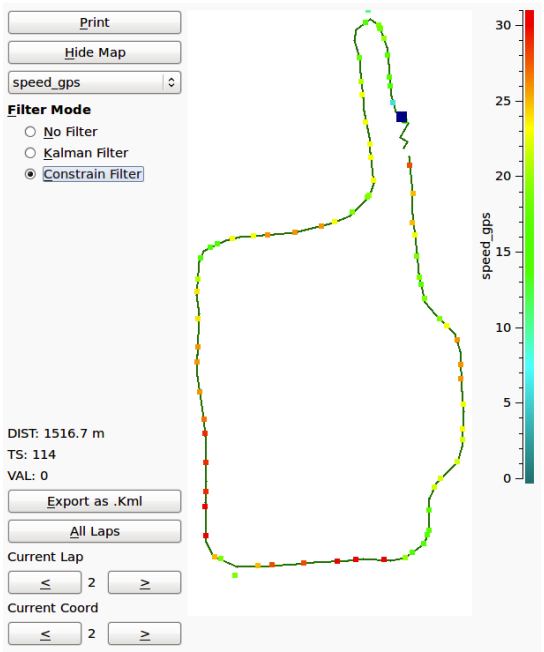


Figure 6.9: Acquired coordinates using the constrain filter.

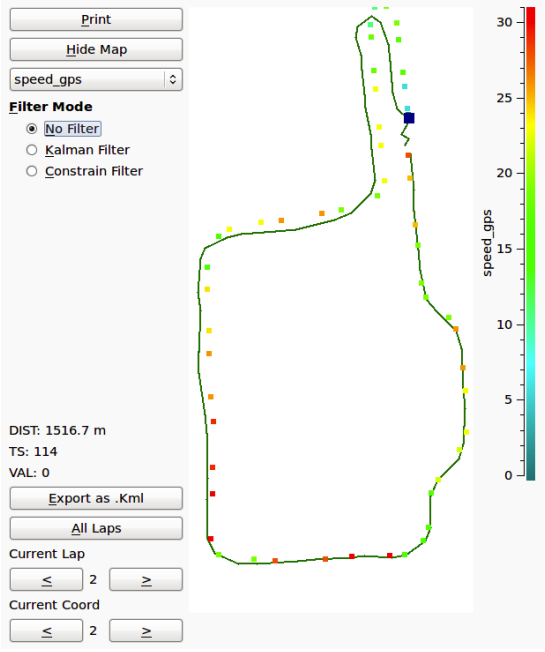


Figure 6.10: Acquired coordinates without any filtering.

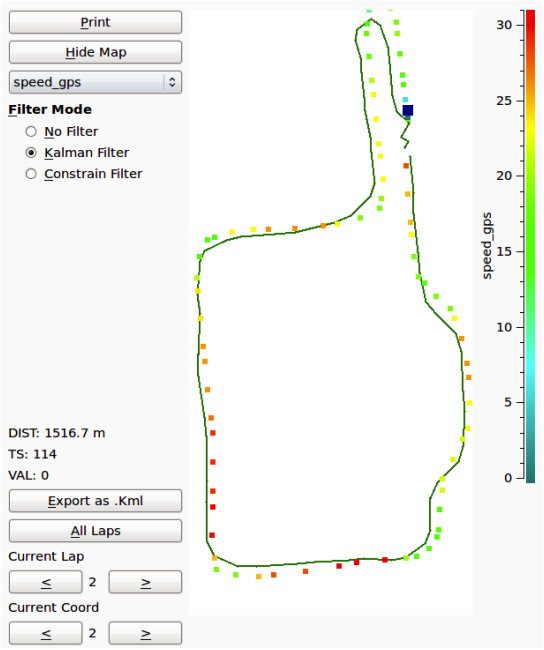


Figure 6.11: Acquired coordinates using Kalman filter.

By observing the three presented screenshots, it is possible to draw the following conclusions:

- Although the vehicle has passed in the same track, the representation in the map can be slightly different when compared with the previous lap.
- The kalman filter provided results more similar to those corresponding to the more accurate KML generated map.
- The constrain algorithm provides better results than those corresponding to the previous filtering modes. However, besides the fixes that have been introduced in order to guarantee the constraining of all the points, some tuning is still needed in order to eliminate the +/- 2% of the points that still not constrained in this lap.
- The distance calculated in the three laps corresponds to 1516.7 meters. By looking at the tool represented in figure 6.2, which calculated the value of 517.8 meters per lap, it means that both tools differ in 35.44 meters corresponding to 2% of the measured distance.
- The colors of the map are aligned with the characteristics of the circuit, meaning that the car moved faster in the straights (20km/h and the 30km/h, and more slowly in the curves with the values (10km/h and 20km/h). This indicates that the geo-positioning of the speed sensors are in accordance to what was expected by the team.

In this test, besides only three complete laps had been performed, there are available in this offline session four distinct laps to be consulted. The reason is that the last lap had crossed the virtual finish line of the map, starting a new lap with only few coordinates indicating that the this functionality is working according to that was expected.

The generated report from this session, containing all the laps, is present in Appendice.

6.5 Additional Tests

In order to measure the error margin of the calculated distance, as well as the relation between position and sensors, two other additional tests were made.

6.5.1 Distance Estimation and Sensor Mapping

The first test consisted in a trip of 41km through roads and motorways. Along this trip the speed values varied between 0 and 120 km/h, and the altitude from 33 until 312 meters.

This test was carried out by using a laptop computer running Linux Mint 12.04 operating system, which contained the base station and mobile station software. For the GPS device, it was used the bluetooth GSat hardware, described in Appendice A.

Along this test, it was collected 2343 points during 39m06s. The satellite images corresponding to a sample of the acquired data is presented in figure 6.12. This information was logged in the base station and subsequently extracted by using the KML exporter, present in this application. In the google maps distance tool represented in figure 6.13, it is possible to confirm the approximate distance of 42.9 km. The car's trip odometer was reset to zero at the beginning of the trip and marked a distance of 42.7 km at the end of the test.

Figure 6.12: Sample of an exported KML gathered data.

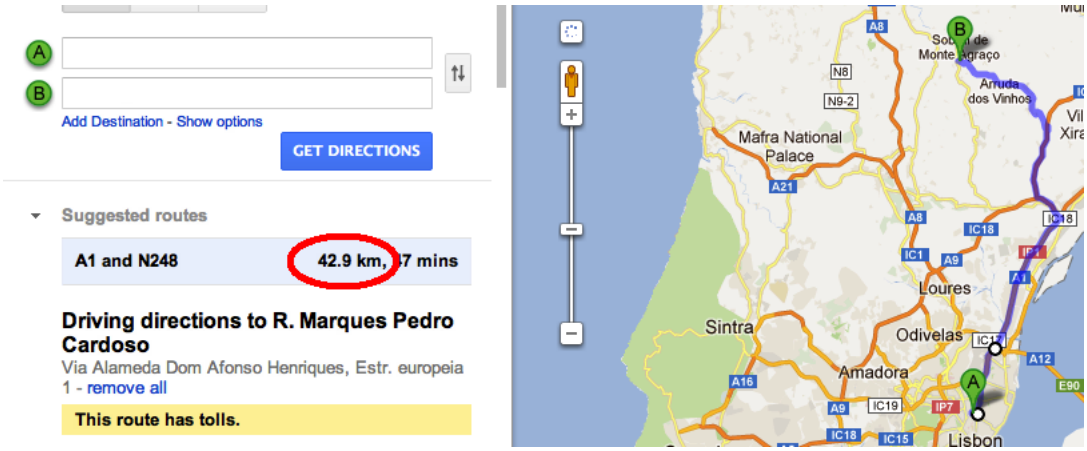


Figure 6.13: Test track with the traveled distance.

By relating the plot in figure 6.14, which represents the speed and altitude along the time, with the speed map (fig. 6.15) and the altitude map (6.16) it is possible to verify the consistency of the representation. In the speed map it is observed at the initial and final moments of the trip small segments represented with the color blue, indicating a low speed. By looking at the plot values and to the speed map it is possible to identify which part corresponds to the motorway and which part corresponds to a slower road. This information can be confirmed at figure 6.13.

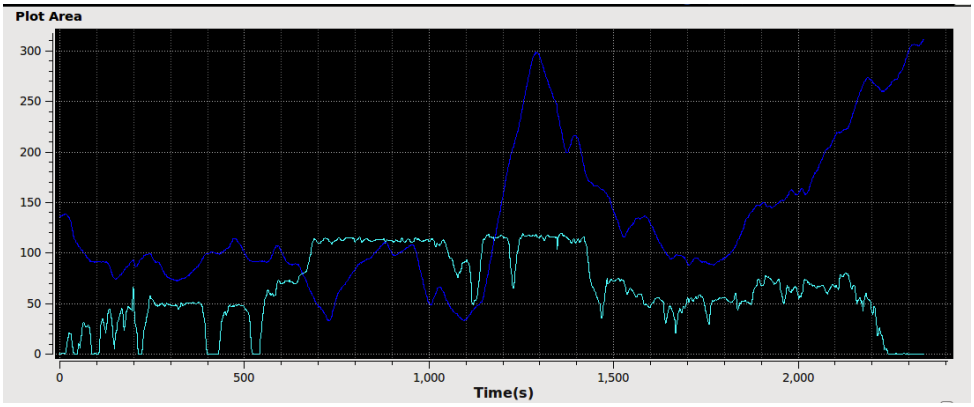


Figure 6.14: Altitude in meters (dark blue) and Speed in km/h (light blue) gathered from the GPS.

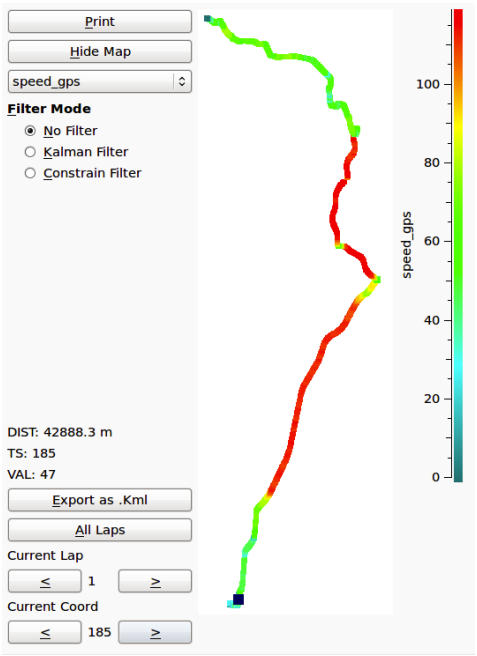


Figure 6.15: Mapping of the speed sensor in the track.

Besides the information presented in the map area, in the left side of the application it is present the calculated distance: 42888.3 meters which lies between the estimated value of the Google maps (42.9 km) and the car odometer (42.7 km). In this left bar of the application window it is also possible to navigate through the coordinates and observe the corresponding value and timestamp.

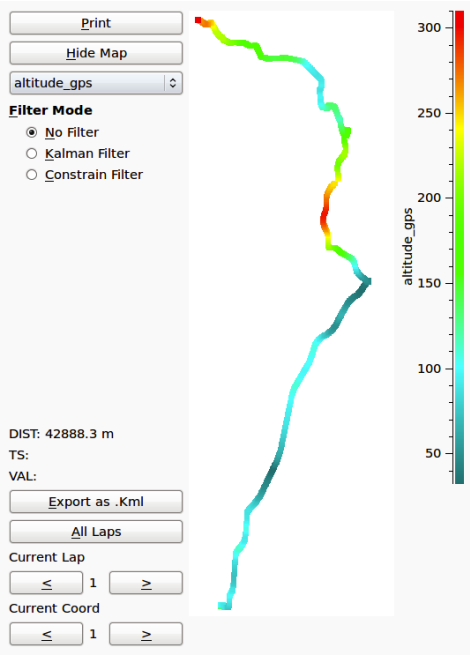


Figure 6.16: Mapping of the altitude sensor in the track.

6.5.2 Location timestamp gathering

By replacing the values corresponding to the several samples acquired by the set of sensors connected to the CAN-Bus of the car with the speed and altitude samples provided by the GPS, it was also possible to test the representation of the sensors georeferencing, since they share the same sensor timestamp. However, other test have to be done in order to confirm that the coordinates timestamp (which is sent by the mobile stationI is in accordance with the CAN bus sensor's timestamp. To evaluate this, the FST team re-configured the hardware present in the button of figure 6.17, corresponding to the hardware used to simulate the CAN bus sensors. This hardware is connected to the CAN-Bus acquisition device, represented in the top of the same figure, and sends messages consisting in random values and id's, with sequential timestamps. These messages are sent with a frequency of 10Hz to the processing unit of the mobile station, which is also receiving messages from the GPS device at 1Hz. An sample of this result can be observed in the following paragraph:

- `38;86;1462`
- `39;88;1463`
- `38.737937;-9.138606;104.610000;0.000000;1463;2012-05-12;23:39:44`
- `40;200;1464`
- `41;200;1465`

The short messages corresponds to values gathered from the CAN-Bus, the first element is the sensor ID, followed by the value and timestamp. The longer message (third) corresponds to the GPS message. From this demonstration, it is possible to conclude that the timestamps are

synchronized, since the GPS message has the same timestamp (5th value) than the previous message.

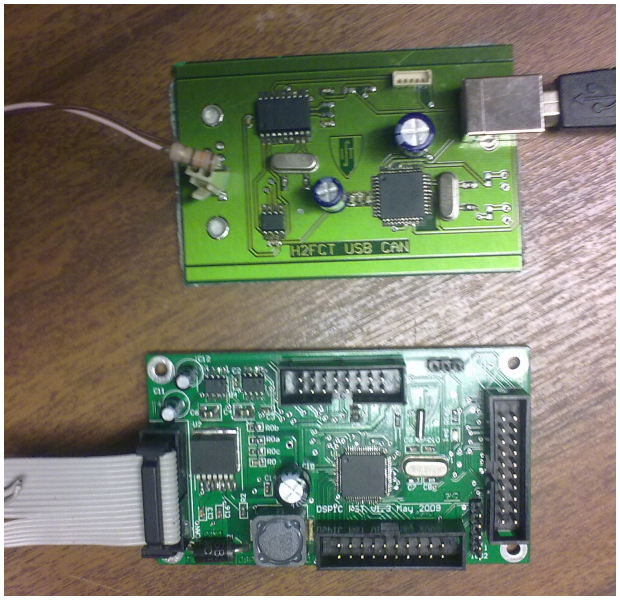


Figure 6.17: CAN-BUS to USB interface and Board used to simulate CAN-BUS sensors.

7

Conclusions and Future Work

Contents

The aim of this work is to add value to the existing telemetry system, by implementing a system which is able to detect the current position of the vehicle and also relate this position with the events occurred in the car (sensor values). The detection of the position has to be made in real-time, and the system must also allow the user to analyze the collected data after the race.

7.1 Conclusions

In order to accomplish with the aims of this work, the following steps had to be made:

- Integration of a GPS device with the mobile station.
- Integration of a GPS gathering tool with the data acquisition, logging and communication framework.
- Integration of a set of algorithms developed by an external team, in order to provide positioning filtering, and map constraining, which significantly improved the accuracy and representation of the car positioning.
- Integration of a set of Matlab libraries to support the implementation of the algorithms described above.
- Integration of the user interface.

To support all of this integrations, it was designed an architecture based on a modular system, with the aim of providing a functional integration and an easily maintenance. For the implementation was also developed: several adaptors in order to interface different technologies; an location module to manage the processment of the GPS coordinates; and finally a module responsible for the entire application which interacts with the user through a developed user interface, and manages the state of the application.

According to the introduction of this document, the objectives defined for this work are:

1. **Create a real-time trajectory and tracking system, that provides the position of the car to the pit-stop team, with good accuracy.**

   The implemented system is able to acquire the position from the mobile station and to send the gathered coordinates to the remote base station. An example of the positioning representation is presented in figure 6.7.

   The system also allows to save the car trajectory and speed information for further analyses, which is demonstrated in figures 6.9 and 6.11. It is also possible to observe the previous positions where the car had been, and the application of the processing algorithms used to improve the accuracy.

2. **Create a georeference system that relates the data from the car sensors with the place where the data were collected.**

   In this application it is possible to consult the information gathered in the previous sessions, by loading an existing session. After loading, it will be presented one plot containing the

sensor values. It will be also presented a map, where is possible to select the data to be analyzed related to the position. The results are represented divided by laps, in the map drawn, or by individual values while the user navigates through each coordinate. In figures 6.15, 6.16 and 6.10 is represented some examples of the geo-referencing of the speed and altitude sensor information according to the position.

3. **Integrate the information concerning to the position of the car in the base station screen, together with the existing widgets that represent the other sensors.**

   The user interface integration can be observed both in Live Mode represented in 6.5 and 6.6, where the widgets appear in the same screen. For the Offline Mode the widgets appear separately due to the amount of information presented in each one. (see fig. 6.14 and 6.15)

4. **Integrate this solution with the existing telemetry and communication platforms, already existing in the car.**

   The integration of this solution in the existing telemetry system can be expressed as three types of integration:

   - User interface integration, by developing a widget which will run along with the other sensors widgets. (see figures 6.5 and 6.6)
   - Communication and database integration, by using the existing communication platform and the interaction with the existing database.
   - Information integration, by treating the GPS values, such as speed an altitude, like the CAN Bus sensors values. In the Offline mode it is also used the CAN Bus sensors values, in order to be related in the new widget and analyzed by the user. (see figures 6.14 and 6.15)

7.2 Future work

This telemetry project has started some years ago, and its being improved year after year by adding new functionalities such as video and GPS, or improving the existing ones like communication platform. Since this is a constant work in progress application, in this section it will be presented some functionalities that can be implemented in the future.

Relating to the positioning, this system is providing a location platform both in hardware and software. With this system it is possible to get and represent the position every second, but for some aspects this may be not enough. The most simple way to improve the position data collection is by using a GPS with a higher rate without loosing accuracy, but this will traduce in a more expensive hardware. Other solutions are provided in the following paragraphs:

- By integrating gyroscopes an accelerometers in the vehicle with higher frequencies than GPS, with the data collected by this sensors is possible to calculate the intermediate positions between the GPS coordinates. This can be done using the kalman filter, by placing the calculations of these sensors in the predicting step of the filter and then when the coordinate information is collected by the GPS this will correct the position in the filtering step.

This solution would increase the number of information about positioning available, and will also increase the precision of the car.

- Other work that could be done in this subject could be implementing a DGPS like system, requiring an additional GPS receiver and placing it in a known position. With the information of the position the fixed GPS could calculate the error of the position calculation and correct the mobile GPS with this information. This solution will improve the precision, but will not solve the low frequency problem.

- In the user interface some improvements concerning the organization of the widgets can be made. The user interface is composed by separated and independent widgets. These widgets are floating on the screen, and they can be moved independently or also can be grouped in one unique window (docked). Docking and organizing the widgets in the position which the user wants is not a user friendly process, is time-consuming and after restart the application the process of organization must be remade by the user, since it will restore to the original position. So the proposed work will consists in improve the user interface, concerning to the organization of the widgets and allow to save the definitions (like size and position of the widgets) in user profiles which can be loaded in the beginning of the application. This will allow the user to set up his dashboard as he wants once, and using it multiple times.

In the Offline session, the collected data can be analyzed with more detail using a plot which represents all the sensors values and a map where is correlated the position with the values form each sensor. The usability and the data analysis can be improved by:

- Augmenting the number of widgets available and the interoperability with them. The user should be free to choose between gauges, bars, and graphs to represent the data. With a set of widgets available, they could be synchronized to show a snapshot of a given moment selected by the user. For instance: having a plot representing the relation between speed and time, where the user could use a bar to navigate between the plot along the time, when he is navigating the steering wheel, the accelerator bar and the map could be synchronized by moving and present the state of each sensors like in live session.

- Adding an formula editor, where the user can perform operations with and between the sensors. It should be possible to create graphs with the difference between the values of two sensors for instance. This editor could accept basic operations (+, -, / , *) with constants, available sensors and allow the user to choose how the result could be presents: in a bar, table, plot, etc.

Bibliography

[1] "Globalsat technology corp," http://www.globalsat.com.tw/, last visit: 7/12/2010.

[2] "Projecto formula student team técnico," http://www.projectofst.com, last visit: 7/11/2010.

[3] "Gps error budget," http://www.sxbluegps.com/gps-error-Budget.html, last visit: 30/04/2012.

[4] "Journal of neuroengineering and rehabilitation," http://www.jneuroengrehab.com/content/2/1/28/figure/F2, last visit: 7/12/2010.

[5] "Racelogic products specification," http://www.racelogic.co.uk/, last visit: 9/11/2010.

[6] P. F. Mendes, "Formula student racing championship: design and implementation of the management and graphical interface of the telemetry system," Master's thesis, Instituto Superior Técnico (IST) Universidade Técnica de Lisboa (UTL), 2010.

[7] "Formula student," http://www.formulastudent.com/, last visit: 7/11/2010.

[8] "James r. clynch, earth coordinates," http://www.gmat.unsw.edu.au/snap/gps/clynch_pdfs/coorddef.pdf, last visit: 8/05/2012.

[9] "Gps history, chronology, and budgets," http://www.rand.org/pubs/monograph_reports/MR614/MR614.appb.pdf, last visit: 7/12/2010.

[10] "U.s.government information about the global positioning system," http://www.gps.gov, last visit: 1/06/2010.

[11] P. Simplício, N. Santos, A. Costa, and J. E. Sanguino, in Uncoupled GPS Road Constrained Positioning Based On Constrained Kalman Filtering, October 2010.

[12] "European spacial agency, egnos," http://www.esa.int/esaNA/GGGQI950NDC_egnos_0.html, last visit: 8/05/2012.

[13] P. Enge, R. Kalafus, and M. Ruane, "Differential operation of the global positioning system," Communications Magazine, IEEE, vol. 26, no. 7, pp. 48 –60, Jul. 1988.

[14] Y. Morales and T. Tsubouchi, "Dgps, rtk-gps and starfire dgps performance under tree shading environments," in Integration Technology, 2007. ICIT '07. IEEE International Conference on, 2007, pp. 519 –524.

[15] R. E. Kalman, "A new approach to linear filtering and prediction problems," Transactions of the ASME-Journal of Basic Engineering, vol. 82, no. Series D, pp. 35–45, 1960.

[16] W. Li and H. Leung, "Constrained unscented kalman filter based fusion of gps/ins/digital map for vehicle localization," in Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE, vol. 2, 2003, pp. 1362 – 1367 vol.2.

[17] "Novatronica products specification," http://www.novatronica.com/, last visit: 7/12/2010.

[18] "Garmin products specification," http://www.garmin.com, last visit: 7/12/2010.

[19] "Tomtom products specification," http://www.tomtom.com, last visit: 7/12/2010.

[20] "Motec products specification," http://www.motec.com.au, last visit: 7/09/2011.

[21] "Simbin games catalog," http://www.simbin.se/games/gtr.htm, last visit: 7/09/2011.

[22] "Motec interpreter manual," http://www.motec.com/filedownload.php/Interpreter_Manual_A5.pdf?docid=1084, last visit: 4/11/2011.

[23] "National marine electronics association," http://www.nmea.org, last visit: 07/12/2010.

[24] NMEA Reference Manual, Revision 2.1. •, 2007.

[25] "Gps daemon documentation," http://gpsd.berlios.de, last visit: 7/11/2010.

[26] M. St-Pierre and D. Gingras, "Comparison between the unscented kalman filter and the extended kalman filter for the position estimation module of an integrated navigation information system," in Intelligent Vehicles Symposium, 2004 IEEE, 2004, pp. 831 – 835.

[27] "Nokia, qt framework," http://qt.nokia.com/products/, last visit: 18/05/2012.

[28] "Matlab deploy tool documentation," http://www.mathworks.com/help/toolbox/compiler/deploytool.html, last visit: 8/12/2010.

[29] "Qwt library," http://qwt.sourceforge.net/, last visit: 8/12/2010.

[30] "Gps fake documentation," http://gpsd.berlios.de/gpsfake.html, last visit: 7/12/2010.

[31] "Kml to csv converter tool," http://chooncherlim.com/kmlcsv/, last visit: 7/05/2012.

[32] "Matlab c compiler documentation," http://www.mathworks.com/help/toolbox/compiler/mcc.html, last visit: 8/12/2010.

[33] "Sqlite database browser," http://sqlitebrowser.sourceforge.net/, last visit: 7/05/2012.

Appendix A

Contents

A.1 GlobalSat BT-359W Bluetooth GPS Receiver Specification[1]

Figure A.1: GPS Test Module

Electrical Characteristics	GPS Chipset	SiRF Star III
	Frequency	L1, 1575.42 MHz
	C/A Code	1.023 MHz chip rate
	Channels	20 channel all-in-view tracking
Accuracy	Position Horizontal	10 meters, 2D RMS
		1-5 meters 2D RMS, EGNOS/WAAS corrected
	Velocity	0.1m/sec
	Time	1 micro-second synchronized to GPS time
Datum	Datum Default:	WGS-84
Acquisition Rate	Hot start	1 sec., average
	Warm start	38 sec., average
	Cold start	42 sec., average
	Reacquisition	0.1 sec. average
Protocol	GPS Protocol Default:	NMEA 0183 (Secondary: SiRF binary)
	GPS Output format	GGA(1sec), GSA(1sec), GSV(5sec), RMC(1sec), GLL, VTG is optional
Dynamic Condition	Acceleration Limit	Less than 4g
	Altitude Limit	18,000 meters (60,000 feet) max.
	Velocity Limit	515 meters/sec. (1,000 knots) max.
	Jerk Limit	20 m/sec**3
Bluetooth Specification		
Electrical Characteristics	Bluetooth Chipset	CSR BC4
	Frequency	2402MHz to 2480MHz
	Standard Bluetooth	V2.0
	Bluetooth Profile	SPP (Serial Port Profile)
	Operation Range	10 meters (33 feet)
	Output Power	0 dBm (class II)

Figure A.2: GPS Module

A.2 GlobalSat ND-100 GPS USB Dongle [1]

Feature	Item	Description
Chipset	GSC 3f	SiRFStarIII single chip
General	Frequency	L1, 1575.42 MHz
	C/A code	1.023 MHz chip rate
	Channels	48 all-in-view tracking
	Antenna	Built-in ceramic patch antenna (18 x 18 x 4mm)
Accuracy	Position	10 meters, 2D RMS
		2.2 meters CEP without DGPS
		¡5meters(50%), DGPS corrected
	Velocity	0.1 meters/second
	Time	1 microsecond synchronized to GPS time
Datum	Default	WGS-84
	Other	selectable for other Datum
Time to First Fix (TTFF)	Reacquisition	less than 1 sec., average
	Snap start	1 sec., average
	Hot start	1.5 sec., average typical
	Warm start	32 sec., average typical
	Cold start	34 sec., average typical Protocol
	Protocol	NMEA 0183 v3
	GPS protocol	GGA(1sec), GSA(1sec), GSV(5sec), RMC(1sec),
		GLL,VTG is optional
	GPS transfer rate	38400
	USB Protocol	USB 2.0
TTFF		
Dynamic Conditions	Altitude	18,000 meters (60,000 feet) max.
	Velocity	515 meters/second (1000 knots) max.
	Acceleration	4g
Power	Main power input	5.0V, USB bus power.
	Supply Current	$\approx$ 55 mA
Environmental Characteristics	Operating temperature	range -20^o C to +60^o C
	Storage temperature	range -30^o C to +70^o C
Physical Characteristics	Length	65.5 mm
	Width	23 mm
	Height	11 mm(with 4 mm Antenna)
	Weight	20g

B

Appendix B

Contents

B.1 Example of a Generated Report

Session id: 59

Sensor: speed_gps

Session distance: 3416 m