Universitat de les Illes Balears

escola politècnica superior

Bachelor Final Thesis

GRAU D'ENGINYERIA ELECTRÒNICA INDUSTRIAL I AUTOMÀTICA

Vehicle data acquisition and display system applied to motorcycles

MIQUEL FONT MAS

Tutors

Gabriel Oliver Codina Miquel Massot Campos

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, September 8, 2017

To my family, for their unconditional love.

ABSTRACT

The automotive industry, increasingly focused on intelligent driving, requires systems able to obtain data from the vehicles in order to perform data analysis.

This trend does not appear with the same strength in the world of motorcycling, where analogue systems and non-standard communication networks are frequent in the industry, which prevents users and amateurs from studying and understanding their own transport machines. Furthermore, it should be taken into account that most motorbikes do not have proper displays to visualise parametric values of the vehicle such as speed, acceleration, inclination, motor, wheel and ambient temperatures, and so many others.

Some products can be found in the market, but the high cost of these solutions makes them non-viable for limited budgets. As it can be seen, there is a need for affordable and easy-to-use data acquisition devices, which could be used to modernise old machines or to obtain data for amateur competitions.

The main goal of this project is the study and design of a data acquisition system based on Arduino, ready to be installed in any vehicle, regardless of its date of manufacture. The project will be focused on motorcycles, so systems based on gyroscopes and accelerometers will be used to obtain the vehicle tilt. This projects also includes a real-time viewing system based on Android, that offers the user a intuitive graphic interface.

SUMARI

La industria automobilística, cada vegada més centrada en sistemes de conducció intel·ligent, requereix de sistemes de presa de dades dels vehicles per al seu tractament i anàlisi.

No obstant, aquesta tendència no apareix en el món del motociclisme aficionat, on els sistemes analògics i les xarxes de comunicació no estandaritzades són habituals i dificulten als usuaris del motociclisme esportiu l'estudi del seu mitjà de transport. A més, cal remarcar que moltes motocicletes no compten amb sistemes de visualització de paràmetres del vehicle, tals com puguin ser la velocitat, l'acceleració a l'hora de frenar, la temperatura del motor, de les rodes o de l'ambient, entre altres.

Els productes que es troben al mercat són de cost elevat, dificultant l'accés a persones amb recursos limitats. Ja sigui per actualitzar i modernitzar vehicles antics, com per oferir noves dades de conducció per competició amateur, queda palesa la necessitat de cobrir aquesta veta de mercat.

El principal objectiu d'aquest document és l'estudi i disseny d'un sistema de recollida de dades basat en Arduino que es pugui instal·lar a qualsevol vehicle, amb independència de la seva data de fabricació. Aquest projecte es centrarà en motocicletes, i per tant es farà ús de sistemes basats en giroscopis i acceleròmetres per determinar la inclinació del vehicle, entre altres sensors. A més, es desenvoluparà un programa per dispositius mòbils, que permeti visualitzar les dades del vehicle en temps real amb una interfície gràfica intuïtiva.

Resumen

La industria automovilística, cada vez más centrada en sistemas de conducción inteligente, requiere de sistemas de toma de datos de los vehículos para su tratamiento y análisis.

No obstante, esta tendencia no aparece en el mundo del motociclismo aficionado, donde los sistemas analógicos y las redes de comunicación no estandarizadas son habituales y dificultan a los usuarios del motociclismo deportivo el estudio de su medio de transporte. Además, hay que remarcar que muchas motocicletas no cuentan con sistemas de visualización de parámetros del vehículo, tales cómo puedan ser la velocidad, la aceleración a la hora de frenar, la temperatura del motor, de las ruedas o del ambiente, entre otros.

Los productos que se encuentran en el mercado son de coste elevado, dificultando el acceso a personas con recursos limitados. Ya sea para actualizar y modernizar vehículos antiguos, como para ofrecer nuevos datos de conducción para competición amateur, queda patente la necesidad de cubrir este nicho de mercado.

El principal objetivo de este documento es el estudio y diseño de un sistema de recogida de datos basado en Arduino que se pueda instalar a cualquier vehículo, con independencia de su fecha de fabricación. Este proyecto se centrará en motocicletas, y por lo tanto se hará uso de sistemas basados en giroscopios y acelerómetros para determinar la inclinación del vehículo, entre otros sensores. Además, se desarrollará un programa para dispositivos móviles, que permita visualizar los datos del vehículo en tiempo real con una interfaz gráfica intuitiva.

Contents

Acronyms

ABS Anti-lock Braking System

bps Bits per second

BT Bluetooth

CCTV Closed-circuit Television

CES Consumer Electronics Show

CoM Centre of Masses

DAD Data Acquisition Device

DIY Do It Yourself

DMP Digital Motion Processor

DoF Degree of Freedom

ECU Electronic Control Unit

EDR Event Data Recorder

EoF End of frame

FPGA Field-Programmable Gate Array

GPRS General Packet Radio Service

GPS Global Positioning System

HMI Human Machine Interface

I/O Input/Output

IoV Internet of Vehicles

OBD On-Board Diagnostics

OEM Original Equipment Manufacturer

RPM Revolutions Per Minute

RTC Real Timer Clock

RX Reception

SoF Start of frame

SPP Serial Port Profile

TS Telematics System

TX Transmission

UART Universal Asynchronous Receriver/Transmitter

UI User Interface

YPR Yaw, pitch and roll

CHAPTER 1

INTRODUCTION

Each passing day vehicles are equipped with more technology: ABS, air-bags, traction control, collision avoidance systems, and so on. To measure and control the different parameters of the vehicles a huge amount of sensors have been designed [1]. These improvements have made our transport means to be more reliable, assisting the driver through passive and active safety systems, whose purpose is the assistance and avoidance of accidents and reducing its consequences.

Furthermore, the current trend of connecting all the devices has also affected the vehicle industry, named as Internet of Vehicles (IoV). The main goal of IoV is to use the data acquired from a huge amount of vehicles and road devices (such as CCTV cameras, road conditions sensors and traffic lights) in order to manage traffic with more ease [2].

1.1 Motivation

To achieve the goal of IoV, all vehicles should have a data logging system connected to the cloud. This arises some problems, not only because most vehicles do not have internet connection, but old vehicles do not even have any accessible data to the user, either because of not having electronic control unit (ECU) and sensors or because the diagnosis interface does not follow any standard.

Besides, this constraints the development of accessories only to the original equipment manufacturer (OEM), reducing the aftermarket. Imagine that our vehicle's speedometer is broken, then the only solution is to buy a new one from the manufacturer; but if the manufacturer doesn't have (because it is a very old vehicle, the manufacturer is bankrupt and no longer exists or even because there is no technical assistance in the area) the user will have no means to update and replace the component.

To avoid these issues, the On Board Diagnostics II (OBD-II) and Controller Area Network bus (CAN) standards have been adopted in the car industry. However, not all the automotive industry has followed this path, specifically the motorcycle industry has several diagnosis and interfaces standards depending on the OEM or even the product.

Some brands, such as Harley Davidson (since 2013), fully support CAN and OBD-II. But the majority of the brands support CAN standard but with proprietary wiring and electric signals schemes, and their own diagnosis port (instead of OBD-II). This is the case of BMW [3], Yamaha [4] and Honda [5].

The main issue about current motorcycles lies on the inaccessibility of data and the poor use of sensors. Solving this problem is the first step to enhance bikes' dashboards, providing better information to the rider. Furthermore, if the data is stored, it can be used in case of accident to analyse the causes and even simulate what happened. This last use is known in the automotive industry as event data recorders (EDR), or black boxes.

1.2 Aims and scope

As mentioned in the introduction, the goal of this thesis is to develop a system to acquire vehicle data for its later display and storage. Taking into account the context of the project, we will focus our research in motorcycles. Nevertheless, our development should also be suitable with some modifications to any type of vehicle.

It is possible to divide this general goal into more specific objectives as follows:

- **Research about available technologies:** A primary study of the solutions available in the market is required in order to develop a system up to date. It is also of our interest to analyse the approaches adopted in different branches of the automotive industry and the implementation of these in competitions.
- **Development of a Telematics System:** Design a platform composed by a given number of sensors and a data acquisition device (DAD), which is the core of the project.
- **Design a Human Machine Interface (HMI):** Develop a real-time viewer for key performance indicators following safety measures, not to disturb the driver but to give at-a-glance information, to be installed next to the motorcycle dashboard.
- **Create a storage service:** Being able to record trips is the key to analyse driving behaviour and use the system as a black box. The goal pursued is to storage the data for later access.

1.3 Outline

The different topics introduced in this thesis are distributed in six chapters, structured as follows.

Chapter 2: State of the Art. This focuses on the study of the state of the art in telemetry systems for vehicles. Firstly, common schematics will be introduced and then uses of the information obtained will be presented. To finish this chapter, some competitors alternatives will be shown and analysed.

Chapter 3: Framework. Technologies available at the current time will be introduced, in order to explain the component selection and configuration of our product. Later on, the electric and mechanical schematics will be shown.

Chapter 4: Implementation. In this chapter, the implementation of our system will be discussed. Mainly we will focus on three topics: the telemetry device, the dashboard unit and the storage service.

Chapter 5: Field Trials. This presents all the experiments performed throughout this thesis. Additionally, the results obtained and the lessons learned will be explained.

Chapter 6: Financial Analysis. In this chapter the viability of the project will be evaluated. The cost of the prototype and the market price will be calculated.

Chapter 7: Conclusions. This section summarises the results achieved and its future development.

CHAPTER 2

STATE OF THE ART

Since the early development of the automotive industry, new technologies have been introduced in order to improve the driving experience, safety and reliability of vehicles. The IoV has pushed the limits of vehicles' connectivity allowing an easy access to the its information.

In order to accomplish this goal, new technologies have been developed. In this chapter we will introduce them and study how they are being implemented in motorcycles.

2.1 ECU and OBD-II

To improve the performance of internal combustion engines, some device able to control key functions had to be developed. These devices, called electronic control unit (ECU), substituted the old mechanical and pneumatic means that dynamically controlled ignition and idle speeds.

Nevertheless as time has passed, the increasing number of sensors used in other tasks has established ECU as the controller of almost all parts of the vehicle, such as brake and steering system. This evolution from engine control to a global vehicle control, introduced a new definition of ECU: an embedded system that controls one or more of the electrical system or subsystem in a transport vehicle.

In order to simplify diagnosis and viewing of the data, OBD-II network was developed. OBD-II network connects the main sensors and the ECU, and outputs the information through the interface from Fig. 2.1. The OBD-II is an standard that can be used with different communication protocols, the most important of them is CAN. Since 2008 in EU, USA and almost all Asia have adopted CAN and OBD-II as the standard in cars.

Having such a connector allows any user to link his own vehicle with another device (a smartphone, for instance), which can be used to diagnose the vehicle, driving analysis or to send information to IoV.

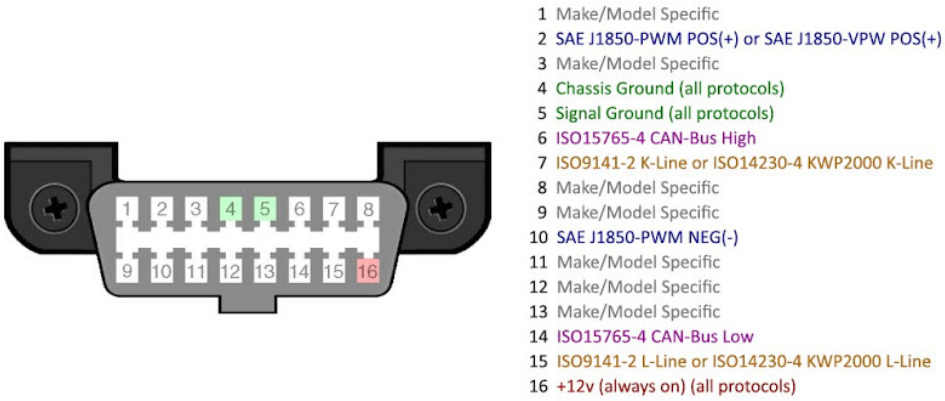


Figure 2.1: Data Link Connector: OBD-II port

2.2 Historical development of motorcycles

The earlier stages of motorcycle series production began at the end of the 19th century. With First and Second World War, motorcycles production ramped up in order to replace horse riding couriers, making it a very popular mean of transport [6].

In order to meet emission standards in the United States and in Europe at the very early 1980s, manufacturers had to improve engines' performance. The carburetors that were being used could not handle the combination of performance and emissions at their then-current state of development. To fulfil the new requirements, OEM began using electronic fuel injection systems (EFI). At this stage, EFI were very simple systems, in order to be cheap and easy for their mechanics to learn. The introduction of EFI meant the introduction of ECU to control it [7].

Despite having motorcycles with ECU since the beginning of the 80s, there is no standard data link connector adopted by the industry, as opposed to the car industry where OBD-II is used by all new cars. As mentioned in the introduction chapter, this last decade (2010) some brands started using OBD-II as the diagnostic port. Nevertheless, it is not very common to find this scenario.

2.3 Market solutions

To solve some of these problems, aftermarket solutions have been proposed. In this section the most important ones will be analysed.

2.3.1 Bosch mySPIN

At the beginning of 2017, Bosch (one of the pioneers to develop stability and control systems and ABS for motorcycles and so many more) introduced at CES [8] a connectivity suite for motorcycles. This system interconnects the motorcycle's onboard brain to the bike's digital dashboard and also the user's smartphone with a technology called mySPIN [9].

This connectivity suite with the Bosch two-wheeler technology [10] is the most advanced consumer product available in order to obtain bike's data.

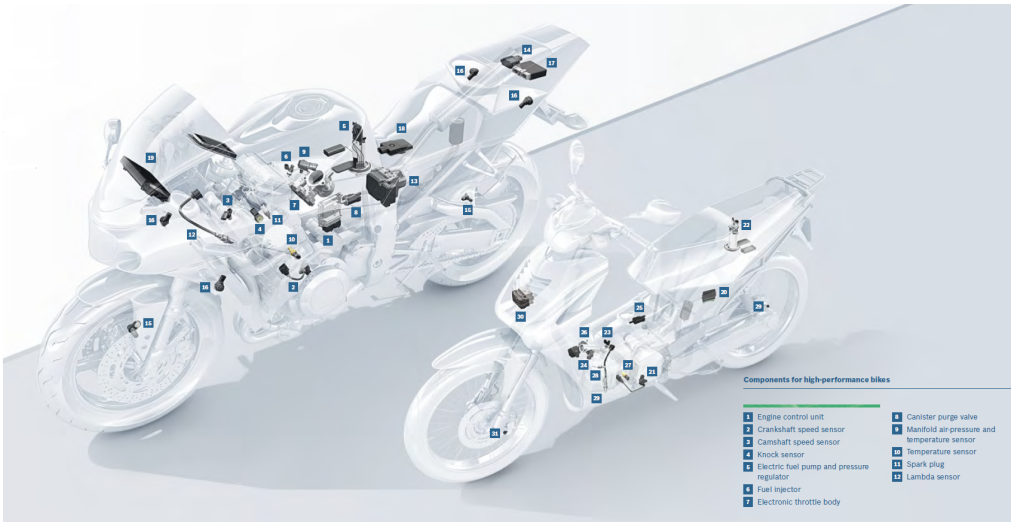


Figure 2.2: Schematic of two motorcycles with Bosch sensor technology

As it can be seen in the figure 2.2, the different sensors from the Bosch two-wheeler technology suite are connected in order to provide an accurate reading of the state of the vehicle. Then, this data is sent to the display and to the smartphone.

Nevertheless, this technology is only available in a limited amount of new and expensive motorcycles. Furthermore, this technology is installed in the factory by the OEM at the manufacturing process, so it is only available for new motorcycles. And due to this system high cost, manufacturers only install it in high-end machines.

2.3.2 Midas Connect

Another similar approach is the one adopted by the american chain of automotive services Midas. This company has developed a system that can be used in any car, connecting its OBD-II connector to a proprietary device, which obtains the data from the CAN bus and external sensors (such as GPS) and sends it to a smartphone with the Midas Connect app [11]. Currently, the price of this product (installation included) is affordable for the consumer, around 60€.

The main issue of the Midas system is how to collect the data. Given that the principal source of information is the OBD-II, it is not suitable for most motorcycles.

2.3.3 Bicycle solutions

As we have seen previously, the common problem with the solutions studied is the use of information from the OBD-II. For this reason, it is interesting to evaluate solutions proposed in vehicles with no ECU or OBD-II: bicycles.
Recently, a lot of start-ups have come up with different ideas to connect the bicycles with the smartphones, which can be used as a dashboard as well.

COBI One of the most important exponent of this trend is COBI [12], a product that consists of a phone holder with integrated lights with a wireless connection to some other devices (such as tail light, turn indicators and a controller). Basically, the phone

Figure 2.3: Screenshot of the KDS dashboard developed by Thomas Riebmann

is running an application that controls the external devices, while obtaining speed, position and slope from the phone sensors. Despite of being a simple device, the cost of the system is around 300€.

Connected Cycle Anther approach is the one used by the french company Connected Cycle [13], which is based on a phone as a dashboard and getting the data from the GPS, accelerometer and GPRS from a pedal that it is installed in the bike.

2.3.4 DIY

In the last decade, the interest in solving problems through technology by hobbyist and amateurs has increased, creating a phenomenon known as Maker Culture. Makers have also came up with different solutions to obtain data from vehicles.

OBDuino OBDuino is an open source on-board electronic gauge based on Arduino. This project uses the OBD-II interface to get the data, and then it displays the information requiered by the user, such as speed, RPM, fuel consumption, trip and more [14].

KDS to Bluetooth This project goal, based on OBduino, is to submit the data from a Kawasaki Diagnostic System (KDS) via Bluetooth in order to record videos with driving information, as in MotoGP [15]. A screenshot of the recorded video with the KDS information can be seen in the figure 2.3.

Chippernut Shift Light Tachometer The open-source sequential shift light developed by Chippernut is one of the most known projects in the hobbyist world. It has been used in cars and driving simulators to display RPM, due to its simple installation:

only a connection between the RPM Input wire to the ECU or aftermarket ignition system is needed [16].

2.4 Project grounds

The solutions available in the market do not satisfy our needs. On the one hand, most systems are designed for specific models of motorcycles, with new technologies (CAN and OBD-II for instance). Moreover, the systems available are usually proprietary and not expandable, not allowing easy data access to the user. Imagine, that we want to have in our dashboard a tachometer or a light sensor. With these kind of systems we are limited to what the company offers. Lastly, these solutions are very expensive.

It is clear the need to develop an open-source project with a reduced cost, accessible to everyone, expandable and suitable any motorcycle.

CHAPTER 3

FRAMEWORK

This chapter will be focused on the framework used to accomplish the objectives from section 1.2. First of all, the structure and requirements of our system will be presented. And then the selection of components will be discussed.

3.1 Introduction

As stated in previous chapters, the goal of this project is not to gather data from the ECU, but to complement it with new sensor information. Thus, by not depending on the ECU data, the solution to develop should be suitable for any kind of vehicle: cars, motorcycles, bicycles, etc.

Nevertheless, this project will be focused on the development of a system for motorcycles. To do so, data will be obtained from a given number of sensors: at minimum we should have data about the inclination, temperature and position of the vehicle. This data will be processed by the acquisition device and then it will be saved in the available storage and, if possible, uploaded into a server. Simultaneously, this data will be displayed in the HMI, providing to the rider useful information of the bike.

3.1.1 System description

As explained in section 1.2 three main components of the project are defined: a telematics system, a HMI and a data logger. This structure can be seen in Fig. 3.1, where the telematics system is composed by the DAD and the sensors.

In the block diagram we find the so called *cloud services*, these are all the services that are provided over the internet, which comprises cloud storage, analysis of riding, emergency assistance, and many more. As this is not the object of the project, it will be left for further research and development.

Nevertheless, we will have two different configurations depending on the complexity of the system:

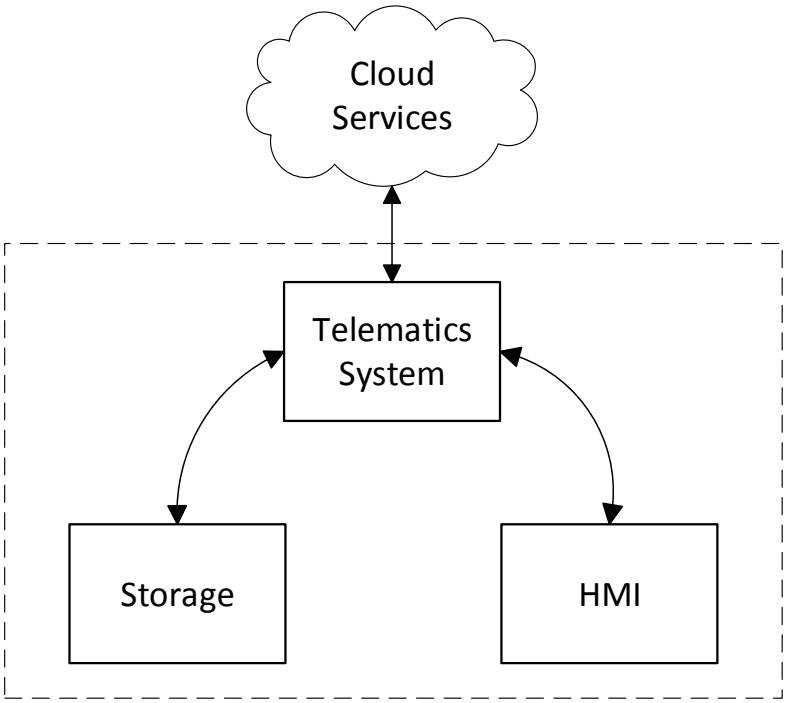


Figure 3.1: Block diagram of the system

Basic configuration It includes the telematics system and the HMI, but not the data logger. This configuration has a lower cost and it is intended for the day to day driver, who is not interested in saving riding data nor its analysis.

Extended configuration In this configuration all three parts are present, so we are dealing with a more complex product whose cost is expected to be higher.

3.1.2 Requirements and limitations

The proposed system needs to address a number of constraints. As seen in section 2.3, telemetry systems available in the market are normally proprietary and designed for a specific vehicle. Nevertheless, DIY projects have in mind a more global use, providing general solutions that can be adapted to the specific requirements of each case. This is the adopted approach in this project, where the adaptability of the system is a priority. It has to be expandable and even modular.

Another factor to consider is the cost of the whole product, which should be affordable for motorcycle owners. This not only affects the cost of the components used, but also the installation. In order to accomplish that, the product should be easy to install for any customer, so no technical service will be required.

Given the low storage capacity of motorcycles, the last limitation is found in the size of the device.

3.2 Telematics System

The telematics system (TS) is composed by the sensors and the DAD. Its main task is the gathering of vehicle information. It will be installed in a fixed location of the vehicle, and connected to a power supply, which can be the battery of the vehicle. Thus, the power consumption should be as low as possible.

3.2.1 Data Acquisition Device

The DAD is the core of the whole system: it receives data from the sensors and transforms the signals to a comprehensive measurement to the user. Then it sends this data to the HMI and to the data-logging system. There are several platforms to carry out this task, of which the most important ones are:

Smartphone application

Smartphones are devices widely adopted and used, due to its portability and large amount of convenient features. Smartphones are very capable devices, with a high computing power and great connectivity, supporting a large amount of communication standards, such as WiFi, Bluetooth (BT), NFC, GPRS, 4G... They are also equipped with a wide number of sensors.

The main advantage of smartphones is the cost, because the customer does not need to buy an extra device, as he can use his day-to-day phone.

Nevertheless the biggest issue regarding smartphones is the difficulty to add peripherals. This restricts its configuration to the default one, not being able to increase the number of sensors. Another problem of using smartphones is the battery consumption, which is very high. Also, turning on and off is very slow given that the whole operating system must load.

As we can see, smartphones are not very well suited to develop the task of a DAD, specially for the issues regarding to expansion of peripherals.

Raspberry Pi

Raspberry Pi is a tiny low-cost computer, with most common ports: USB, Ethernet, HDMI, audio, MicroSD slot and GPIO header [17]. The main advantage of a Raspberry Pi is its computing power, and its expandability. Moreover, the community support is very notorious, and a lot of projects already developed are open source. The boot up time is very short, and the power consumption is relatively small (circa 400-500 mA, 2.5W [18]).

We consider this device very appropriate to perform as a DAD.

Arduino

According to Massimo Banzi, co-founder of the Arduino platform and author of *Getting Started with Arduino* [19], Arduino is an open source physical computing platform based on a simple input/output (I/O) board and a development environment. Arduino boards are microcontrollers, to which external modules can be added depending on the project. The low price and the fact that they are open source hardware have makes

these boards market competitive. Despite not being very powerful devices (compared to smartphones or Raspberry Pi), Arduino boards are very reliable and very good at obtaining data from sensors. Its energy consumption is very low: according to the Arduino webpage [20], when active, Arduino boards consume around 20mA.

In addition, the size is not an inconvenience because from the several boards available there are very small ones, and they can be added easily to homemade electronic boards (due to its DIY philosophy).

Finally, a competitive advantage of Arduino is that they are real-time control systems.

FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer. Despite being very popular in the industry due to its performance, the difficulties to program them make them not very suitable for projects where adaptability is a key feature [21].

3.2.2 Sensors

The data processed by the DAD will come from a given number of sensors. In this section we gathered information about sensors that could match the purpose of the project, as they provide extra information about the state of the motorcycle.

IMU

An inertial measurement unit (IMU) is an electronic device that combines accelerometers and gyroscopes in order to measure the angular rate and forces, and thus angular position and lineal acceleration.

In our project, an IMU will be used in order to obtain the tilt of the motorcycle and the slope, and also to obtain the deceleration when braking.

In the market there is a wide range of IMU, whose prices can be either low or even cost thousands of Euros, depending on their characteristics.

Magnetometer

Magnetometers are electronic devices that measure magnetic fields, normally used as compasses. In our project a magnetometer could provide an extra degree of freedom (DoF). Combining it with the IMU we can achieve the 3D orientation model of the vehicle.

Barometer

A barometric pressure sensor will be used to obtain the pressure and the altitude of the vehicle.

GPS

Global Positioning System (GPS) is a global navigation satellite system that provides geolocation and time information to a GPS receiver. It can be used to obtain not only the vehicle position, but also the speed, orientation and time.

Thermometers

On the one hand, having knowledge of the DAD's temperature will provide feed of the proper operation of the device.

On the other hand, it is also from our interest to obtain wheel temperature. Knowing the wheel temperature provides interesting feedback as the vehicle behave changes: the grip of tire depends directly on the temperature of the rubber. Nevertheless, a regular temperature sensor can not be used because the wheel is always turning and no contact can be done. Therefore an infrared (IR) temperature sensor should be used.

3.3 HMI

The HMI is the bridge that allows communication between the user and the TS. We need a bidirectional communication.

In one hand, the HMI should provide feedback about the current status of the vehicle, displaying the information obtained by the TS. We should notice the importance of the safety measurements to adopt, in order to avoid any kind of distraction to the user while riding. For this reason, the data provided has to be understandable at a glance, and the interface has to be very user-friendly.

On the other hand, the HMI should allow some user input in order to perform the calibration when installing.

To accomplish both tasks, a display fixed next to the dashboard and some input device are required. These can be two different components, but also a single one: for instance a touchscreen.

3.4 Data Logging

Being able to record the information gathered by the DAD can be very useful to perform riding analysis and to have a black box in case of accident, to name some of the many possible applications.

The data logger should store the information from the TS, and a time stamp as well.

3.5 Component selection

After the study of the the components that conform our project we will proceed to the selection of each of them.

As it will be explained with more detail, in this project we will make use of a Arduino as the DAD. This Arduino will be wirelessly connected to a smartphone, which will be used as the HMI and will provide some sensor data as well. Some other sensors will be attached to the Arduino.

3.5.1 Telematics System

DAD: Arduino Nano and Mega

A very easy solution for our project would be using a smartphone as the core of the system. Nevertheless, being not able to expand the number of sensors easily (to add engine temperature for instance) and the high power consumption makes them not the very best system to be used.

FPGA on the other hand are very difficult to setup, limiting the use cases of our system.

The best platforms to be used as the core of our system are Arduino or Raspberry Pi. The main advantage of Raspberry Pi is its computer power, which at the same time makes it to drain more energy. Moreover, Arduino are very well suited to perform easy tasks such as receive data and process it.

We decided that an Arduino board will be the controller used, due to the previous reasons and the cost of it.

Due to limitations of flash storage and size, the controller will very depending on the configuration from section 3.1.1.

For the basic configuration, we decided to use use an Arduino Nano, whose size suits the space limitation of motorcycles. Nevertheless, as it will be explained in Chapter 4 the flash storage is rather short, being only 30 kB accessible to the user.

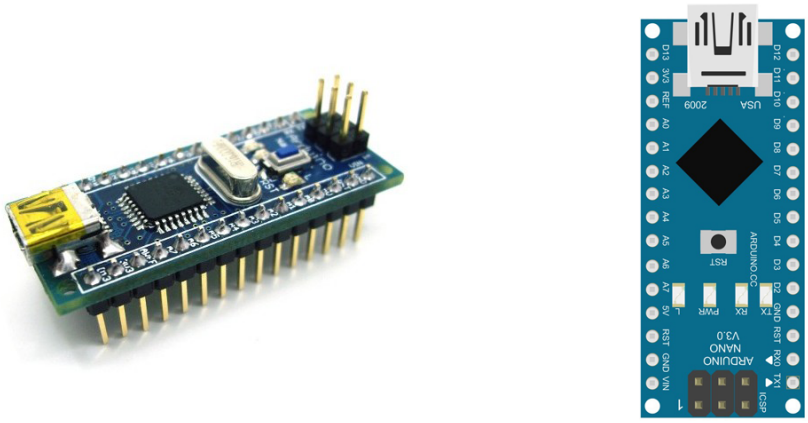


Figure 3.2: Arduino NANO

For the extended configuration, more storage is required since more tasks will be performed. We decided that the Arduino Mega is well suited, not only because of the storage but also the large number of I/O, which can be used for further development. Being one of the most used and accessible Arduino boards, it is very easy to find it for a very affordable price.

GY-87 Breakout Board

The GY-87 breakout board will be used. This inexpensive breakout board combines an IMU, a magnetometer and a barometer providing 9 DoF. The components used are the:

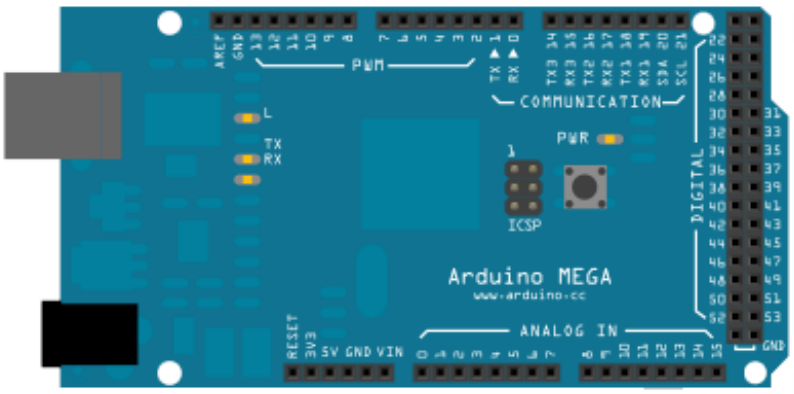


Figure 3.3: Arduino Mega

MPU6050 This IMU from Invensense is one of the most popular in the hobbyist world. It combines a 3-axis gyroscope and a 3-axis accelerometer with an on-board Digital Motion Processor (DMP) capable of processing complex 9-axis motion fusion algorithms. The gyroscope full-scale range of ± 2500 to ± 2000º/s, while the accelerometer's goes ±16g.

HMC5883 This surface mount chip from Honeywell is designed for low field magnetic sensing. It is used by the MPU6050's DMP to improve the results obtained.

BMP180 The digital barometric pressure sensor of Bosch Sensortec is the barometer used. In addition to the pressure, it is also capable of obtaining the temperature; which will be used to know the DAD's temperature.

This breakout board uses the Inter-Integrated Circuit (I^2C) communication bus.

Wheel Temperature Sensor

Given the high cost of the IR temperature sensors available in the market, we decided not to include one in our TS in order to maintain the price to the minimum.

GPS

After doing a lot of research, there is no affordable GPS in the market: all of them cost more than the expected price of our product. To overcome this inconvenience, we decided to use the one already available on the HMI. This limits the acquisition of geolocation data to only when the phone is connected to the system. We accepted this drawback in order to keep our product as a low-cost system.

3.5.2 HMI

The problem with displays (specially touchscreens) is the cost, which is very high in comparison to the overall cost of the project. For this reason a smartphone will be used as a HMI. Smartphones are devices more than capable of handling graphics, and given

that almost everyone has one, the cost of it can be considered null because it won't be used exclusively for the motorcycle dashboard purpose.

The platform of development is Android, currently the most used mobile operating system. An application will be developed to be the portable motorcycle dashboard.

TS-HMI connection: Bluetooth

As stated previously, we will use an Android device as a display, which will receive data from the TS. Therefore a connection should be established. To simplify the wiring, a wireless communication will be used. Several wireless protocols could be used, which are supported and available in the Android ecosystem: GPRS, WiFi, Bluetooth and NFC.

The distance between both devices is very short, around 1 or 2 meters, depending on the vehicle. The GPRS is not intended for this use and is more expensive because an extra phone plan has to be used. As NFC is intended for shorter ranges and low traffic data will not be useful either. Bluetooth and WiFi are both good solutions. Nevertheless, the communications will be done through Bluetooth, because when a WiFi connection is established in Android, it is expected to receive and send data through it instead of 4G or GPRS, which we may want to use for other tasks.

The interface used in the TS is the HC-06 module, a very affordable breakout board that supports Bluetooth class 2 with a low power consumption. This model it can be used only as slave, if we needed to use it as master we would use the HC-05. The module will be connected to the Arduino board using Serial communication.

3.5.3 Data Logging

Given that Arduino have a very limited memory we will use an external SD to record the information. We will use a SD breakout board which uses the Serial Peripheral Interface bus (SPI). It is important to emphasise the advantages of using the Arduino as the data-logger instead of the Android device: it is not mandatory to have the phone connected all the time, being able to work as black box. Moreover, having this configuration makes very easy to develop HMI applications in other platforms, such iOS, Windows or even for a Raspberry Pi.

Furthermore, in order to have a data-logging system we do not only have to store the sensor information, but also add a time-stamp to this data. Arduino boards can not provide a time stamp, because their internal clock is initialised to 0 each time the controller boots. In order to overcome this shortcoming, a Real Time Clock module (RTC) will be used together with the datalogger.

In this project the DS1307 module will be used. This affordable RTC communicates to the Arduino via I^2C.

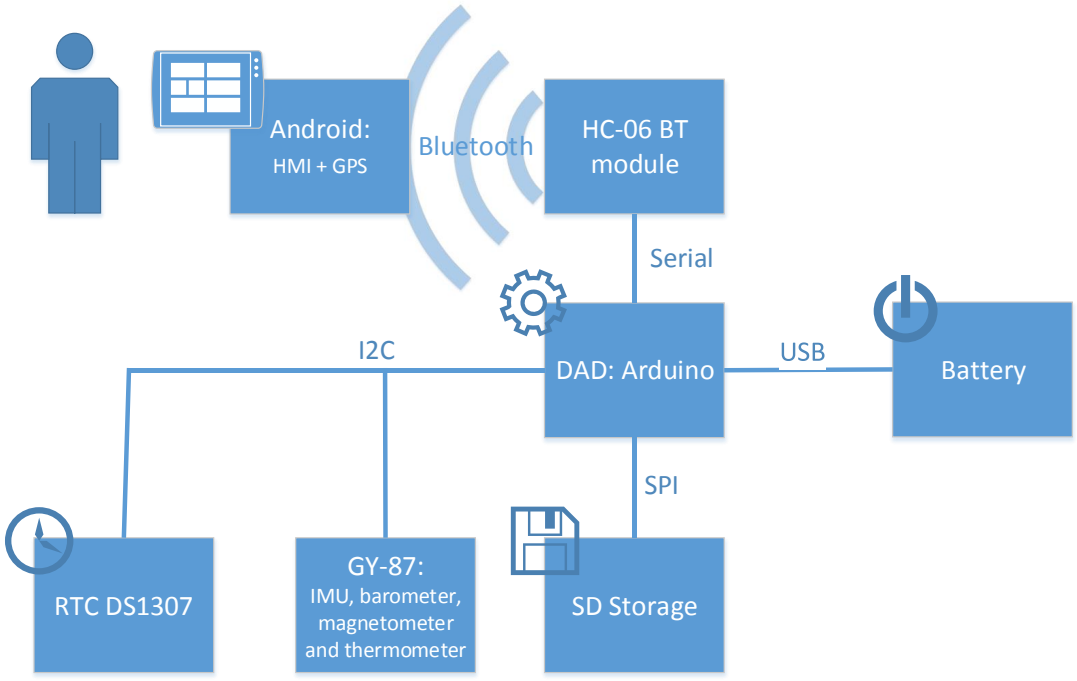


Figure 3.4: System overview

CHAPTER 4

IMPLEMENTATION

This chapter is divided in four different main parts. Firstly the implementation of the DAD will be discussed. Later on, the development and structure of the Android application will be introduced. The hardware configuration will then be presented. At the end of the chapter some considerations regarding the implementation and issues found during the whole process will be explained.

4.1 Introduction

One aspect that was taken into account when considering the different implementation options was the ease of use by the customer. Therefore, when the user starts using our platform, he should not be concerned about technical issues or even having to program anything. So, only one Arduino sketch must be load into the end product and it must include all the functions to satisfy the customer requirements. The same happens with the Android application, which must be unique and easy to install. This fact implies that despite having two platform configurations, as stated in 3.1.1, only one Android application should be available and compatible with both products.

4.2 Telematics system and data logger

Despite having two configurations available, both of them are very similar and so the core code is the same. For instance, the structure of the code, the variables and functions must be as similar as possible, making easy to maintain and update both products. Having a similar structure also makes possible to use the same HMI version, as the data received will have the same structure.

Before going into further detail, let's stop to name and explain the main features and functionalities the TS should have to ensure a proper user experience: the main objective of the TS is to gather data from the IMU and other sensors. Nevertheless, to do so the IMU must be calibrated so the data obtained is meaningful under our frame of reference.

This calibration should only be done once when the TS is installed on the bike, or after changing the placement of the DAD in the vehicle. Despite the sporadic use of it, to simplify the user experience it will be an option available at any moment to the user. Therefore, a calibration function will be found in the sketch installed in the DAD.

To use the Bluetooth module, it must be previously configured. Nevertheless, this configuration should not be available to the user, and it should be done only before assembling all the modules together. For this reason we have developed an independent sketch to perform this single task (see 4.2.1).

These and the other features will be discussed in more detail in their corresponding sections.

Now let's elaborate upon the structure of the program. The following diagram (Fig. 4.1) presents a basic concept of the task-flow carried by the DAD. Note that the asterisk (*) symbolises those tasks related to the data logger feature, and so to be performed only by the extended configuration.

As it can be seen, the system boots when the motorcycle is on, so the maximum amount of data is obtained. To achieve so, it is recommended to use the vehicle battery through the main switch as the power supply (with a 12V to 5V converter in between) when installing the TS.

Then, the system will setup. If the extended version is being used, a logging file will be created in the appropriate folder path. In case this path does not exist, it will automatically create the folders required. The date and time will be obtained from the RTC module. If we are using the basic version, this step will be omitted.

Once created the logging file, it will be check if any new income BT message has been received. If so, it will proceed as stated in the extended Specification and Description Language (SDL) diagram (Fig. 4.2). In this diagram we can see something called *Cmd*, as it will be later explained in section 4.2.5, it is a variable

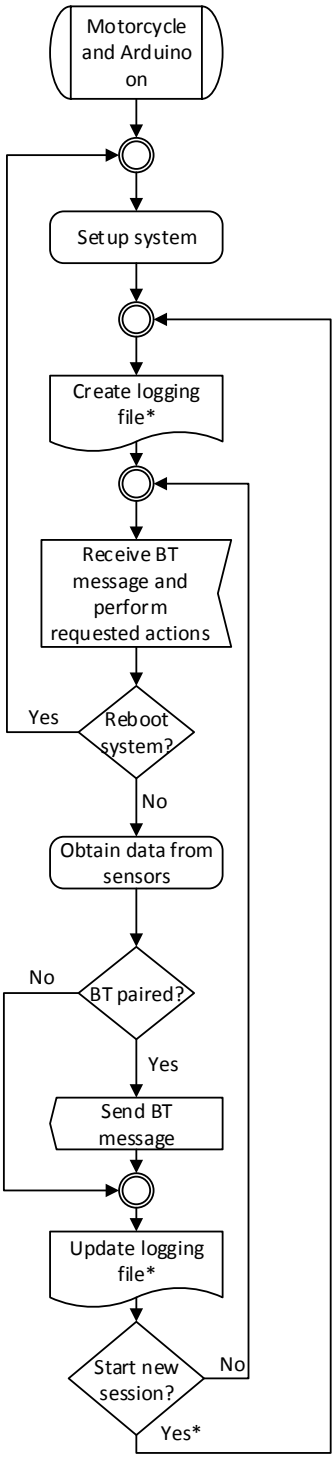


Figure 4.1: Arduino SDL diagram overview

that informs the user about the current state of the controller, which is sent to the HMI (for debugging purposes for instance).

Depending on the message received (see section 4.2.5), the program will:

- **Pair BT:** Indicates that a new connection has been successfully established. Once the device is paired the system will suppose that remains paired. The only way to unpair the device is to reboot the system.
- **Reboot the system:** This feature is very useful for debugging purposes and to reset the machine in case of failure. Before reseting the Android, a message will be sent to the user, and after the reset another message will be send as a confirmation of the successful procedure.
- **Update GPS data:** Gets the latitude, longitude and speed information from the GPS module, which is the Android device. Is only available in the expanded version, due to the lack of interest of knowing this data if it is not stored.
- **Calibrate IMU:** Performs a calibration of the IMU (see 4.2.2). During the calibration the IMU should be still, because any slight movement will interfere in the result of the calibration. To inform the user, a message will be sent to the HMI pointing the beginning of the calibration. When the calibration is performed, another message will be send. This last message will be send with the new data gathered, so it will wait until the next programmed sending. Notice that in the following loop a new logging session will be created in order to preserve accurate and consistent measurements during the whole logging session.
- **RTC synchronisation:** Receives the timestamp from the HMI in order to update the registers from the RTC (see 4.2.3). Take into account that the DS1307 IC accuracy is not very high due to time drifting, which can be as high as 5 seconds per day. So if we want to have an accurate time-stamp when logging it is recommended to perform a weekly clock synchronisation. Notice that in the following loop a new logging session will be created in order to preserve accurate and consistent measurements during the whole logging session. It is only available in the extended version, because no time-stamp is required in the basic configuration.
- **Create a new session:** It creates a new logging session after finishing this last loop. Only available at the extended version, because no logging sessions are available in the basic configuration.
- **Empty message:** Does nothing. It has been created for debugging purposes only.

After that the DAD will proceed to gather all the data from the IMU and the barometer (see 4.2.4). Then all this data will be send to the HMI (see 4.2.5) and stored to the SD logging file (see 4.2.6). Finally, depending on the received BT message we will create a new log file or repeat the loop from the receiving BT message point.

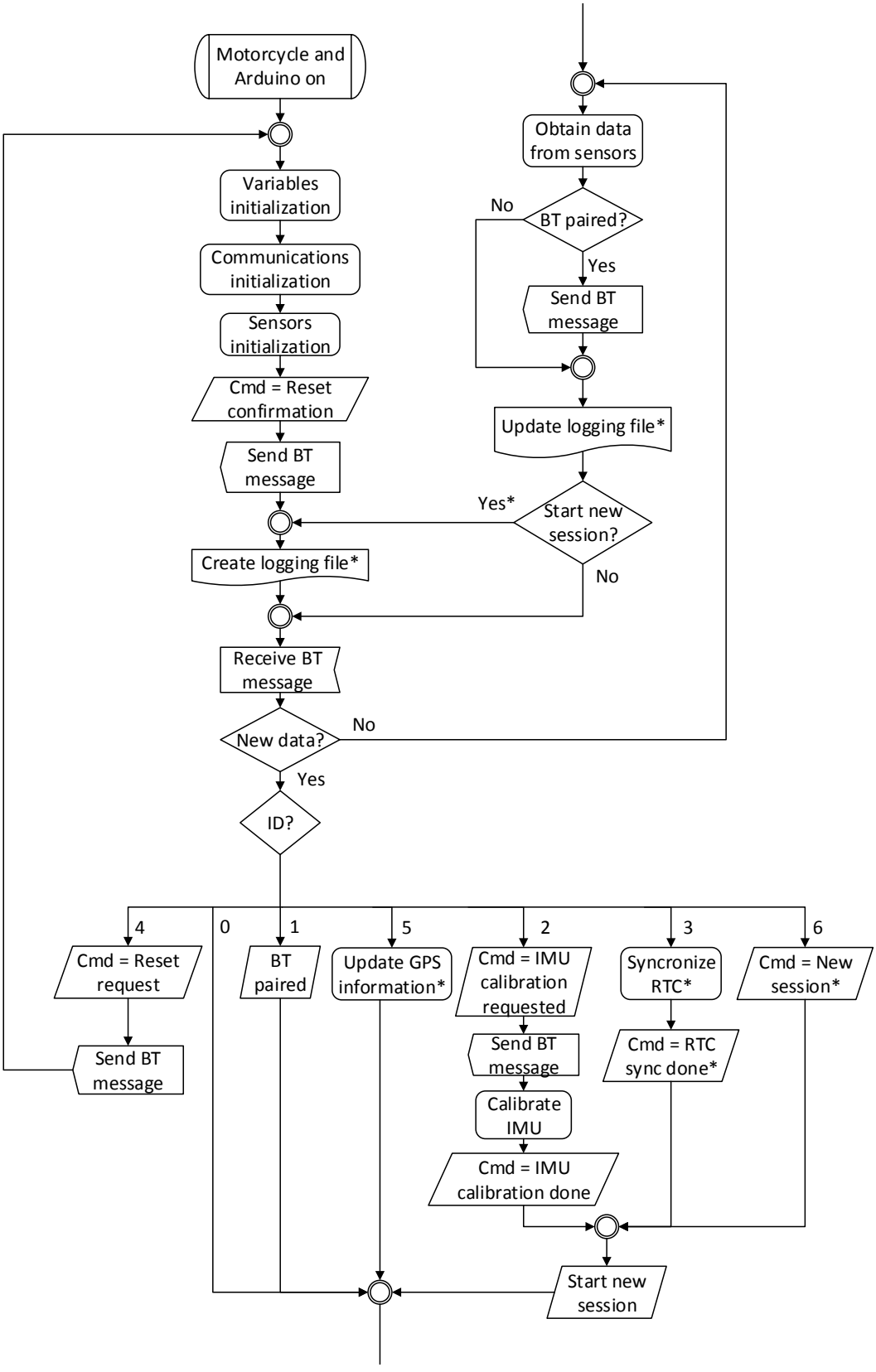


Figure 4.2: Detailed Arduino SDL diagram

4.2.1 HC-06 configuration

As explained, before being able to use the BT module, it must be configured. In order to do so, a setup sketch has been developed.

This BT module (HC-06) communicates to the Arduino through serial communication. Using this type of communication we can send AT commands to the module. It is important to notice that depending on the breakout board, the HC-06 could respond differently to our instructions.

The AT commands that we use to configure the BT module are [22]:

- **"AT":** Starts the configuration mode of the module. After this message, the module will expect to receive AT commands.

- **"AT+NAME":** After sending this command a character array of maximum 10 bytes will be sent (remember that in Arduino, 1 char occupies 1 byte). This array must contain the desired name of the BT device, which will be seen for the others BT devices around. In our case we chose to name the BT module *PONIZ*.

- **"AT+BAUD":** After this command we will send a byte with the baud rate that we want the BT device to operate. The baudrate specifies how fast data is sent over the serial line, and it is expressed in units of bits-per-second (bps) [23]. 8 different baud rates can be setup: 1200 bps (the content of the byte is 1), 2400 bps (2), 4800 bps (3), 9600 bps (4, which is the default one in the BT module), 19200 bps (5), 38400 bps (6), 57600 bps (7) and 115200 bps (8). As we want to send data as fast as possible because our HMI is very capable of receiving and processing data at high speeds, we will use the maximum baud rate supported by the module: 115200 bps, so we will send a byte containing the character 8.

- **"AT+PIN":** It expects a 4 byte array containing the PIN to connect with the module. We are using the PIN "0000".

For proper operation it is recommended to wait 1 second between AT commands, but not between command and operator. After a couple of seconds without command, the BT module will leave the configuration mode.

4.2.2 IMU calibration

When placing the IMU we will make sure that the y axis is fixed facing the direction of the movement. We want to achieve a system as the one shown in Fig. 4.3, nevertheless we must notice that the position of the IMU does not require to be in the origin of the motorcycle's coordinate system, because due to being a solid, the yaw, pitch and roll (YPR) will be the same, which are the data we want to get. The acceleration in the 3 directions is also a interesting value to obtain, and because of the previous reason it does not matter where we place the IMU (notice that we simplify the analysis supposing that the vehicle it is not subjected to any flexure).

Nevertheless, we need to make sure that in rest position the values obtained from the YPR are zero, or very close. To do so, a calibration algorithm has been designed.

Figure 4.3: Motorcycle's coordinate system

This algorithm (see Fig. 4.4) is designed to find the offset of the gyroscope and accelerometer by iteration means. It starts setting the 6-axis offsets at certain values. Then it obtains the mean of 100 measurements. If the mean of these values are all in the deadzone (close enough to zero to suppose it is zero) we finished the algorithm and the last offsets used are the required. Else other offset values will be used and the process is repeated until we get the desired offsets. The iteration values are picked following the recommendations from Luis Ródenas and Jeff Rowberg, developer of the MPU6050 library and the I2Cdev library as well [24].

This process can take as much as 30 seconds. To avoid any movement it is send a reminder to not touch the vehicle from the still position while the algorithm is running, as explained in section 4.2.

Once the calibration has been successfully performed we will store the values of it to the EEPROM of the Arduino. When initialising the controller and the sensors, we will access to these registers to set the offsets of the IMU. Given the fact that the bike and TS are not going to change after the installation, no more calibrations are required. In case we change the vehicle or place the TS in another position of the vehicle we should perform another calibration, whose new offset values will replace the previous from the EEPROM.

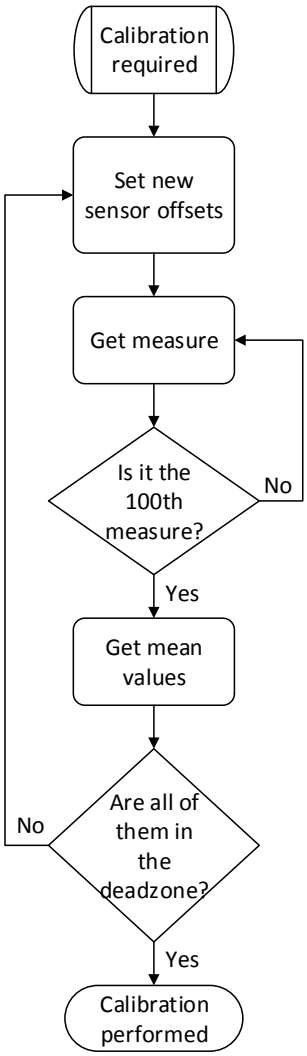


Figure 4.4: IMU calibration algorithm

4.2.3 RTC adjustment

As explained before, the main issue with the DS1307 RTC is time drifting, which after several month or even years can distort greatly our results. To avoid this problem, an easy synchronisation function has been developed. Android devices have their own internal clock, which is updated constantly through the network, therefore it is very accurate and reliable. When connected as HMI, the user will see the option of updating clock. If selected, a new BT message will be sent from the Android to the Arduino. This message will contain a time stamp which will be updated to the RTC.

Notice that a small delay between both devices will be introduced during the synchronisation. This delay is the result of the time required to obtain the time stamp from the Android (which is not 100% accurate), the time to transmit the frame, the time required to process it by the Arduino (which is variable) and the time needed to update the DS1307. Nevertheless, as this delay is very small (from the order of milliseconds at maximum) we will neglect it.

4.2.4 Sensor data acquisition

To interact with all the sensors from the GY-87 breakout board, a library has been designed. This library includes a function which is in charge of gathering all the data and store it in a register.

This register contains the last readings of the sensors, therefore it is checked to send the BT messages and update the logging file. This register is an array of 11 floats. In previous versions, the YPR was transformed from radians to degrees, in order to store a meaningful value, but this conversion took too long and slow the whole Arduino program. For this reason, we decided to use the default format which is radians stored as floats, so no operations are required.

The function to obtain the data first gets 32 values from the buffer of the MPU6050 DMP, to obtain the YPR and acceleration in the 3-axis. Then a low-pass filter is used in order to remove the noise from the measurements, which are mostly caused by motor vibrations. The low-pass filter used is an average of 32 values. This introduce a little bit of delay, so the resulting value does not represent the current state but a past one. The values obtained will be stored in the register according to the structure shown in Fig.4.5.

Then the temperature from the IMU will be stored as well, after converting the value obtained to Celsius degrees as stated in the MPU6050 datasheet.

The next step is gathering the data from the pressure sensor, from which the temperature, pressure in hPa and Atm and the altitude will be obtained. Notice that the temperature must be always the first value requested, to avoid issues with the sensor which calculates the pressure relying on the temperature.

4.2.5 Bluetooth messages

In this section we will discuss how the incoming messages are addressed and the process to send a BT message to the HMI. We will focus on the structure of these messages. For more information about the processing of the data contained in incoming messages refer to section 4.2. For the outgoing messages processing see section 4.3.

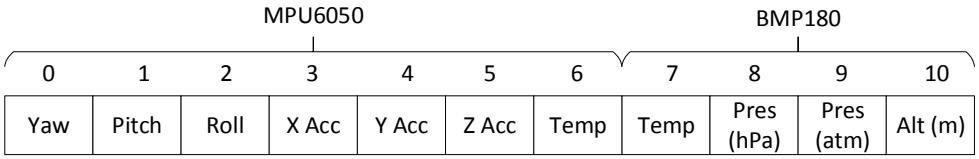


Figure 4.5: Structure of the sensor library data register

Before going into greater detail, we should clarify some concepts. The Bluetooth is a wireless technology that comprises a series of protocols that operate under a common protocol frame. These protocols build upon the basic Bluetooth standard are called profiles and they define what application is geared towards [25]. The BT profile used in this project is the Serial Port Profile (SPP). The SPP goal is to replace a serial communication interface, such as RS-232 or UART. To use SPP, the devices should be connected to the Serial RX and TX lines, and regular serial communication should be used.

Nevertheless, the data frame of serial communications can vary from device to device [23]. To avoid issues regarding configurations a data frame structure has been developed. This new protocol is a layer build upon the existing SPP. Depending on the direction of the message, into or from the Arduino, the length of the message will differ, as it will be explained briefly. Nevertheless, all the messages consist of a array of bytes that contain a start and end of frame (SoF and EoF, respectively), and a fixed structure. We will use little endian, which is the endianess used in Arduino and Android platforms. So, each variable will be stored from the least significant bit to the most significant bit.

Lastly, we have to remind that some variables type may change from platform to platform. In Arduino a character is 1 byte, an integer is 2 bytes and a float is 4 bytes. Whereas in Android a character is 2 bytes, and integers and floats are 4 bytes long. As the core of our project is the DAD we will use the variables format from Arduino, and therefore this issue has been addressed in the HMI, which transforms the data received and sent to the Arduino format.

Incoming messages

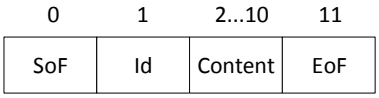
As explained before the data will be received in packages of fixed structure, as it can be seen in Fig. 4.6. This is very useful in order to dismiss corrupted messages. This incoming package is an array of 11 bytes.

Basically, once every operation cycle we will look if any data has been received during the loop. If so, it will save to a buffer the data between the SoF and EoF. If no SoF or EoF are found (in its proper place), the message will be rejected.

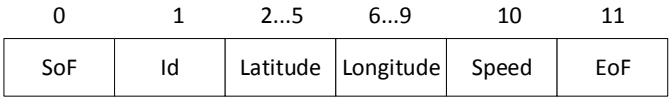
Then, the first byte of the buffer will be evaluated, which identifies the type of message and its content. The possible identifiers are the following (introduced in section 4.2):

0 **Empty message:** All the content bytes are empty. See Fig. 4.6a.

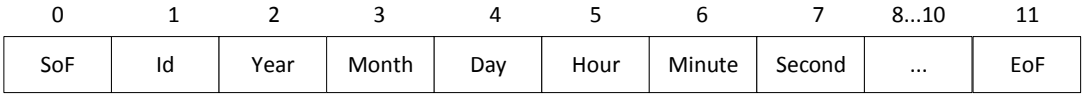
1 **Handshake:** To confirm successful BT pairing. The content of the message is empty (Fig. 4.6a).



(a) Generic message



(b) GPS message



(c) Clock message

Figure 4.6: Structure of the incoming BT messages

2 **IMU calibration:** Request to perform a calibration of the IMU. The content is empty (Fig. 4.6a).

3 **Adjust time:** Request the adjustment of the RTC registers. 6-byte timestamp is send as content and rest of the bytes are empty, as it can be seen in Fig. 4.6c. Notice that in one byte only 256 different values are possible, for this reason is not possible to send the year in a byte. Nevertheless, the content of this byte is the years passed from 1970 (following the UNIX time criteria [26]): we can send years from 1970 until 2225. Rest of values require less than 256 values.

4 **Reset Arduino:** A reset of the controller is being requested. No more data is required apart from the id (4.6a).

5 **GPS:** The data from the GPS receiver is received. This data consist on the latitude (as a float, so 4 bytes long), the longitude (as a float) and the speed (one byte). The structure of this type of message can be seen in Fig. 4.6b.

6 **Create a new session:** Request of create a new logging file, so no more data is required (4.6a).

As it can be seen, the efficiency of the transmission is very low because as maximum only 9 out of 12 bytes contain data, and in a lot of cases there is only 1 useful byte out of 12. Therefore, the efficiency oscillates between 75% and 8.3%.

The performance could be increased using a flexible frame length. Nevertheless, having a fixed structure is easier to implement and the transmission time is marginal for our requirements (at 115200 bps, to send 12 bytes we need 104µs). Moreover, most frames contain GPS data. So the efficiency is closer to 75% than 8.3%, but will vary depending on how many commands are sent from the HMI.

In addition, only the meaningful bytes will be processed because of the identifier, so we can say that the time lost in the transmission is compensated in the processing

speed. Moreover, this message structure is very robust and the transmission is very fast due to buffer small size.

Outgoing messages

The outgoing messages contain useful at-a-glance data for the user. The data which concerns to the user is the pitch, roll, acceleration in the direction of the movement (y-axis) and the temperature of the TS (from the barometric sensor). The structure of the message will be always the same, which is shown in Fig.4.7. To simplify message construction, the register from the library (see section 4.2.4) will be used, as it already contains all the data to be sent. As explained before, this register used to be from integer in previous versions, but now it contains float values. Thus, the outgoing buffer will contain float values. Hence, the frame is an array 19 bytes long, including SoF and EoF.

Unlike incoming frames, the outgoing messages will always contain the same data, so no identifier is required. However, in this case a control byte will be used. This variable was introduced before in section 4.2 when in the extended diagram (Fig. 4.2) appeared something called *Cmd*. This is a control variable that can provide useful information to the developer in case of debugging or even for control the status of the TS. The possible values of *cmd* are:

0 **Empty:** System working as expected.

1 **IMU calibration requested:** To let the user know when the calibration has begun in order not to interfere during the calibration process.

2 **IMU calibration finished:** Let the user know when is safe to ride the bike.

3 **Time adjustment requested:** Report time change synchronisation.

4 **Time adjustment finished:** Notify when the time has been updated.

5 **Reset requested:** Acquaintance about a reset request.

6 **Reset finished:** To report every time that DAD is turned on.

7 **New session created:** Notify a successful file creation.

For debugging purposes, the developer can introduce new commands, for instance to control the current state of the DAD, errors and so on. This case scenarios are not implemented because it was decided not to, but could be easily imported if desired.

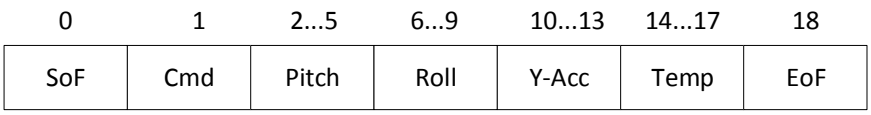


Figure 4.7: Structure of the outgoing BT messages

4.2.6 SD implementation

The SD is a very important extra feature of the project, allowing to record all the data. However, it is important to log this information in a comprehensive way, being easily understandable and classify it in following a stablished criteria. If we take these objectives in mind, the logging system obtained will enable performing quick searches (find the session from a certain date), easy analysis (the data will be saved always following the same layout) and exportation to other platforms and programs.

To achieve this goals, we decided to store the data in comma separated values files (.csv) which are easy to import to spreadsheet software. The values will be separated by semicolons (";") and the decimal separator mark is a dot ("."), despite according the International System is recommended to use a comma (","). Data will always be stored in the same column, so no identifier is required (the position is he identifier). In each operation cycle all data will be saved to the logging file, without prejudice to different sensor acquisition frequencies.

Create file

In order to have all the files well classified, all the files will be named according to the time of creation (as *"hhmmss.csv"*), in a folder path from the date ("*/Year/Month/Day/*". This system makes very handy to back up files in batch and look for specific logging files.

Essentially, the algorithm used firstly obtain the timestamp. Then it performs a search of the path, and if some of the folders is missing, creates it. Then it creates the file in the path, as it is not expected to have two files created during the same second, and if so the first one will be overwritten.

Update file

As explained before, the data will be separated according to type (in different columns, in between semicolons) and by time (in different lines, so in spreadsheets it will be seen as rows).

The structure will be always the same and it is shown in Fig. 4.8. These values have been obtained from the RTC module, from the sensor library register (see Fig.4.5), from the Android device (GPS information) and some internal values. Notice that there is a boolean record (1 if true, 0 if false) which point the BT pairing status. If no device is paired, then it will not save the GPS data, else it will store the last value obtained.

There is a developer register as well, which for ease of configuration it is set to be the same as the cmd value from the BT outgoing frames (see 4.2.5).

4.3 HMI

The HMI consists of a graphic interface that displays the data received from the TS, through a BT connection. The BT code is based on the Bluetooth example provided by Google [27].

When opening the application, the main screen will appear, which can be seen in Fig. 4.9a. At the top of the screen there is the application name and the status of the BT

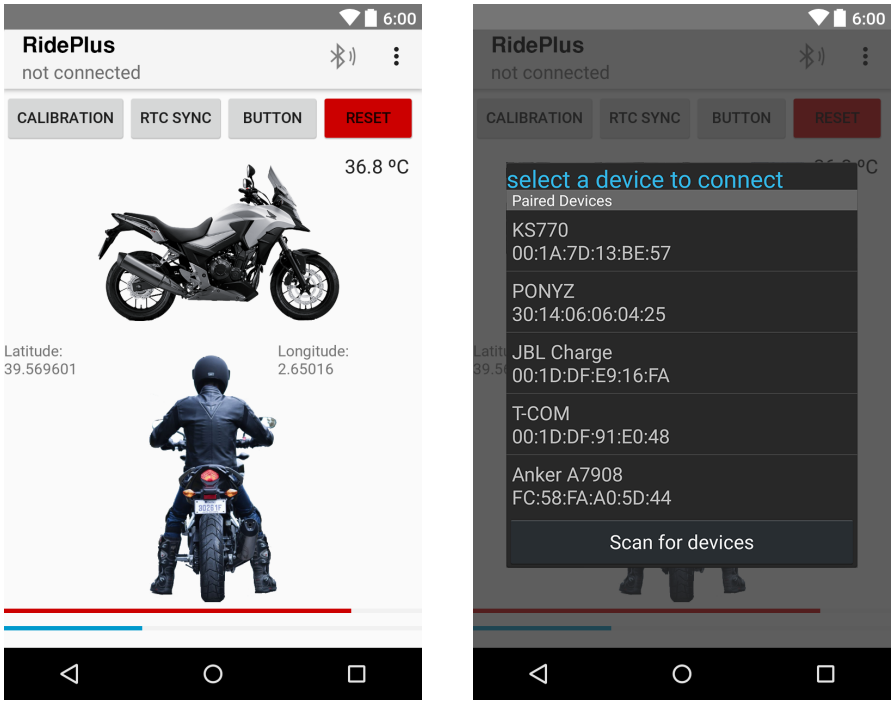
Timestamp:	year; month; day; hour; minute; second;
MPU6050:	rot_x; rot_y; rot_z; acc_x; acc_y; acc_z; temp;
BMP085:	temp; pressure; atm; altitude;
Android:	paired; speed_km/h; latitude, longitude;
Developer:	cmd;

Figure 4.8: Structure of the logging line in the SD

connection. To pair the TS, the BT icon at the top right should be pressed. Then the list of the available BT devices will appear, from which the TS will be selected, as it can be seen in Fig. 4.9b. If the connection is successful a pop message will appear and it will return to the main screen, where the connection status will be updated.

Below the BT status, we have the control buttons, that perform the tasks of: IMU calibration, RTC synchronisation, create a new logging session and reset the DAD. Below there are two motorcycle figures (a lateral and a back view) that rotate accordingly to the angle of pitch and roll, respectively. The centre of rotation is the rear wheel, as expected.

At the bottom of the screen we have two bars. The bar on the top shows the positives values of the Y-axis acceleration in red. The blue bar displays the module of the acceleration values when braking. The HMI also displays the TS temperature and the GPS coordinates.



(a) Main screen (b) List of available BT devices

Figure 4.9: Android application layout

4.4 Hardware arrangement

In this section, main schematics of the project will be shown and it will be explained the details. If needed, theoretical concepts will be presented.

4.4.1 Background

Before introducing the electronic schematics, we should address some topics related to the assembling of the DAD with the sensors.

On the one hand, the operating voltage of the BT module is 3.3V, instead of the 5V from the Arduino. Therefore the serial input and output from this devices should work at this voltage. Thus, we must transform the signal from the Arduino TX pin from 5V to 3.3V, and the RX pin the other way around. To do so different solutions are possible, from using a voltage regulator to using diodes. We think that the best solution is the use of 2 MOSFET transistors as level shifter (one for each line). The schematics of the configuration used to obtain a correct level shifting are shown in Fig. 4.10.

On the other hand, it is important to notice that the GY-87 breakout board (IMU, barometer and magnetometer) and RTC DS1207 are not connected to the same bus, despite both of them supporting I2C. This is due to an address overlap, that will be explained in section 4.5. Therefore, the GY-87 will be connected to the hardware I2C pins, while the RTC will be connected to a software simulated I2C bus.

Both circuits will be printed only on one layer in order to reduce the cost of the whole product. the schematics and the board files Eagle CAD [28] software has been

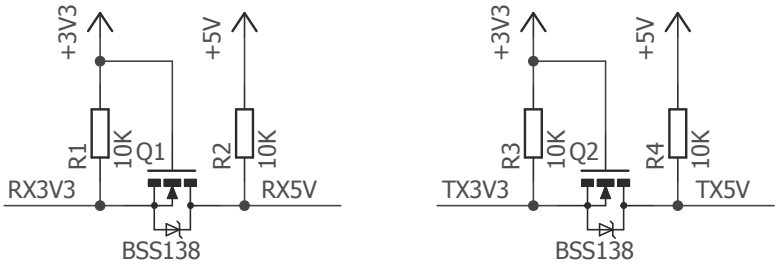


Figure 4.10: Level Shifter

used. For more information about the design process refer to the Aaron Eiche [29] and Open Electronics [30] guides.

To design the Mega enclosure the TinkerCAD [31] and SolidWorks [32] software have been used. The process will be explained later on in section 4.4.4

4.4.2 Arduino Nano Circuit

In the basic configuration, the Telematic System consists of an Arduino Nano (acting as the DAD), the GY-87, the BT module and its level shifter. The connections can be seen in the schematic from figure 4.11.

The Arduino Nano configuration was the first to be developed, and during the implementation process a prototype was used.Once the prototype was functional, a printed circuit was developed. After some versions and modifications, the final design is the one from Fig. 4.12.

4.4.3 Arduino Mega Shield

For the extended version, the controller is an Arduino Mega. Besides the GY-87 and the BT module, there is the datalogger which consist of the SD module and the RTC. The schematics can be seen in Fig. 4.14 and 4.15.

During the implementation of the data logger a prototype on a breadboard has been used. The Fig. 4.13 show the configuration used during the testing.

The result of the printed circuit is shown in Fig. 4.16 and in Fig. 4.17 with the sensors attached.

4.4.4 Arduino Mega Enclosure

A enclosure has been designed with SolidWorks from scratch to contain the Arduino Mega with the shield. The plans of the enclosure are attached in the annex of this document (see Appendix A).

Some aspects of the design should be highlighted. This enclosure is a box formed by two pieces. The bottom part is a flat surface with indents for the nuts. The Arduino will be attached to this part. On the other hand, the top piece is a 5 walls box. This part, has two holes in order to access to the power plug and the SD slot. Furthermore, it has reinforcements where the nuts will be placed.

Notice that the placement of the screw holes are given by the Arduino design, whose plans are accessible at the official webpage [20]

The rest of the design considerations are for aesthetic reasons.

The result of the print is shown in Fig. 4.18

4.5 Considerations

To end this chapter some major issues found during the implementation process will be outlined, and it will be explained how this problems have been addressed.

4.5.1 Libraries issues

At the beginning of the implementation we had to try different libraries for the MPU6050, because the available ones have some downsides such as few documentation and bugs. Finally, we decided to use the library developed by Jeff Rowberg, but some modifications had to be done in order to be suited for our project.

4.5.2 Android development platform

Regarding the HMI implementation, it was clear from the start of the project that it would be done with Android devices. Nevertheless, different programming environments have been used until achieve the final results. The first versions of the HMI were developed using the AppInventor platform from MIT, because it was very easy to use and no experience in Android development was required. However, when adding new features it was clear that it was not powerful enough for our requirements. Then we start programming using Processing. At the begging we developed a java application for Windows that was able to receive the IMU data and create a 3D model in real-time. Once it was finished, we tried to port it to Android. Despite being Processing adapted for Android development as well, we encountered a lot of issues during the migration because Google changed the Bluetooth API in 2012, and the existing processing libraries were not updated for Processing v.3 and Processing v.2 was not stable in Android. Therefore, we decided to use the Android Studio suite. Despite being a more complex and difficult platform, it was much more powerful than the other two and best suited for our needs.

4.5.3 Broken components

It is also important to comment, that during the implementation stage some sensors were damaged. For instance, while testing a bad connection on the breadboard burned the RTC module. Luckily, it was easy to replace as it is one of the most used RTC modules on the market.

4.5.4 I2C address collision

Nevertheless, the biggest issue encountered is the problem of the I2C addresses. As explained in previous sections, when the TS implementation was completed, the data logger development begun. Once all the example code were operational, we start to

combine the TS and the datalogger. Nevertheless, we faced a lot of unexpected errors: the SD did not store correctly the IMU values, and the time stamp obtained seemed to be corrupted, the time did not advanced as expected and so on. After a lot of debugging, we found that the problem was that both MPU6050 and DS1307 used the same I2C address: 0x68. As stated in the MPU6050 documentation, it is possible to change the address of the sensor to 0x69. Nevertheless, in our breakout board (GY-87) this is not possible because the pin that selects the address (if this pin is low then the address is 0x68, if it receives a high voltage the address is 0x69) is directly connected to the ground, and we had no means to short it to a high voltage, due to physical limitations. On the other hand, it is not possible to modify the DS1307 address because it is not supported by the manufacturer.

Some solution had to be found in order to obtain a timestamp for the logging files. Firstly we thought that it would be possible to control the module in use (or the MPU6050 or the DS1307) by powering them independently. A demo sketch was develop which at the begging of the program the MPU6050 was turned off and the RTC on. Then a timestamp was obtained and the inner Arduino clock begun. After that, the RTC module was disabled and the MPU6050 was enabled. Then it would start to record the IMU data with the correct timestamp (the combination of the first timestamp and the Arduino clock). Nevertheless, this sketch was a complete failure. After some research, we found the root of the problem: according to the I2C characteristics, if a module connected to the bus is not turned on, then the whole Serial Data Line is shorted to ground. Therefore, all the modules must be turned on in order to use the I2C bus.

4.5.5 Flash storage usage

Then another solution was found: using a multiplexer on the I2C bus. This configuration allows us to have as many virtual I2C busses as we want to. But it was too late to buy the multiplexers, as we needed immediately to resume the project development.

The best solution found was to simulate by software another I2C bus. There are a couple of different libraries that accomplish this goal. The most used library to simulate a I2C bus is not compatible with the standard SD library: to use this library I2C emulated pins must be defined as macros, but they are already defined in the configuration file of the SD library. Finally, we were able to find a useful library for our case.

After solving this issue with the I2C addresses, we resumed the regular development of the extended version. At that moment, the extended version had an Arduino Nano as the core of the system. This had to be changed, because the use of the software I2C library and the new features occupied more flash storage than the available in the Arduino Nano (32kB). After some days trying to optimise the code, we notice that it was not possible to use less than the 30kB available (as 2kB are already used by the bootloader). At that moment we decided to change the Arduino Nano for the Arduino Mega, which has 4 times more Flash storage. This boost of storage is very positive for our interests, because further development can be done as more features can be added.

All the schematics from the previous extended configuration (using the Arduino Nano) are attached in the Annex.

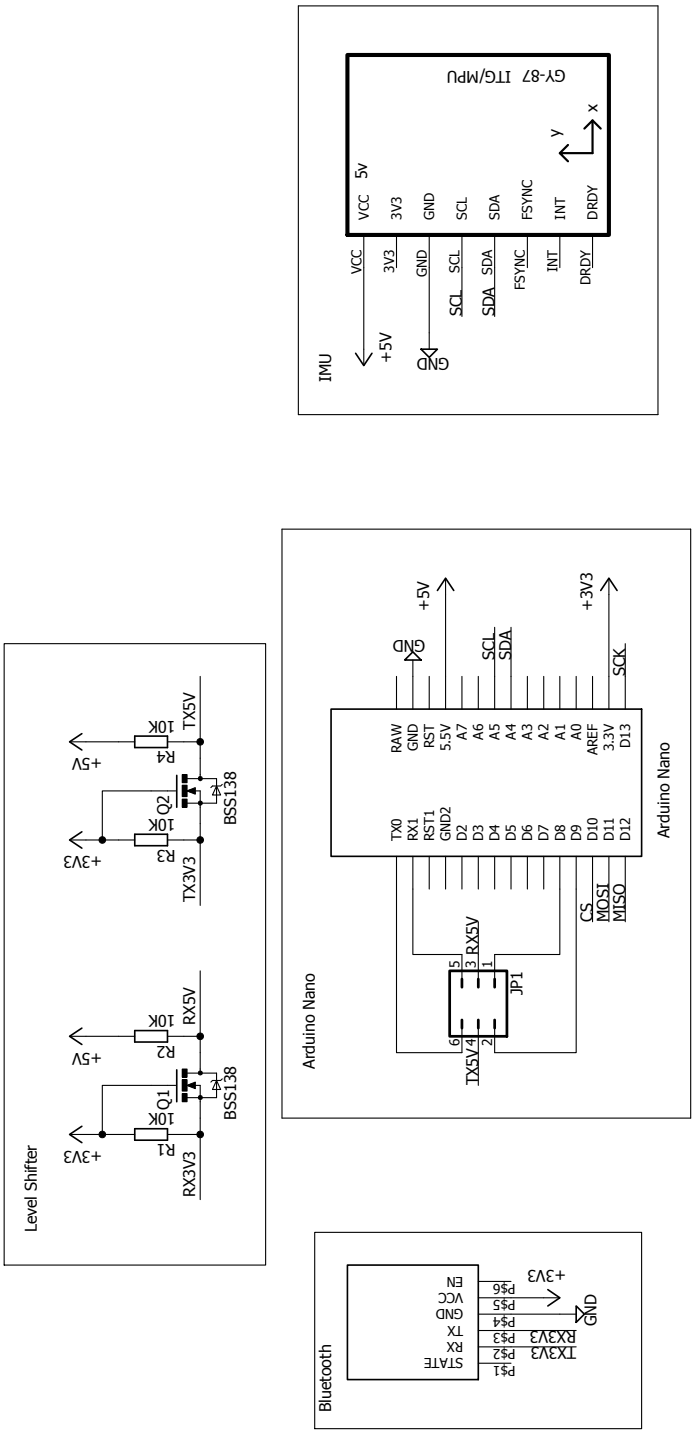


Figure 4.11: Schematics: Arduino Nano platform

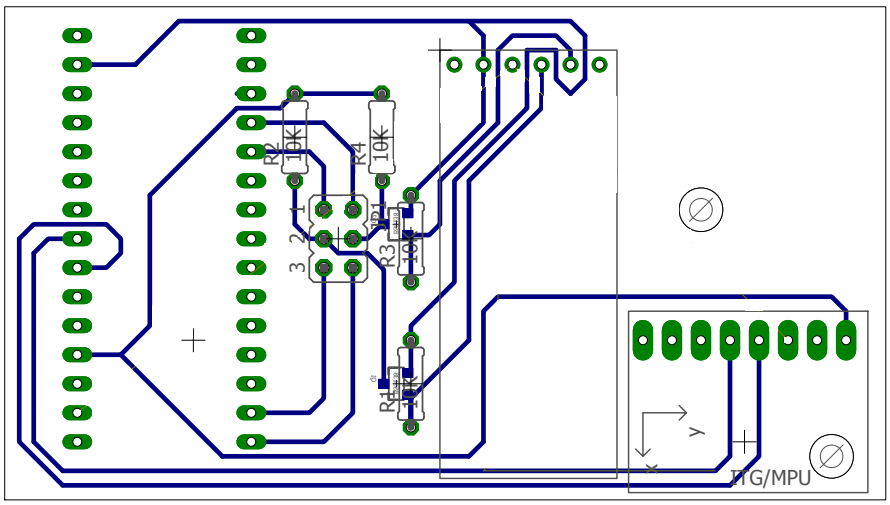


Figure 4.12: Board overview: Arduino Nano platform

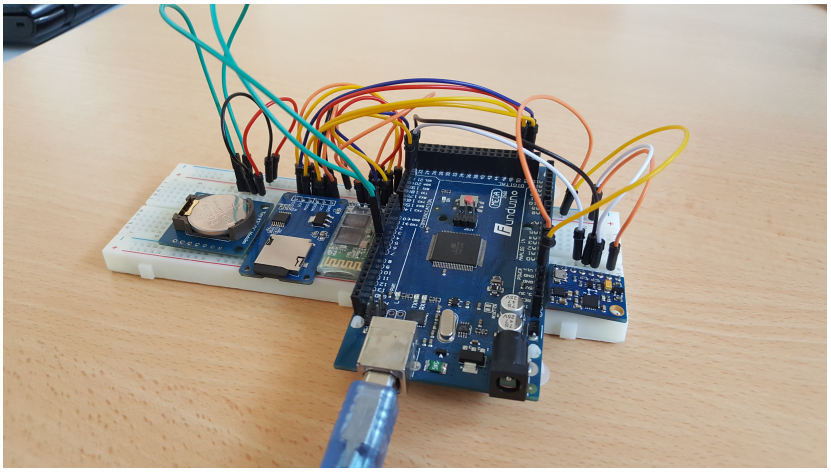


Figure 4.13: Mega prototype

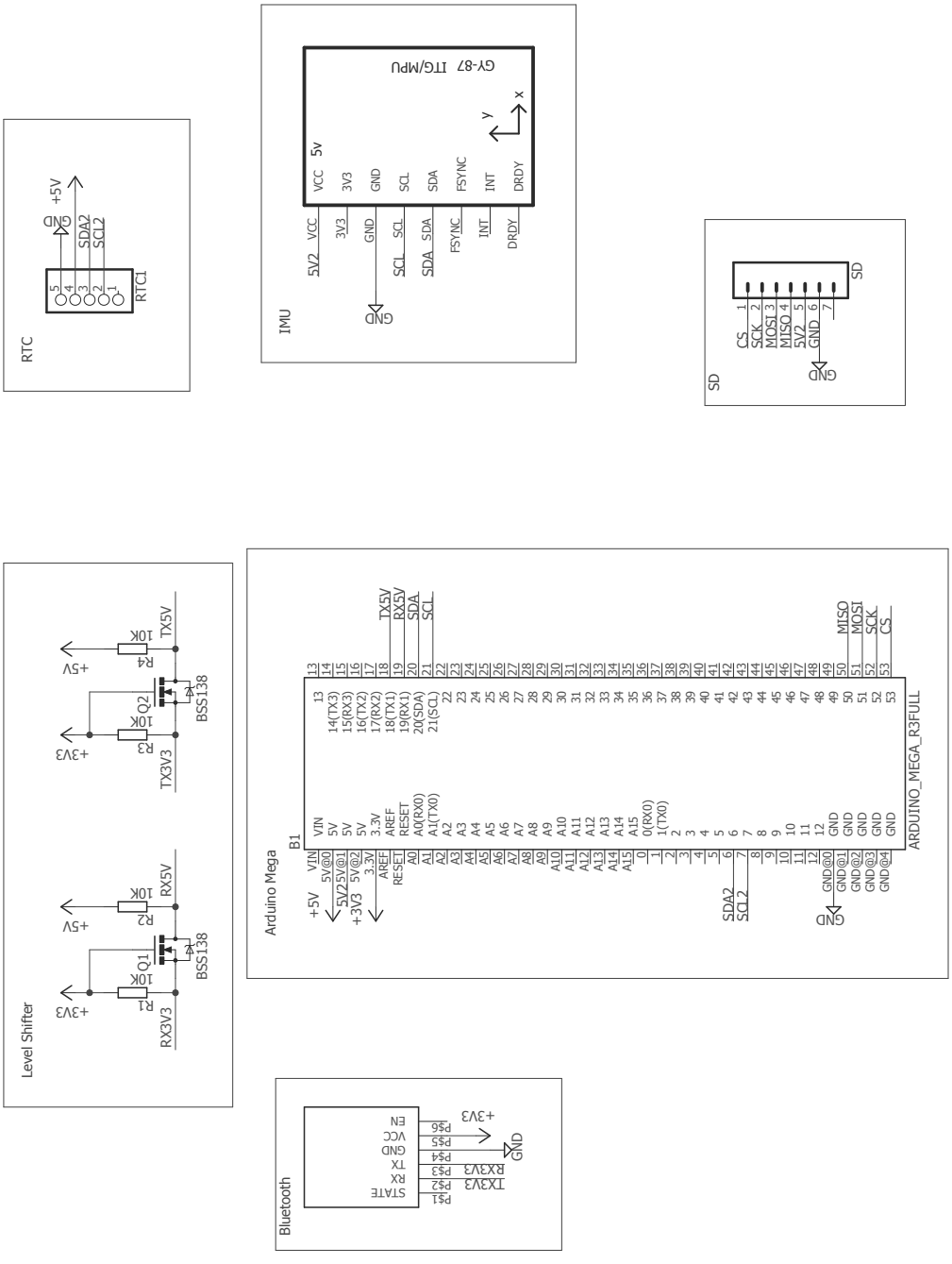


Figure 4.14: Schematics: Arduino Mega platform

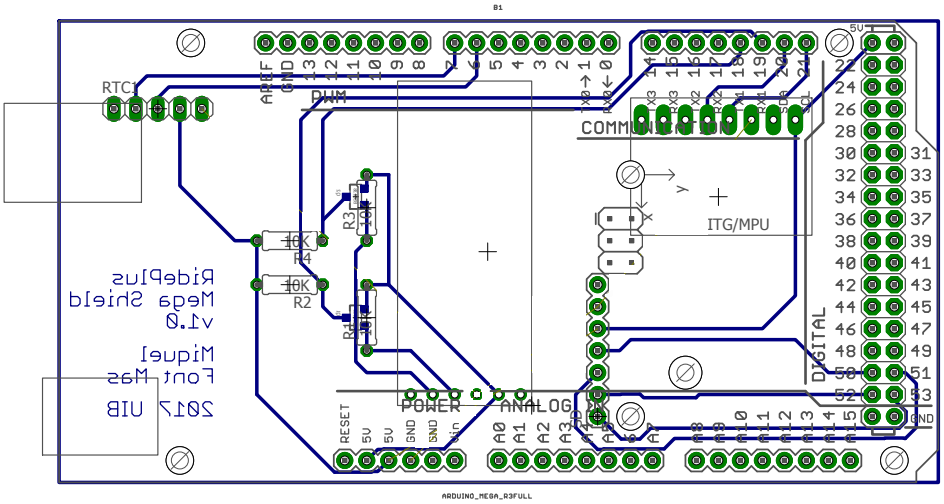


Figure 4.15: Board overview: Arduino Mega platform

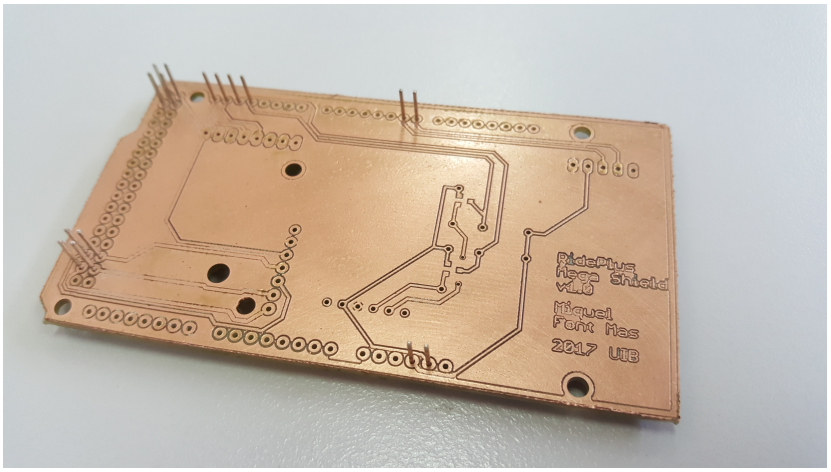


Figure 4.16: Printed circuit: Arduino Mega platform

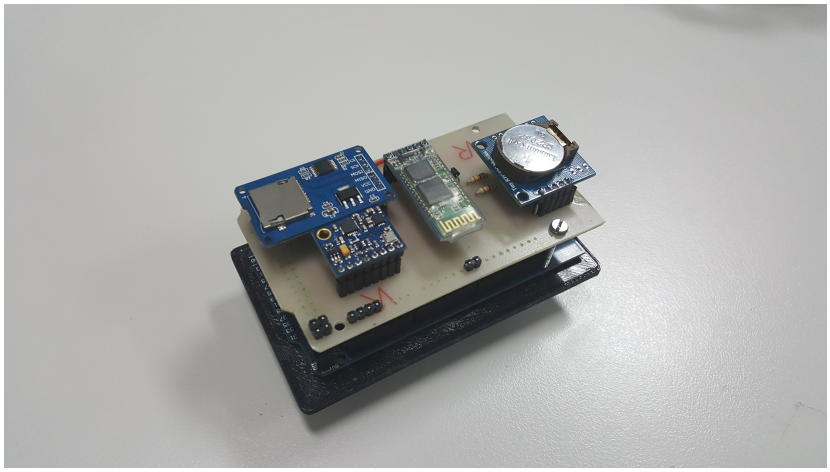


Figure 4.17: Final result: Arduino Mega platform

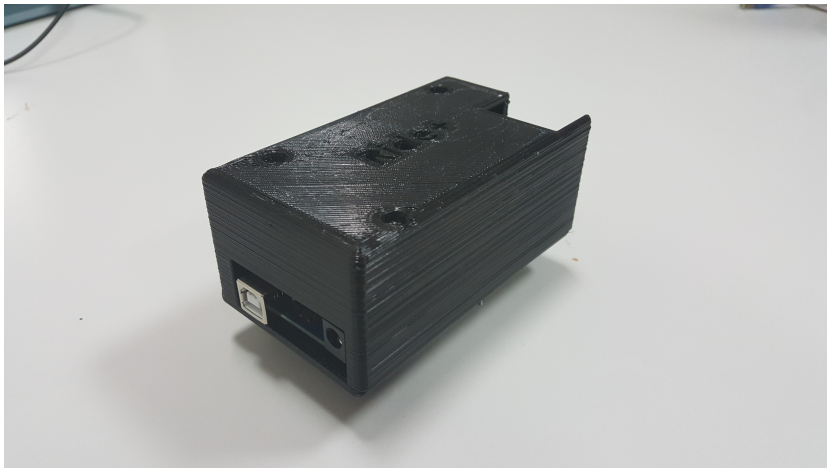


Figure 4.18: 3D print result

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter we are going to analyse the data obtained by the TS and stored in the SD. First we are going to perform a series of tests under controlled conditions in order to ensure the proper operation of the device. Later on, we will perform field trials and the data resulting from this experiments will be studied.

It is important to notice that the acceleration in the Z-axis has the gravity component removed.

5.1 Laboratory tests

The following tests are performed under controlled conditions in the laboratory. The movements have been performed by manual means and vibration from hand shaking can be expected.

5.1.1 Calibration

Firstly, TS has been placed in a flat surface and the IMU offsets have been set to zero to ensure that the sensor it is not calibrated. Then we values acquired at rest will be measured. After some time, a calibration will be performed and the same process will be repeated.

The values obtained before calibrating the sensor are shown in Fig.5.1. As it can be seen, the sensor is in a transient state at the beginning, and finally it ends in a steady state. The values obtained do not match with the position of the sensor, as it should have zero roll and pitch, and the roll of 69º and pitch of 16º are measured.

Then a calibration is performed, requiring a total time of 68 seconds. Next, offsets obtained from calibration are set and the test is repeated, obtaining the results from Fig. 5.2, where both roll and pitch are zero. After calibration, the transient state has been disappeared. This test has been repeated, and the results have always been consistent.

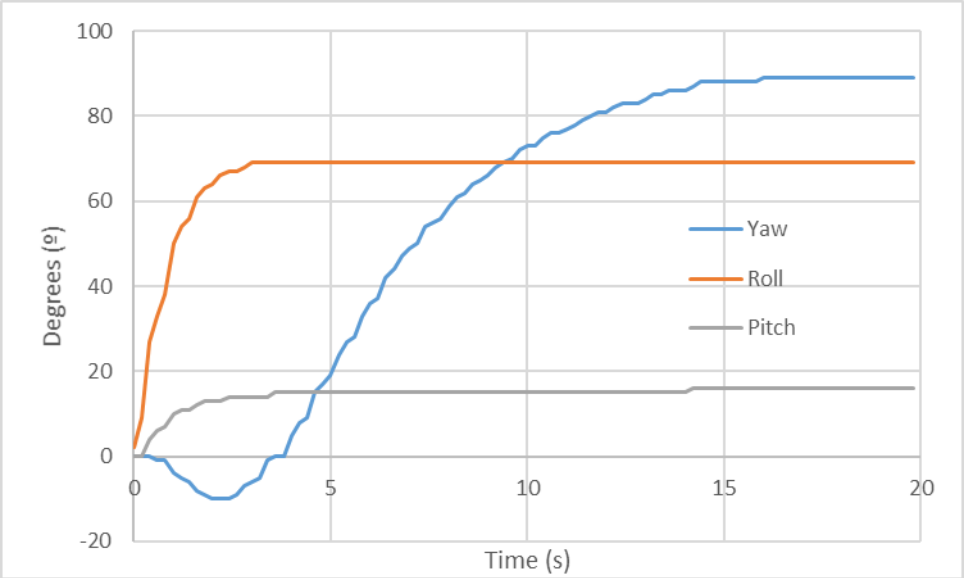


Figure 5.1: Calibration test: YPR before calibration

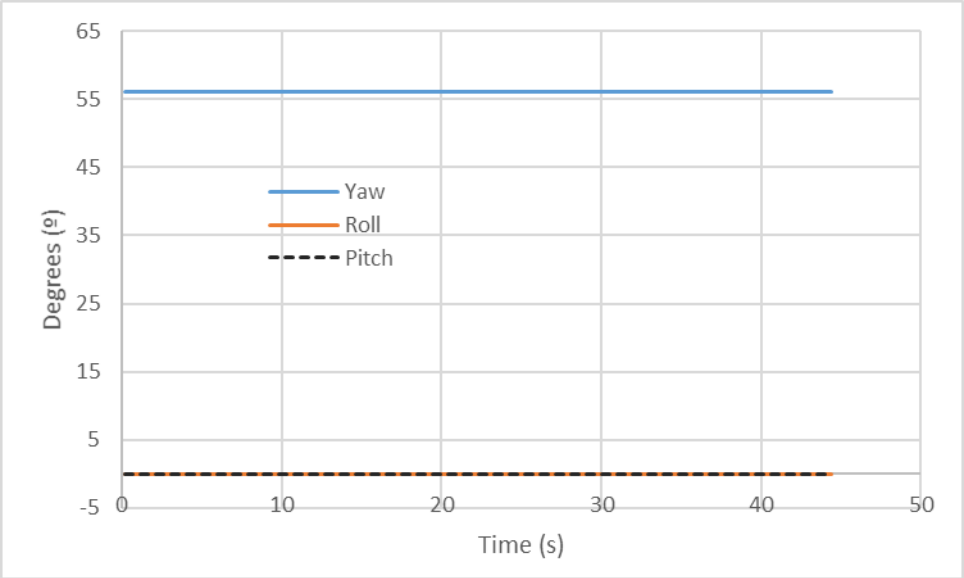


Figure 5.2: Calibration test: YPR after calibration

5.1.2 Data reliability

Now we have to ensure that the data is reliable when the system is dynamic. To do so, a number of tests will be performed, ensuring that values of angular position and lineal acceleration are good.

Acceleration

We are going to start from rest. Then, a gentle shaking in the X, Y and Z axis is going to be performed. The results can be seen in Fig. 5.3.

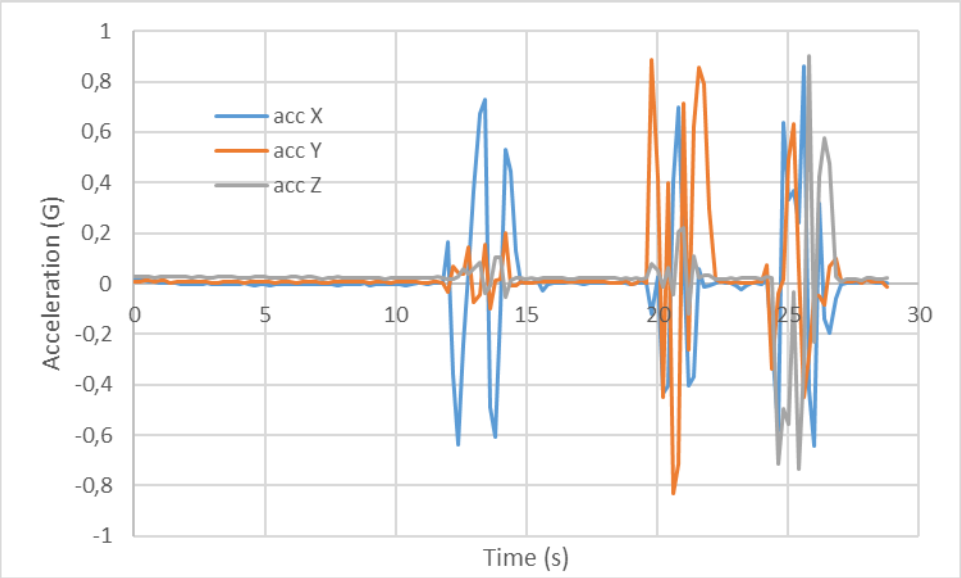


Figure 5.3: Reliability test: X, Y and Z accelerations

We can see that the results are the expected: after a acceleration it comes a deceleration of the same magnitude.

We must comment that the Z-axis movement was done by hand with no support, while the other two a table was using as a reference. Therefore, during the Z-axis movement we can see vibrations in the other axis as well.

Roll

The goal of this experiment is to check the proper acquisition of the roll values. From rest we are going to perform a 90 degree roll to the left of the sensor (negative roll direction) and return to the initial position. Then we repeat the same process but leaning to the right (positive roll). The results can be seen in Fig. 5.4.

We observe that the values meet with the dynamic behaviour performed. Nevertheless we must notice a implementation detail. In the peak of 90º we get 3 unexpected changes of direction of the roll (ergo two peaks). Nevertheless, as we only change the direction once, only one peak is expected instead of two.

This data behaviour is due to a implementation consideration: the sensor can not differentiate two symmetrical inclinations. Therefore, it understands 80º and 100º as the same value: 80º.

Then, if we take this consideration into account, we realise that we actually performed a roll of 100º, instead of 90º.

Pitch

We repeat the same process as last section but in this case we are going to perform a pitch rotation, first pointing to the sky and then to the ground. The results are available in Fig. 5.5

In this case we can see very clearly how the sensor is moving as expected.

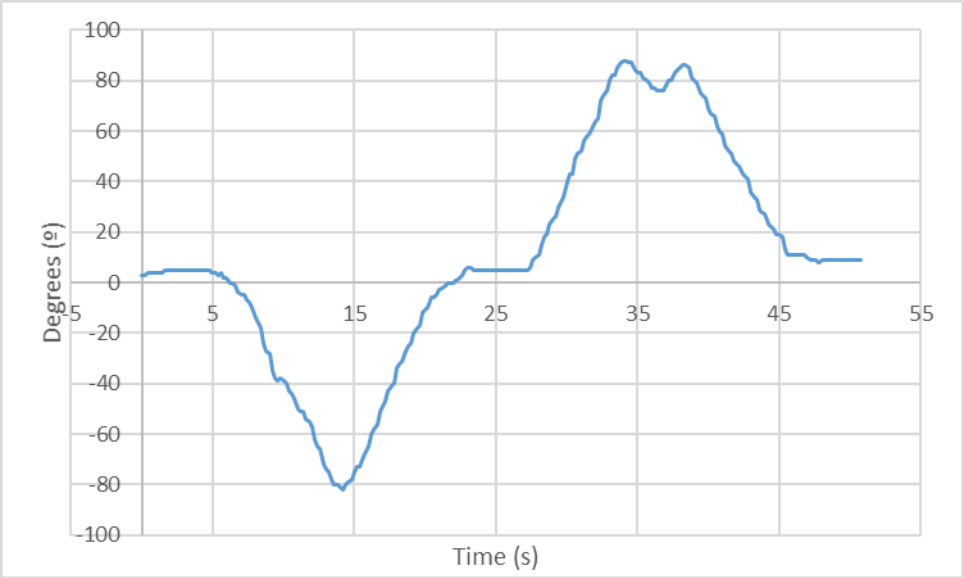


Figure 5.4: Reliability test: Roll

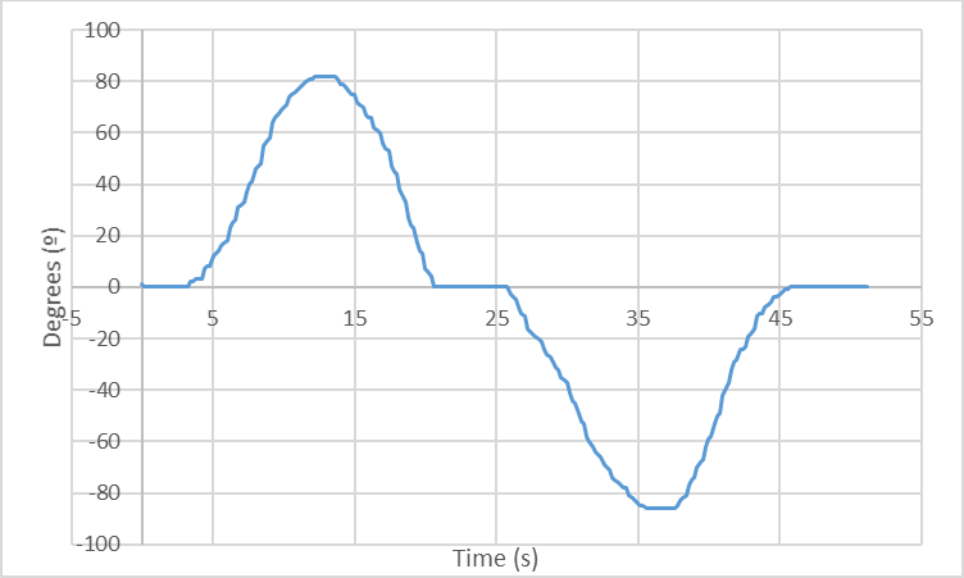


Figure 5.5: Reliability test: Pitch

5.1.3 Value drifting

In this test we want to analyse if the results drift after experiencing a high amount of force in different directions.

To do so, we will place the sensor in a flat surface. After some time in steady state, the sensor will be subjected to a rough shaking, which will originated by moving randomly the sensor with rush. Next, we will place the sensor in the same position as before.

In Fig. 5.6 we can see the angular position of the sensor, while in Fig. 5.7 we can see the accelerations.

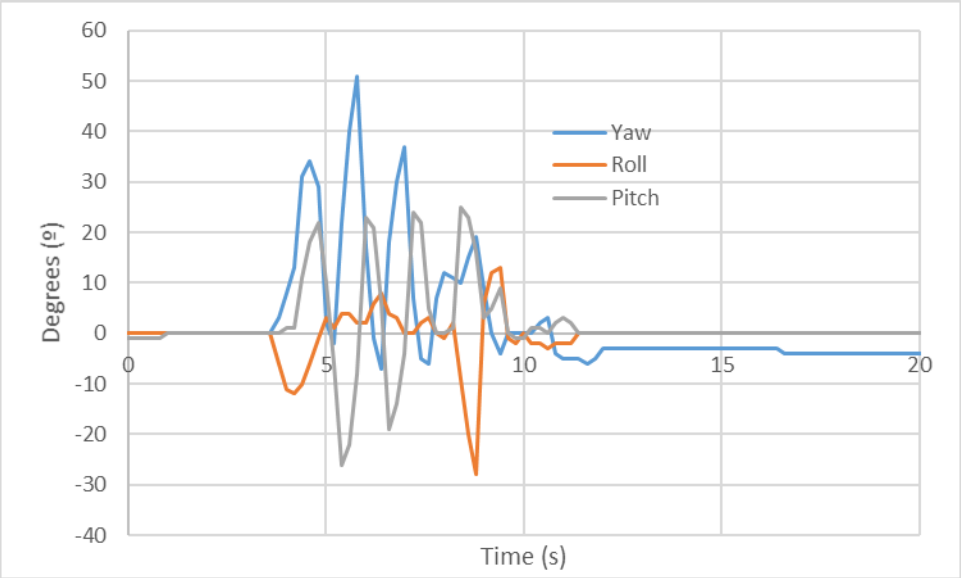


Figure 5.6: Reliability test: YPR evolution during turbulence

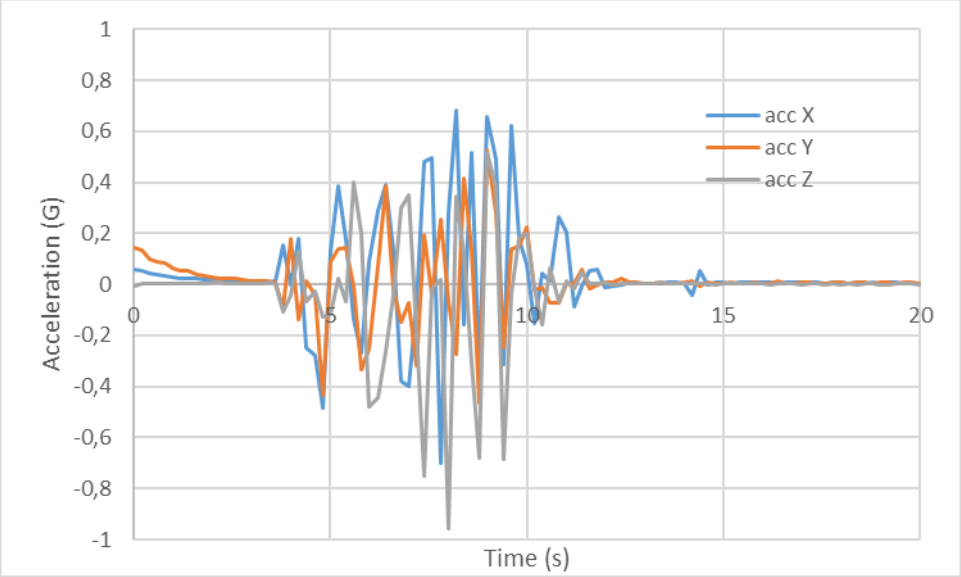


Figure 5.7: Reliability test: Acceleration evolution during turbulence

Once sensor is in stationary state after being shaked, we observe that the roll and pitch return to their original positions. Therefore there is no drift in these two directions. Nevertheless, there is a 5º drift at the yaw. Despite not being desired, this is very common in affordable IMU, as the gyro experiences a high drift. This effect can be removed using the magnetometer.

5.1.4 Conclusions

After performing these tests, proper proper operation of the device in a controlled environment has been checked.

5.2 Field trials

In this section we are going to analyse the data obtained when the TS is already installed on the vehicle. To perform the following tests, the IMU has been placed in a flat surface of the vehicle and with the Y-axis pointing to the movement direction. The setup can be seen in Fig. 5.8

Then, a new calibration has been done in order to obtain accurate results.

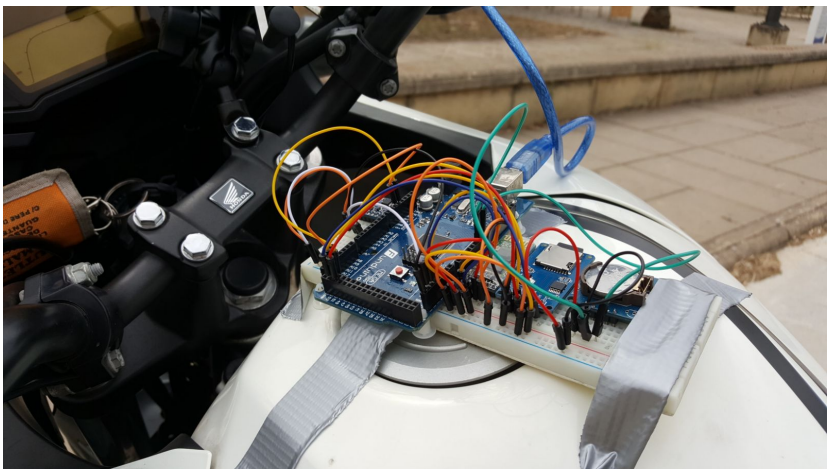


Figure 5.8: Testing Mega unit

At the beginning of all the tests, the vehicle is going to rest in a steady position as it is shown in Fig. 5.9. So the roll and pitch angle should be 0.

Figure 5.9: Initial motorcycle position

5.2.1 Lineal movement

In this series of tests we are going to study the behaviour of the motorcycle when it is moving in a straight line. All these tests consist in performing a acceleration until a certain speed is reached, and braking until the motorcycle stops.

Low speed test

The first experiment has been be done at low speed. The maximum speed reached is 40 km/h. In this test our interest is mainly focused in the YPR, but also on the acceleration.

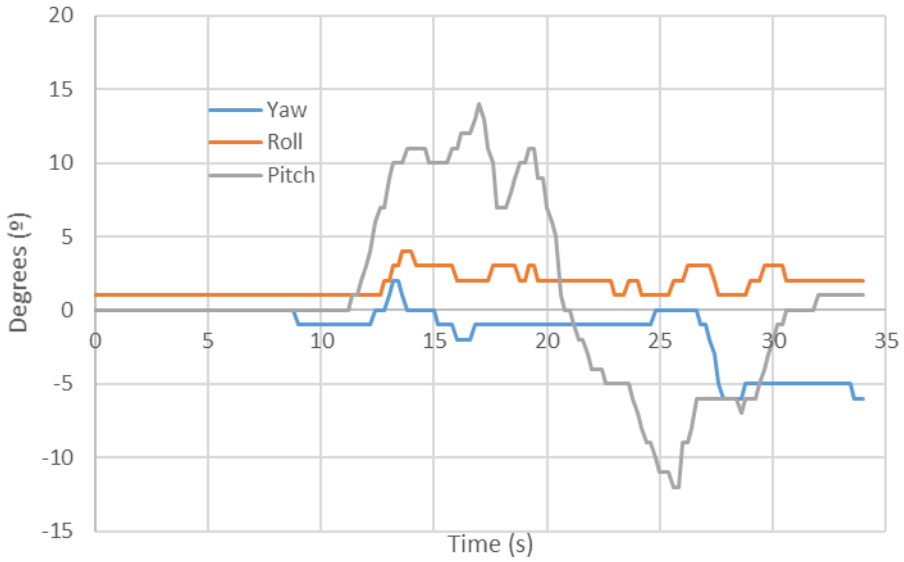


Figure 5.10: Straight line at low speed: YPR

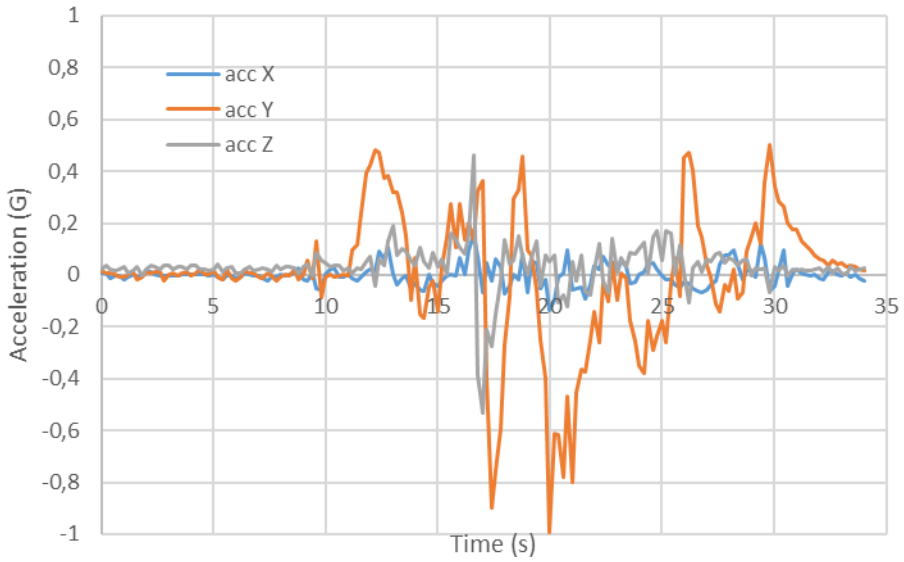


Figure 5.11: Straight line at low speed: lineal accelerations

We can see that we start accelerating at the 12th second. At this moment a positive acceleration in Y axis is experienced as we are increasing the speed of the motorcycle. We can observe that the pitch angle also increases from 0º to 10º. This behaviour is explained by the fact that a force is applied to the wheels in the direction of the y-axis. The force vector is not aligned with the centre of masses (CoM) of the vehicle, and therefore the resulting momentum pushes the vehicle up, expanding the front suspension.

At second 17, we change to second gear, experiencing a negative acceleration as the vehicle reduces the speed in a short time. In the Z-axis a negative acceleration peak is experienced as well. As the motorcycle reduce its speed, the front suspension absorbs

the forces and the pitch angle is reduced.

We accelerate until the second 20, when we brake. When braking, the same behaviour as in acceleration is experienced: a force acts on the contact surface of the wheel, pushing the vehicle backwards. As this force is not aligned with the centre of masses of the motorcycle, we obtain a resulting momentum, which in this case tends to decrease the pitch angle.

Furthermore, when braking, motorcycles tend to use much more the front brake than the rear one, contributing to the reduction of the pitch angle.

Once the forces do not act anymore, the suspension recovers its original position.

Nevertheless, we obtain a unexpected result when analysing the Y-axis acceleration. Despite reducing the speed of the vehicle until full stop, we see in the graph a positive acceleration instead of a negative one. This result will be clarified later on in section 5.2.1.

If we now analyse the yaw and roll values we can say that they are the expected, because of its closeness to zero. The variations around this value are because of the road conditions, wind and other physical factors.

We performed the same test four times and the results were always similar.

High speed test

A similar test as before is going to be performed but at a higher speed. In this case the bike will accelerate from 0 to 120 km/h, and once the top speed is achieved, the vehicle will slow down until its complete stop.

Notice that the road where this test was done is not a perfect straight line, as it has a small curvature to the left. This condition can bee seen in the yaw evolution from Fig. 5.12. In the same figure, we can also see that the roll values are very close to 0º.

On the other hand we obtained the same yaw evolution as in the low speed test: first the motorcycle has a positive pitch, which gets its maximum at 22º (almost 10º more than previously), and when braking the pitch becomes negative, with a minimum of -16º (4º more than before).

Looking at Fig. 5.13, the peak acceleration in the y-axis is at the beginning, experiencing an acceleration of 0.91G. The minimum is -0.65G, value obtained when shifting gear and at the start of the braking. In this experiment, we obtain the same unexpected acceleration values when braking as in previous tests. To ensure the reliability of the data, this experiment has been repeated three times and the results are analogous.

Now we will focus the analysis on the relation between the pitch and the acceleration when going from 0 to 120km/h. In figure 5.14 this relation can be appreciated.

The first thing to notice is the delay between the acceleration and the pitch. This is because the pitch is a result of the acceleration.

It is noticeable when gear shifting has been performed. The vehicle starts in first gear, which has the most acceleration as the sprocket is the smallest.

At second 6, a gear shifting is performed: when clutching, the vehicles decelerates drastically. Next the new gear is engaged to the engine, and we experience a new acceleration. The result of this process is a peak followed by a valley. We can observe that the gear shifting is performed at seconds 6, 7.5, 10, 12 and 13.5: from first to sixth gear.

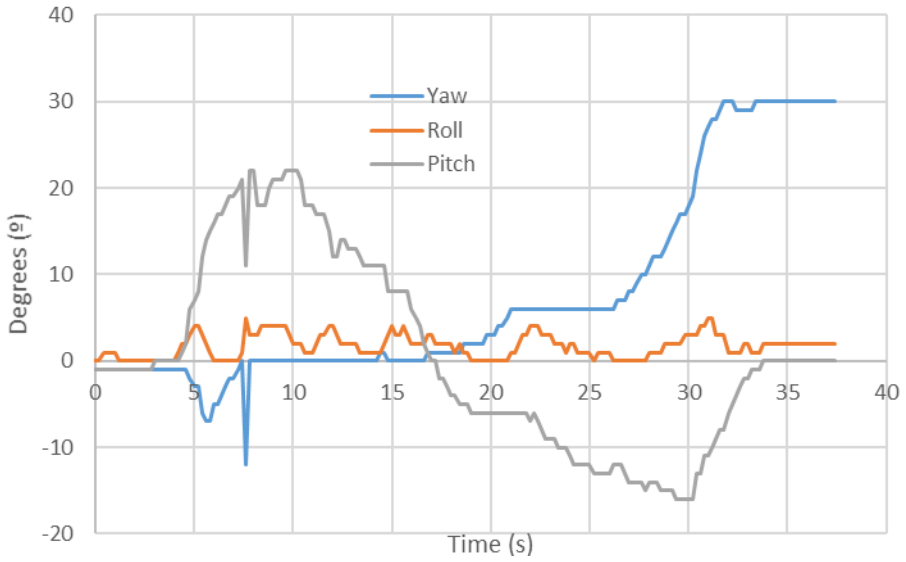


Figure 5.12: Straight line at high speed: YPR

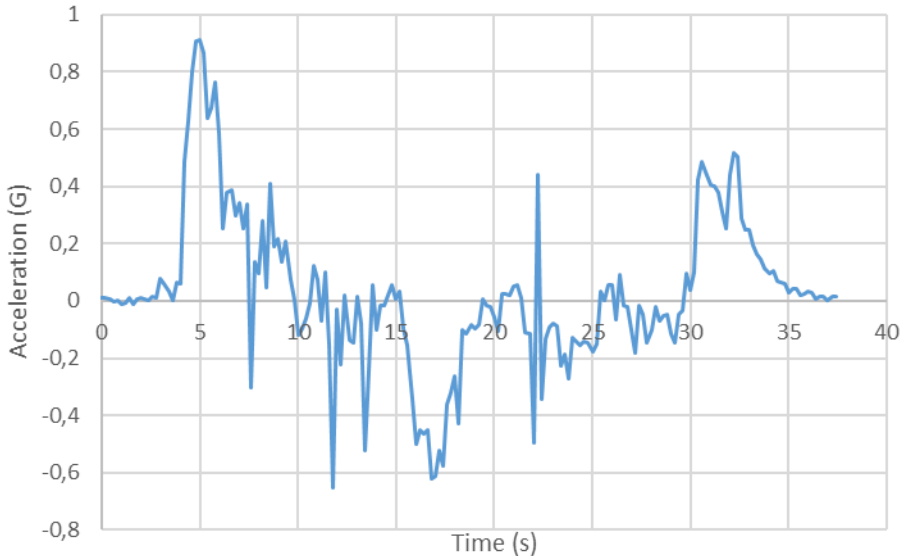


Figure 5.13: Straight line at high speed: Y-axis acceleration

Moreover, the figure shows that as higher the gear, the lower the acceleration. Which is due to the size of the sprockets: torque decreases with size.

As it can be seen in the figure, the suspension expands until the second 10, and then it starts to compress despite the pitch is still positive.

Acceleration analysis

As stated before, some Y-axis acceleration values are unexpected. For instance, when decreasing speed a negative acceleration should be seen. Nevertheless, just before complete stop, acceleration is positive. This behaviour can be seen in Fig. 5.13 and in

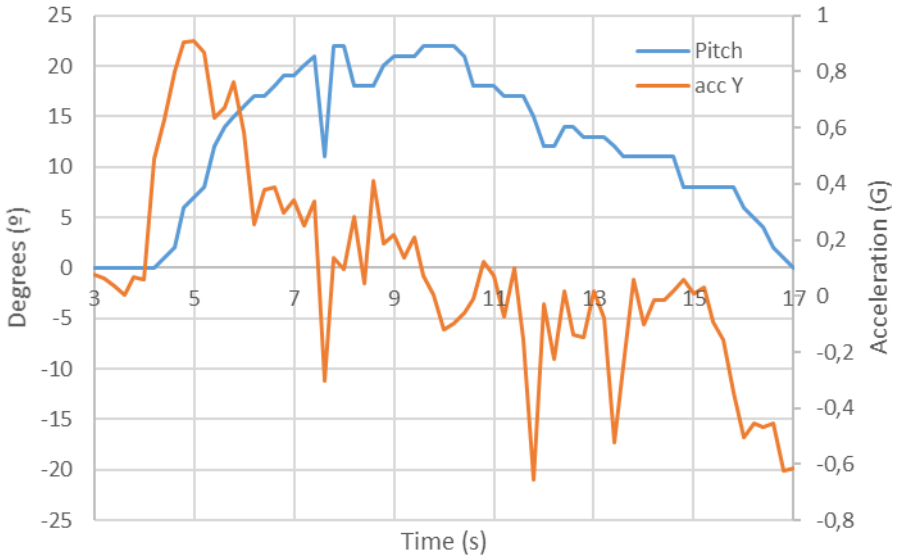


Figure 5.14: Straight line at high speed: pitch and Y acceleration from 0 to 120km/h

Fig. 5.11.

In Fig. 5.14, the acceleration is negative despite the speed and pitch are increasing. Positive acceleration is expected, while its magnitude decreases with time.

All the tests have been performed several times and the results are always the same when testing on a motorcycle.

Therefore, the dynamics of other types of vehicles have also been tested. These tests have been performed attaching the TS to a car. Nevertheless, the results regarding acceleration match with the obtained on the motorcycle. Thus, these issues are not caused mainly by the behaviour of the vehicle, despite it can contributes a little.

A possible root of the problem resides in the way data is sampled. On the one hand, we sample accelerometer data at a rate of 160Hz (5 averaged values, containing 32 samples each). Due to the possible loose of data when acquiring the data, a higher sampling rate could be used to obtain more reliable results. On the other hand, the way data is averaged acts as a low pass filter.

This data sampling indeed reduces the effect of non-homogeneous values. But it should affect the same way to positive and negatives values. To see so, when drawing the trend line of the acceleration from all the tests performed, the value is very close to 0. Which is expected, as we start and finish at rest.

Another cause can be noise due to physical factors, such as air force, vibrations and inertial force. This behaviour should be studied in detail in further research.

5.2.2 Curved movement

In this section the motorcycle response when leaning will be studied. Two different types of test will be performed, depending on the sense of rotation. These experiments consists on performing a 540º rotation (one turn and a half) in a 3 meters radius long circumference. The rotation speed should be as constant as possible.

Clockwise rotation

To perform this experiment the motorcycle will lean to the right. The results obtained are shown in Fig. 5.15.

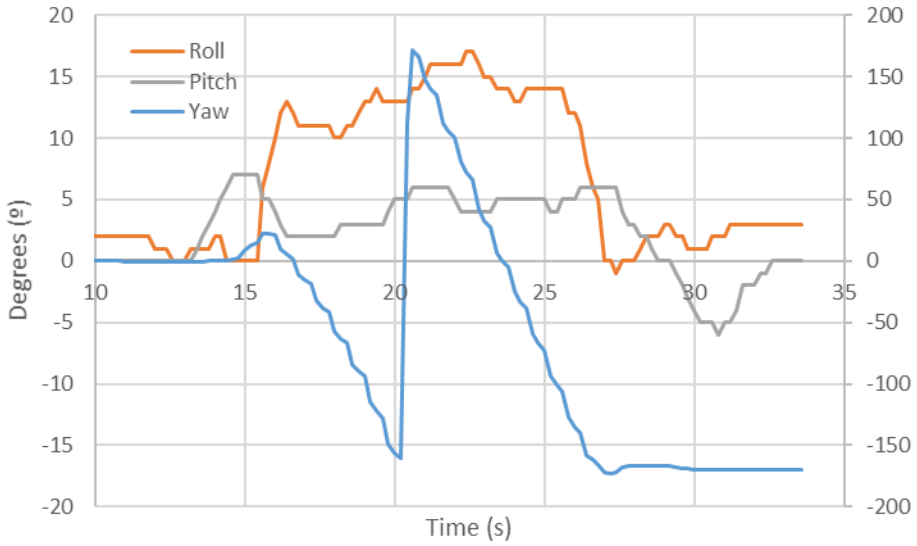


Figure 5.15: Evolution of YPR in a clockwise rotation. For better understanding of the information, the yaw values correspond to the secondary axis.

Looking to the yaw data, it is clear that a 540º rotation has been done, ending the the movement at 180º from the initial orientation.

Studying the roll data, we performed a positive lean (rotation to the right of the Y-axis), of an approximate mean value of 15º during the whole rotation. It should be underlined the flatness of the curve during the rotation, which is affected by the rider's technique. Notice that the leaning of the bike from 0º to 12º and vice versa are performed in less than a second.

It is important to notice that, during the rotation, pitch is positive, as the vehicle counteracts the effects of the centripetal force, through a constant acceleration in the Y-axis. When going out of the curve, the pitch decreases constantly as the speed is acceleration is decreasing. At second 30 we see how pitch returns to the initial value, as the suspension retrieves.

Anticlockwise rotation

The results obtained in this test are the opposed as the previous one, according to the Fig. 5.16.

Nevertheless, when leaning it is observed the following: a turn into the opposite direction of in order to enter cornering in a tangent line to the curve radius. It can be seen in the yaw and roll values from second 10 (when entering to the curve) and second 23 (when exiting).

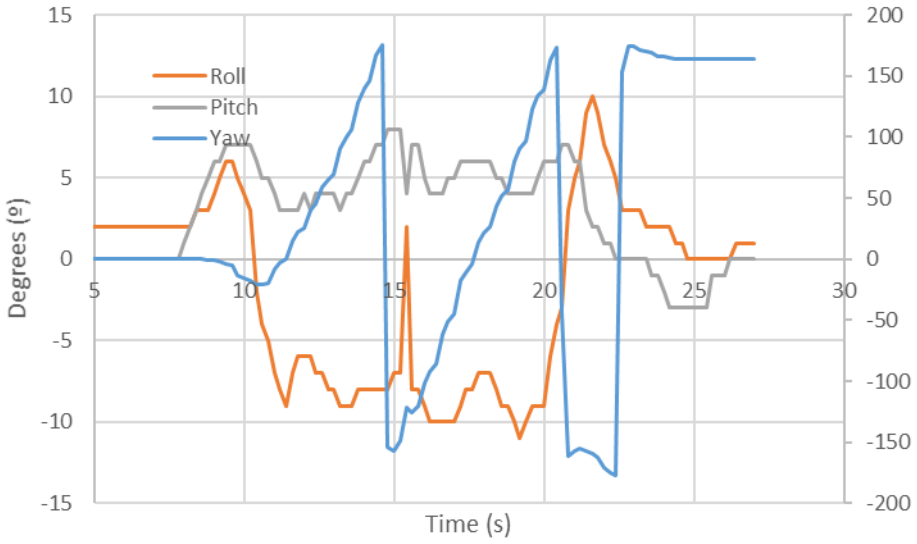


Figure 5.16: Evolution of YPR in an anticlockwise rotation. For better understanding of the information, the yaw values correspond to the secondary axis.

CHAPTER 6

FINANCIAL ANALYSIS

In this chapter, a study of the cost of the material involved in the prototype production will be done. In this analysis, research and work costs are not taken into account. Nor the unit cost during mass production, as factors of scale have to be studied.

Firstly the cost of each individual component at European retailers will be studied. Notice, that some boards are not available, thus individual components should be bought separately.

The subtotals depend on the configuration of the prototype, and the price is calculated by all the products required. Note, that if a product is sold in batches, we have to buy the whole batch, even if only one unit is used.

Product	RS Reference	Price per unit (€)	Units	Total Price (€)
Arduino Mega*	715-4084	43.51	1	43.51
Arduino Nano	696-1667	18.36	1	18.36
MPU-6050	883-7948	9.31	1	9.31
BMP-X80	849-6187	3.55 (x2)	1	3.55
HC-06	N/A	N/A	N/A	N/A
8GB SD + Adaptor*	695-7334	9.27	1	9.87
DS1307*	540-2726	2.92	1	2.92
BSS138	671-0324	1.35 (x10)	2	1.35
Basic configuration				32.57
Extended config.*				70.51

In comparison, here are the prices from Chinese retailers. These retailers offer a bigger range of products at a lower cost, at the expense of a lower build quality.

Product	Reference (Aliexpress ID)	Price per unit (€)	Units	Total Price (€)
Arduino Mega Clone*	1757110-32314375219	6.10	1	6.10
Arduino Nano Clone	1948124-32574307428	1.84	1	1.84
GY-87	615778-32612550940	6.50	1	6.50
HC-06	1948124-32501958088	2.47	1	2.47
SD reader*	343255-32478403262	0.42	1	0.42
8GB SD*	2961147-32804711976	6.02	1	6.02
RTC DS1307*	1950989-32530897478	1.06	1	1.06
BSS138	1087309-32422423954	0.76 (x50)	2	0.76
Basic configuration				10.91
Extended config.*				23.33

The products are 3 times more expensive at European retailers than at Chinese.

Other material costs must be considered. This is the case of products developed at the university facilities, which are funded to the student. Nevertheless, if these products had to be ordered to external companies, the cost would increase highly. This is situation is the same in case of the software used, as we can see in the following table:

Product	Source	Price (€)
1-layer board	UIB	Free
3D print	UIB	Free
M3 screw (x6)	UIB	Free
M3 nut (x6)	UIB	Free
Testing facilities	UIB	Free
Microsoft Office	Educative license	Free
ShareLatex	Free version	Free
Android Studio		Free
PlatformIO		Free
EagleCAD	Educative license	Free

CHAPTER 7

CONCLUSIONS

The project was proposed, in this case by the student, to address a clearly unattended market niche. With this work a solution to this situation has been proposed.

The research presented covers not only a market review, but also the component selection of both the TS and the HMI in a simple, affordable and effective way. The solution shown can be installed in any kind of vehicle, and its clearly modular and expandable, fulfilling the objectives listed in Chapter 1.

The states of this project have been:

1. Research of platforms and sensors: As seen in chapters 2 and 3, a deep insight into developments on the automotive industry and the technologies used has been provided.
2. Design of the device: This stage has involved programming the different platforms, as well as designing the schematics and assembling of device, as it has been seen in chapter 4.
3. Experimentation: Performing laboratory and field tests and its analysis, ensuring the proper operation of the device and obtaining data about the vehicle performance and behaviour, as stated in chapter 5.
4. Writing the thesis: The elaboration of this document has the goal to recap all the basic information about the work developed.

In conclusion, the solution meets the initial objectives as a fully functional prototype has been designed.

7.1 Experience

Developing a project of such a great scope has been a opportunity to improve our skill set:

- learn about historical evolution and operation of motorcycles,

- understand the task of sensors in automotive applications and how they work,
- acquire a deeper knowledge on Arduino controllers and their uses,
- understand the role of open source,
- improve C, Java and CAD (SolidWorks and Eagle) skills,
- learn how to do a good design and understand the consequences of not to,
- understand all the phases involved in product development: time management and design for instance,
- acquire experience in the research field,
- write a technical document,
- overcome problems and constraints.

7.2 Work done

This project was developed using tools from external sources and own resources.

The initial idea is original as it emerges from a personal desire. The main Arduino code is developed from scratch, and the sensor library as well, which is in charge of managing the data gathering. To obtain this data, external libraries have been used but never modified preserving the original source code. If some modifications were required, they have been done through managing internal registers through the sensor library.

The calibration algorithm is based on the recommendations from the creator of the MPU6050 library, as mentioned in its section.

The Bluetooth protocol layer and data logging structure are designed and implemented by ourselves. The Android code is developed from the Bluetooth example from Google [27], but the UI and data processing are created from scratch. The BT example has been modified in order to suit our BT messages structure.

The design of all the electronics schematics, the soldering and assembling, as well as the enclosure design and print are done by own means, assisted when required by the university staff.

The tests were performed with our vehicle and the data has been analysed by ourselves. Same applies to the writing of this document.

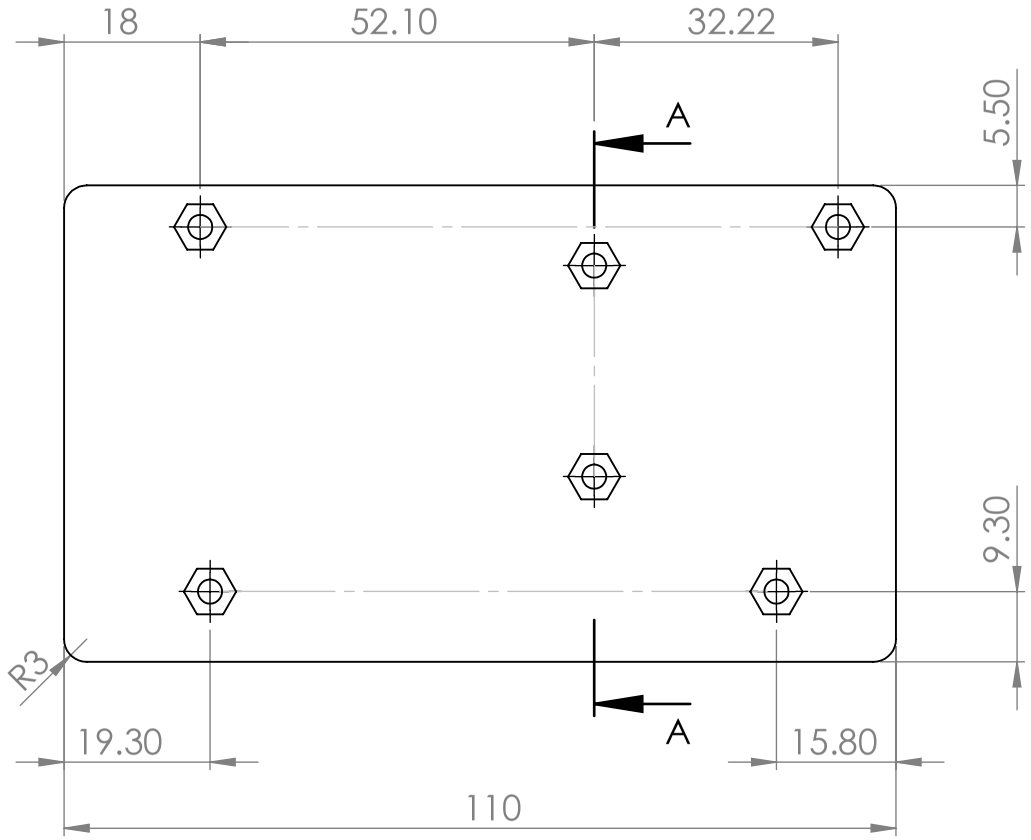
This work has been developed thanks to the guidance of Miquel Massot, who provided assistance and consulting during all the stages of the project, specially during the implementation. Also the council of Dr. Gabriel Oliver, from who I learned to look to the great scope, while being able to focus on simpler tasks.

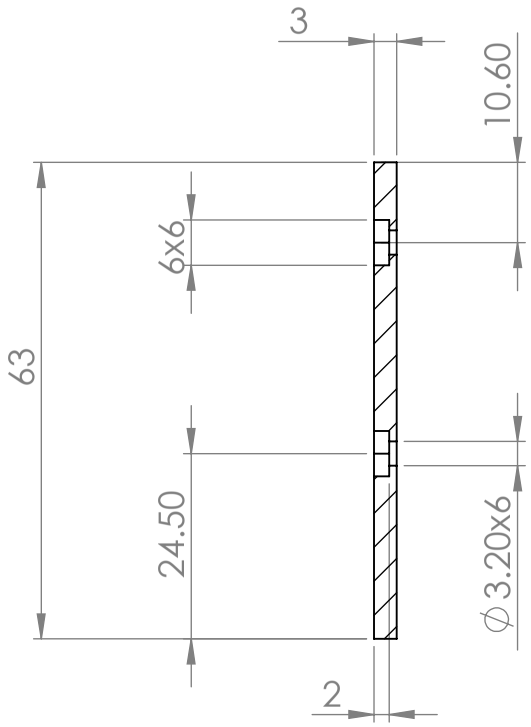
7.3 Further research and development

The development of this project opened different future lines of research and development, such as:

- Continue the data analysis, studying the dynamics and kinematics of motorcycles and how its behaviour in non-controlled environments.
- Connect the data-logger to the cloud. Update a server with the data from the SD, allowing remote access to the motorcycle data. This could be used to track the state of the bike in real time.
- Develop a assistance tool in case of accident. This tool can use the already available data and the BT message structure, using a new command identifier.
- Improve the Android functions and develop an application for iOS devices.
- Create a cloud service, to which we will dump the SD data (by manuals means or through the cloud synchronisation proposed). This service will provide instant data graphs and useful information, while it will display this data using a map service (using the GPS data).
- Extend our project through the combination of the OBDuino [14].
- Study the market and viability of launching the device.
- Use IMU information to enhance motorcycle stability, traction, launch, wheelie and cornering light control. As this is current topic of research in the industry [33].

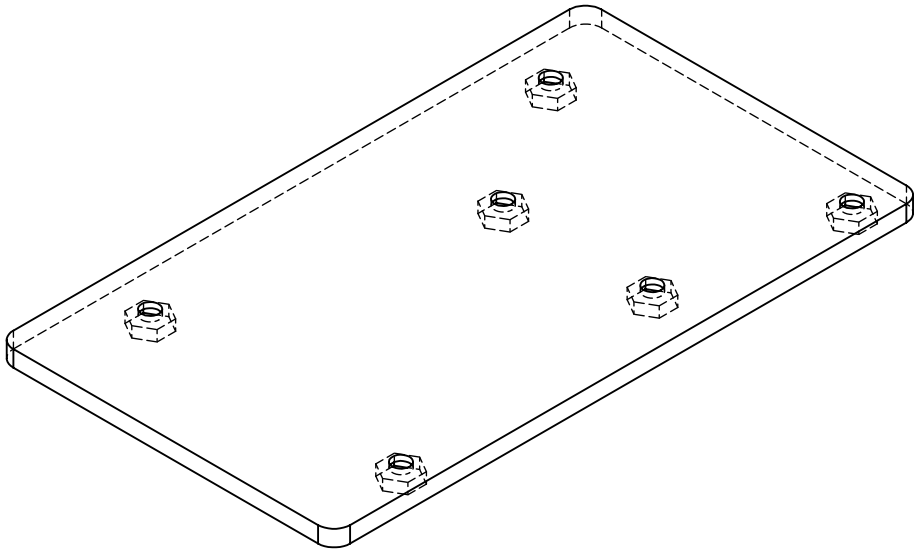
ENCLOSURE PLANS



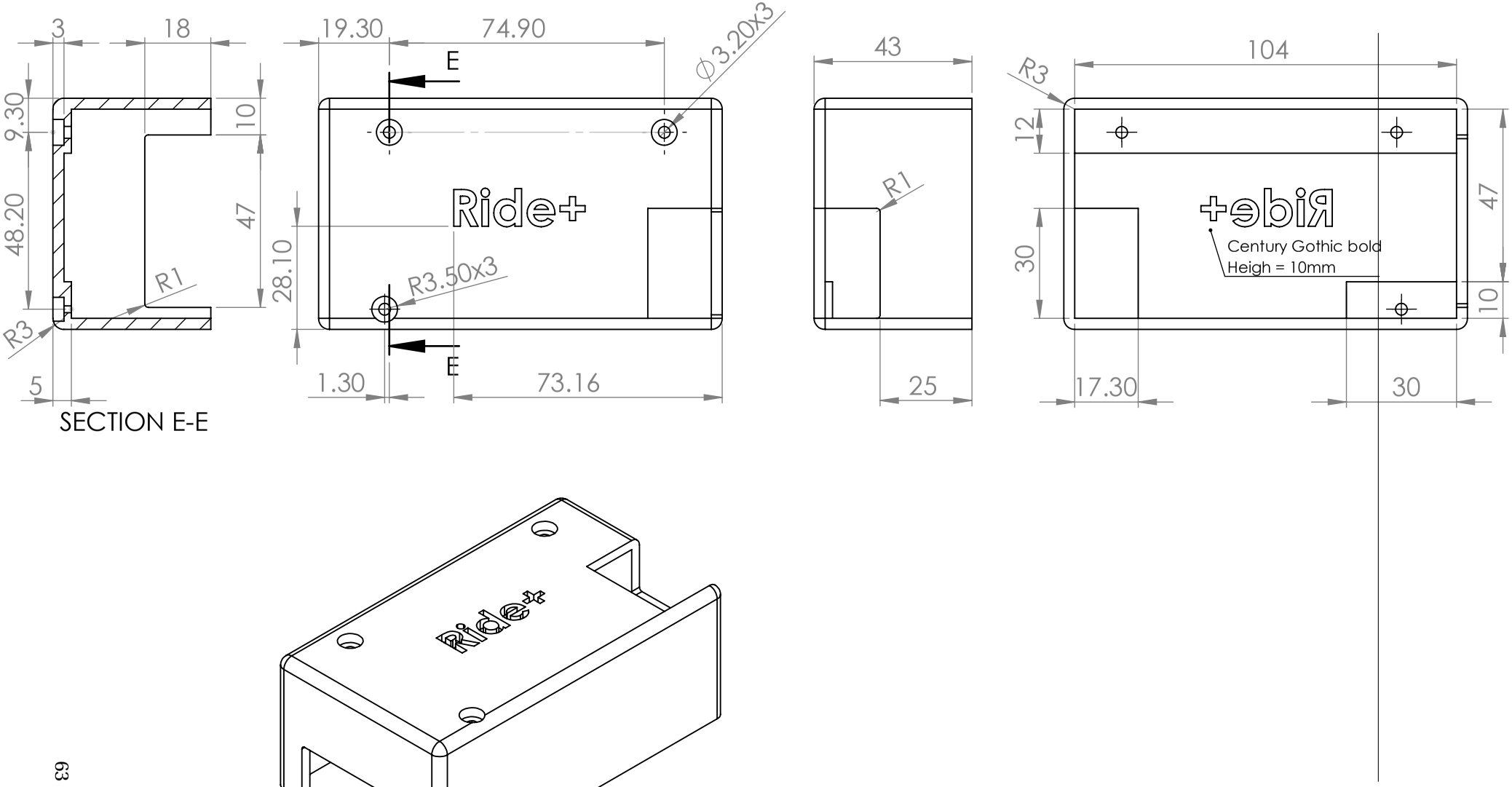


SECTION A-A

A. Enclosure plans



Name: Miquel Font Mas	Revision: 06/09/2017
DWG NO. Bottom	A4
SCALE:1:1	SHEET 1 OF 1



63

Name: Miquel Font Mas	Revision: 06/09/2017
DWG NO. Top	A4
SCALE:2:3	SHEET 1 OF 1

BIBLIOGRAPHY

[1] C. Patsakis and A. Solanas, "Privacy-aware event data recorders: cryptography meets the automotive industry again," *IEEE Communications Magazine*, vol. 51, no. 12, pp. 122–128, dec 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6685767/ 1

[2] T. T. Dandala, V. Krishnamurthy, and R. Alwan, "Internet of Vehicles (IoV) for traffic management," in *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*. IEEE, jan 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7944096/ 1

[3] "CAN bus - technology in detail | BMW Motorrad." [Online]. Available: https://www.bmw-motorrad.co.za/en/engineering/detail/comfort-ergonomics/canbus.html 1.1

[4] "MOTORCYCLE DEALERS Multiple Models YAMAHA DIAGNOSTIC TOOL (YDTv3.1) ESSENTIAL SERVICE TOOL SHIPMENT AND USAGE APRIL 2016 M16-017." [Online]. Available: https://www.yamaha-dealers.ca/yec/en/advertising/docs/M16-017.pdf 1.1

[5] "Honda Motorcycle Communication Interface (MCI) Kit." [Online]. Available: https://honda.snapon.com/HondaCycle/ItemDetail.aspx?itemId=47541382 1.1

[6] E. Abdo, *Modern motorcycle technology*, 2nd ed. Clifton Park N.Y. ;Andover: Delmar Cengage Learning, 2013. [Online]. Available: http://www.worldcat.org/title/modern-motorcycle-technology/oclc/758983882 2.2

[7] A. Wade, *Motorcycle Fuel Injection Handbook*. St. Paul, MN, USA: Motorbooks International, 2004. 2.2

[8] "Smartphone integration: Bosch connects motorcycles - Bosch Media Service." [Online]. Available: http://www.bosch-presse.de/pressportal/de/en/smartphone-integration-bosch-connects-motorcycles-74692.html 2.3.1

[9] "mySPIN for two-wheelers - Bosch SoftTec." [Online]. Available: http://www.bosch-softtec.com/myspin_2w.html 2.3.1

[10] B. Corporation, "Two-wheeler & Powersports - Riding innovation: Comprehensive system solutions and passion for two-wheelers." [Online]. Available: http://www.bosch-motorcycle.com/media/ubk_zweiraeder/related_content/downloads/Two-wheeler_folder_EN_292000P1I5.pdf 2.3.1

[11] "Midas Connect: la primera APP que actualiza tu coche | Midas." [Online]. Available: http://www.midas.es/midas-connect?utm_source=interno{&}utm_medium=banner_midasconnect{&}utm_campaign=midasconnect 2.3.2

[12] "Product | COBI." [Online]. Available: https://cobi.bike/product 2.3.3

[13] "Connected Cycle - The Connected bike enabling solution." [Online]. Available: http://connectedcycle.com/ 2.3.3

[14] "OBDuino Project." [Online]. Available: http://obduino.ca/ 2.3.4, 7.3

[15] "OBD II Bike Connector - Pass via bluetooth." [Online]. Available: https://forum.arduino.cc/index.php?topic=334778.0 2.3.4

[16] "Chippernut Electronic Kits - Chippernut DIY Automotive." [Online]. Available: http://www.chippernut.com/ 2.3.4

[17] MagPi, *The Official Raspberry Pi Projects Book*, Russell Barnes, Ed. London: Liz Upton, 2015. [Online]. Available: https://www.raspberrypi.org/magpi-issues/Projects_Book_v1.pdf 3.2.1

[18] "Power Consumption | Raspberry Pi Dramble." [Online]. Available: https://www.pidramble.com/wiki/benchmarks/power-consumption 3.2.1

[19] M. Banzi, *Getting Started with Arduino*, 3rd ed. Sebastopol, CA: Make:Books, 2014. 3.2.1

[20] "Arduino Platform Webpage." [Online]. Available: https://www.arduino.cc/ 3.2.1, 4.4.4

[21] R. Cayssials, *Sistemas embebidos en FPGA*. Alfaomega, 2014. 3.2.1

[22] Prometec, "Módulo Bluetooth HC-06." [Online]. Available: http://www.prometec.net/bt-hc06/ 4.2.1

[23] Sparkfun, "Serial Communication." [Online]. Available: https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial 4.2.1, 4.2.5

[24] Luis Rodenas and I2Cdevlib Forums, "Calculate MPU6050 offsets." [Online]. Available: https://www.i2cdevlib.com/forums/topic/96-arduino-sketch-to-automatically-calculate-mpu6050-offsets/ 4.2.2

[25] Sparkfun, "Bluetooth Basics." [Online]. Available: https://learn.sparkfun.com/tutorials/bluetooth-basics 4.2.5

[26] "Unix Time Stamp - Epoch Converter." [Online]. Available: https://www.unixtimestamp.com/ 4.2.5

[27] "Bluetooth | Android Developers." [Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth.html 4.3, 7.2

[28] "PCB Design & Schematic Software | EAGLE | Autodesk." [Online]. Available: https://www.autodesk.com/products/eagle/overview 4.4.1

[29] “A Beginner’s guide to making an Arduino Shield PCB | Aaron Eiche.” [Online]. Available: https://aaroneiche.com/2010/06/24/a-beginners-guide-to-making-an-arduino-shield-pcb/ 4.4.1

[30] “How to make an Arduino shield with Eagle CAD - Tutorial | Open Electronics.” [Online]. Available: https://www.open-electronics.org/how-to-make-an-arduino-shield-with-eagle-cad-tutorial/ 4.4.1

[31] “Tinkercad | Create 3D digital designs with online CAD.” [Online]. Available: https://www.tinkercad.com/ 4.4.1

[32] “3D CAD Design Software | SOLIDWORKS.” [Online]. Available: http://www.solidworks.com/ 4.4.1

[33] Bosch, “Inertial measurement unit for motorcycles.” [Online]. Available: http://www.bosch-motorcycle.com/en/de/fahrsicherheit_fuer_zweiraeder/sicherheitssysteme_fuer_zweiraeder/schraeglagensensor_1/lean_angle_sensor.html 7.3