Departamento de Ciências e Tecnologias de Informação

Performance Assessment for Mountain Bike based on WSN and Cloud Technologies

Tiago Miguel Nunes Ribeiro

A Dissertation presented in partial fulfillment of the Requirements for the Degree of Master

Engenharia de Telecomunicações e Informática

Advisor:
Doctor Octavian Postolache, Assistant Professor
ISCTE-IUL

Co-Advisor:
Doctor Pedro Passos, Assistant Professor
FMH-UL

October 2016

Abstract

The mountain bike is one of the most used equipment's in outdoor sports activities. The thesis describes the design and all development and implementation of Performance Assessment for Mountain Bike based on Wireless Sensor Network (WSN) and Cloud Technologies. The work presents a distributed sensing system for cycling assessment-providing data for objective evaluation of the athlete performance during training. Thus a wireless sensor network attached to the sport equipment provides to the athlete and the coach with performance values during practice. The sensors placed in biker equipment's behave as nodes of a WSN. This is possible with the developing of IoT-based systems in sports, the tracking and monitoring of athletes in their activities has an important role on his formation as bikers and helps to increase performance, through the analyze of each session. The implemented system performs acquisition, processing and transmission, of data using a ZigBee wireless networks that provide also machine-to-machine communication and data storage in a server located in the cloud. As in many cycling applications use the phone as a module to get the values, this work will be a little different making use of phone/tablet to consult information. The information stored on the cloud server is accessed through a mobile application that analyses and correlates all metrics calculated using the training data obtained during practice. Additional information regarding the health status may be also considered. Therefore, the system permits that athletes perform an unlimited number of trainings that can be accessed at any time through the mobile application by the bikers and coach. Based on capability of the system to save a history of the evolution of each athlete during training the system permits to perform appropriate comparisons between different training sessions and different athlete's performances.

Keywords: machine-to-machine; bicycle; cloud; wireless sensor network; IoT

Resumo

A bicicleta de montanha é um dos equipamentos para desportos no exterior mais usada. A tese descreve todo o desenho, desenvolvimento e implementação de *Performance Assessment for Mountain Bike based on WSN and Cloud Technologies*. Este apresenta um sistema de deteção distribuída para o aumento do desempenho, melhorar a metodologia da prática do ciclismo e para formação de atletas. Para tal foi desenvolvida e anexada uma rede de sensores que está embutida no equipamento do ciclista, através desta rede de sensores sem fios são obtidos os valores respetivos à interação do utilizador e a sua bicicleta, sendo estes apresentados ao treinador e ao próprio ciclista. Os sensores colocados comportam-se como nós de uma rede de sensores sem fios. Isso é possível com o desenvolvimento de sistemas baseados na Internet das coisas no desporto, a observação da movimentação e monitoramento de atletas nas suas atividades tem um papel importante na sua formação como ciclistas e ajuda a aumentar o desempenho. O sistema é baseado numa rede *ZigBee* sem fios, que permite a comunicação máquina-para-máquina e o armazenamento de dados num servidor localizado na nuvem. Toda a informação na nuvem pode ser acedida através de uma aplicação mobile que analisa e correlaciona todos os valores calculados usando os dados recolhidos durante o treino efetuado por cada ciclista. Como em muitas aplicações de ciclismo estas usam o telefone como um módulo para obter os valores, neste trabalho o caso é diferente fazendo o uso do telefone/*tablet* para apenas consultar as informações. Alguma informação sobre o ciclista é fornecida para poder efetuar alguns cálculos, relativos à saúde do ciclista, neste caso toda a energia gasta na prática de um determinado treino. Toda esta informação pode ser acedida através de uma aplicação *Android* e por consequência num dispositivo *Android*. Com a aplicação desenvolvida é possível observar e processar toda a informação recolhida através dos sensores implementados, a observação dos dados recolhidos pode ser efetuada pelo treinador responsável, como pelo próprio atleta. Portanto, o sistema permite a realização de um ilimitado número de sessões de treino, estes podem ser consultados a qualquer momento através da aplicação móvel. Fazendo com que seja possível manter um histórico da evolução de cada atleta, podendo assim observar e comparar cada sessão de treino, realizada por cada atleta.

Palavras-chave: máquina para máquina, bicicleta, nuvem, rede de sensores sem fios; Internet das Coisas;

Acknowledgments

I would first like to thanks to my advisor and co-advisor for the opportunity they gave me to do this project "Performance Assessment for Mountain bike based on WSN and Cloud Technologies".

Firstly, I would like to express my sincere gratitude to my advisor Professor Octavian Postolache for continuous support and committed on the development of this work. Throughout the year he was always available to help under any circumstance.

I wish to express my sincere thanks to my co-advisor Professor Pedro Passos for his contributions to this work, whose opinions were very important.

To both of them I would like to thanks for the continuous support on the development of this work, for his patience, motivation, immense knowledge, and great ideas.

To Instituto de Telecomunicações on ISCTE-IUL, thanks for providing me all the material and resources I needed to implement this demanding work.

I would like to thank the members of Arduino Forum and StackOverflow communities for their invaluable assistance in many a late-night coding session.

Last but not least, I must express my very profound gratitude to my family and to my close friends for providing me with unfailing support, great moments and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you all.

Figures

Tables

List of Acronyms

2G	2nd Generation
3G	3rd Generation
4G	4th Generation
AHRS	Altitude and Heading Reference System
AP	Access Point
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
CPU	Central Processing Unit
CSMA/CD	Carrier sense multiple access with collision detection
DB	Data base
DDR2	Double Data Rate 2
DDR3	Double Data Rate 3
DOF	Degree Of Freedom
DSSS	Direct Sequence Spread Spectrum
EDGE	Enhanced Data Rates for GSM Evolution
EEPROM	Electrically-Erasable Programmable Read-Only Memory
FDDI	Fiber Distributed Data Interface
FFD	Full-function device
FHSS	Frequency-Hopping Spread Spectrum
FSR	Force Sensitive Resistor
FTDI	Future Technology Devices International
G	2.5 Generation
GPIO	General-purpose input/output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile Communications
HDMI	High-Definition Multimedia Interface
HF	High Frequency
HSPA	High Speed Packet Access
HTTP	HyperText Transfer Protocol
I^2C	Inter-Integrated Circuit
I^2S	Integrated Interchip Sound
ICSP	In Circuit Serial Programming
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
IoT	Internet of Things
LAMP	Linux, Apache, MySQL and PHP
LAN	Local Area Network
LCD	Liquid-crystal Display
LET	Long-term Evolution
LF	Low Frequency
M2M	Machine-to-Machine
MEMS	Microelectromechanical Systems
MET	Metabolic Equivalent of Task

OFDM	Orthogonal frequency-division multiplexing
ORD	Object-relational Database
PCB	Printed Circuit Board
PHP	Hypertext Preprocessor
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RF	Radio Frequency
RFD	Reduced-function device
RFID	Radio-frequency identification
RMR	Resting Metabolic Rate
SCL	Serial Clock Line
SD card	Secure Digital Card
SDA	Serial Data Line
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on Chip
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random Access Memory
SSH	Secure Shell
TIFF	Time-to-First-Fix
UART	Universal Asynchronous Receiver/Transmitter
UHF	Ultra High Frequency
UMTS	Universal Mobile Telecomunications System
USB	Universal Serial Bus
VHF	Very High Frequency
WAN	Wide Area Network
WAP	Wireless Application Protocol
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

Contents

Chapter 1 - Introduction

1. Motivation and Overview

The history of bicycle remote us to the year of 1418 where the engineer Giovanni Fontana built a four wheeled "bike" with rope connected by gears. Only 400 years later, in response to the starvation and the slaughtering of horses Baron Von Drais built the first two wheeled bike that has a cord connected to the back wheel. These velocipedes were made entirely of wood and needed to be balanced by directing the front wheel a bit. People then did not dare to lift the feet off safe ground, therefore the velocipedes were propelled by pushing off the feet. Only in 1890 appeared a bicycle which had the name "Safety Bike" with the same design and concept that we can see nowadays (same-size wheels, pedals, gears and bike saddle) (Mozer, 2016). The years passed and the innovations were many, as the pedals, brakes, suspension and the lighter and comfortable materials that we see today in our bikes. Nowadays bicycles can be classified in four types: urban bike, BMX, road bikes and mountain bikes (Sanches, 2015).

For the conception of this work will be used a mountain bike, these bicycles are typically used in single tracks where the terrains can be unpaved. Like in many sports the choice is immense in terms of materials that the bicycles are made and to get the best performance, normally the athletes choose the best that fits to his needs (considering the tracks and is own physical performance) ("Bikes | Trek Bikes", 2016). Because on these tracks commonly we could encounter rocks, loose gravel, roots and step grades inclines and declines. Under these conditions the biker will need to get a functional interaction with his "machine", in order to achieve a balance between speed and safety. The mountain bike is produced to handle with this type of tracks, normally the material that is widely used is aluminum or carbon fiber (offers a lighter stiffer and efficient ride) ("Understanding Bike Frame Materials - REI Expert Advice", 2016). The modern and the most common frame design for an upright bicycle is based on the "safety bike", and consists of two triangles, a main triangle and a paired rear triangle. To control the bicycle along the track the biker needs to use the mechanisms with his hands (e.g., brakes) and feet (e.g., pedals), and the positioning of his body is also fundamental. A functional position will permit the biker make rapid changes of direction and get the desirable speed. The biker position could be essential to deal with some external factors such as the slope of the track and its obstacles. It is the interactive behavior of the bicycle and biker which wanted to acquire, dynamic and kinematic data through a WSN was designed and implemented. These WSN permits to get values to some of the following variables: i) braking intensity and frequency (both hands); ii) pedal strength (both feets); iii) position of bicycle in the three plans of motion (x, y and z); iv) position of biker in the three plans of motion. With this set of variables, it was possible to calculate some important aspects to describe the bicycle ~ biker interaction.

For the athletes the training is the most important way to achieve a better performance whether professional or recreational level. In this era of the Internet of Things (IoT), where vehicles, buildings and a lot of another's things are characterized by sensors, software and network connectivity that enable the data collection and data exchange remotely across existing network. This developed system will allow the users to obtain data from his own training session. The acquired data with this sensors network, can be analyzed and adapt the training methods and improve the performance.

1.1. Internet of Things (IoT)

Internet of Things (IoT) is an environment in which objects, animals or people are provided with unique identifiers and the ability to transfer data over a network without requiring human interaction. IoT evolved through the convergence of wireless technologies, micro-electromechanical systems (MEMS) and the internet.

In the term Internet of things, a thing can be (fig. 1-1), a person with a heart monitor implant, an animal in a farm with a biochip transponder (like the same the vet put in our pets), or a object (sensor) in our car ("What is Internet of Things (IoT) | Engineers Gallery| Technology", 2015), (Stočes, Vaněk, Masner, & Pavlík, 2016).

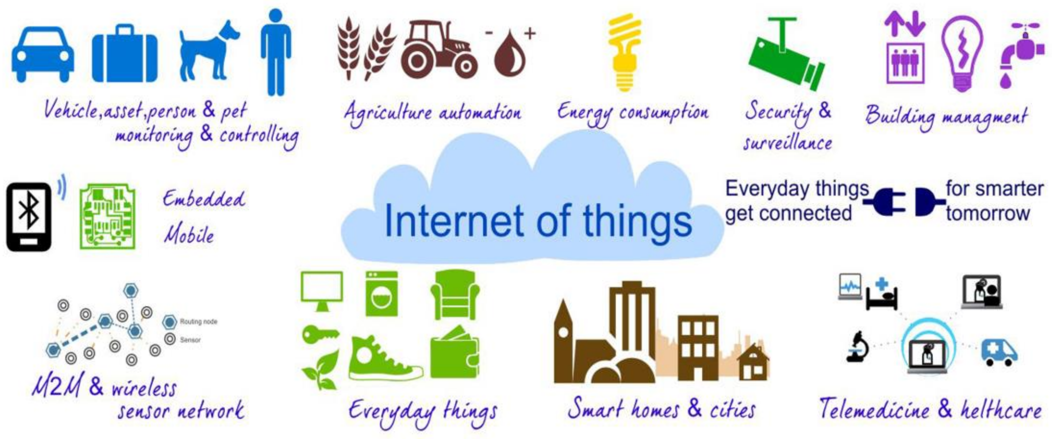


Figure 1-1 - Internet of Things (IoT)

1.2.State of art

On the market many applications about cycling are produced and sent to market by companies or by individual developers, they have in common two aspects they use the phone and all information about the training it is obtained from the position of the phone. They don't have information about the aspects of the biker and his bicycle. So the innovation purposed in this work is to create WSN to capture some data about the interaction of the biker and his bicycle. Some work in this research field was performed in time and the prototype with limited characteristics was developed. (Barreiro, Postolache, & Passos, 2014).

1.2.1. Embedded Systems

An embedded system is an engineering artifact involving computation that is subject to physical constraints. The physical constraints arise through two kinds of interactions of computational processes with the physical world: (1) reaction to a physical environment, and (2) execution on a physical platform. Accordingly, the two types of physical constraints are reaction constraints and execution constraints (Henzinger & Sifakis, 2006).

So an embedded system is a combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular function. In our days there are a huge number of hardware that support software and can act as embedded system.

Below are presented some examples of embedded systems studied and created:

Towards a Real-Time Embedded System for Water Monitoring Installed in a Robotic Sailboat (Goncalves, Thomaz, Sa, & Henrique, 2016).

Network camera with FPGA technology (Guedes, Neto, & Véstias, 2007)

Design and construction of an embedded real-time system based on Linux (Campos, Fonseca, & Lopes, 2011).

1.2.2. The Arduino Family

Arduino makes several different boards, each with different capabilities, is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board referred as microcontroller.
In addition, part of being open source hardware means that others can modify and produce derivatives of Arduino boards that provide even more form factors and functionality. Arduino

hardware consists of an open hardware design with Atmel AVR processor ("What is an Arduino? - learn.sparkfun.com", 2016). Boards can be purchased preassembled, but hardware design information is also available for those willing to build or modify them. Several third-party makers have produced Shields (add-on boards) that are able to extend the basic capabilities of an Arduino. Among these shields, it is worth mentioning the XBee shield allows multiple Arduino boards to communicate wirelessly (Bell, 2013).

Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
101	Intel® Curie	3.3 V/ 7-12 V	32MHz	6/0	14/4	-	24	196	Regular	-
Gemma	ATtiny85	3.3 V/ 4-16 V	8MHz	1/0	3/2	0.5	0.5	8	Micro	0
LilyPad	ATmega168V ATmega328P	2.7-5.5 V / 2.7V-5.5 V	8 MHz	6/0	14/6	0.512	1	16	-	-
LilyPad SimpleSnap	ATmega 328P	2.7-5.5 V / 2.7-5.5 V	8 MHz	4/0	9/4	1	2	32	-	-
LilyPad USB	ATmega32U4	3.3 V / 3.8-5 V	8 MHz	4/0	9/4	1	2.5	32	Micro	-
***Mega 2560**	ATmega2560	5 V / 7-12V	16MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32U4	5V / 7-12V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
MKR1000	SAMD21 Cortex-M0+	3.3 V/ 5V	48 MHz	7/1	8/4	-	32	256	Micro	1
Pro	ATmega168 ATmega328P	3.3 V / 3.35-12 V 5V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512 1	1 2	16 32	-	1
Pro Mni	ATmega328P	3.3 V / 3.35-12V 5V / 7-12 V	8 MHz 16 MHz	6/0	14/6	1	1	32	-	1
Uno	ATmega 328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Zero	ATSAMD21 G18	3.3 V / 7-12 V	48 MHz	6/1	14/10	-	32	256	2 Micro	2
Due	ATSAM3X8E	3.3 V / 7-12 V	84 MHz	12/2	54/12	-	96	521	2 Micro	4
BT	ATmega328P	5 V / 2.5-12 V	16 MHz	6/0	14/6	1	2	32	-	1
Esplora	ATmega 328P	5 V/ 7-12 V	16 MHz	-	-	1	2.5	32	Micro	-
Ethernet	ATmega328P	5 V / 7-12 V	16 MHZ	6/0	14/4	1	2	32	Regular	-
***Fio**	ATmega328P	3.3 V / 3.7-7 V	8 MHz	8/0	14/6	1	2	32	Mini	1
Leonardo	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
Mega ADK	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Mini	ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	1	2	32	-	-
Nano	ATmega168 ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	0.512 1	1 2	16 32	Mini	1
Yún	ATmega32U4 AR9331 Linux	5 V	16 MHz 400MHz	12/0	20/7	1	2.5 16MB	32 64MB	Micro	1

Table 1 - Arduino compare board specs

1.2.3. The BeagleBone Family

The Beagle boards are open-hardware, open-software computers. The BeagleBone is one of the best tools to use to discover embedded programming and electronics. It's a good way to see and understand more closely how a computer works (Santos & Perestrelo, 2015). The BeagleBone is a compact, low-cost, open-source Linux computing platform that can be used to build complex applications that interface high-level software and low-level electronic circuits. BeagleBone runs the Linux operating system, which means that is possible use many open-source software libraries and applications directly. This platform is formed by the integration of a high-performance microprocessor on a printed circuit board (PCB) and an extensive software ecosystem (Molloy, 2015).

	BeagleBoard.org BeagleBone Black	BeagleBoard.org BeagleBone (original)	SeedStudio BeagleBone Green	SanCloud BeagleBone Enhanced
Processor	AM3358 ARM Cortex-A8	AM3358 ARM Cortex-A8	AM3358 ARM Cortex-A8	AM3358 ARM Cortex-A8
Maximum Processor Speed	1GHz	720MHz (1GHz on latest)	1GHz	1GHz
Analog Pins	7	7	7	7
Digital Pins	65 (3.3V)	65 (3.3V)	65 (3.3V)	65 (3.3V)
Memory	512MB DDR3 (800MHz x 16), 2GB (4GB on Rev C) onboard storage using eMMC, microSD card slot	256MB DDR2 (400MHz x 16), microSD card slot	512MB DDR3 (800MHz x 16), 4GB on-board storage using eMMC, microSD card slot	1GB DDR3 (800MHz x 16), storage using eMMC, microSD card slot
USB	miniUSB 2.0 client port, USB 2.0 host port	miniUSB 2.0 client port, USB 2.0 host port	microUSB 2.0 client port, USB 2.0 host port	miniUSB 2.0 client port, 4 USB 2.0 Ports (2 A-type connectors, 2 on pin headers)
Video	microHDMI, cape add-ons	cape add-ons	Cape add-ons	microHDMI, cape add-ons
Audio	microHDMI, cape add-ons	Cape add-ons	Cape add-ons	microHDMI, cape add-ons
Supported Interfaces	4x UART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I^{2}C, A/D Converter, 2xCAN Bus, 4 Timers	4x UART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I^{2}C, A/D Converter, 2xCAN Bus, 4 Timers, FTDI USB to Serial, JTAG via USB	4x UART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I^{2}C, A/D Converter, 2xCAN Bus, 4 Timers, 2 Grove (I2C, UART)	4x UART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I^{2}C, A/D Converter, 2xCAN Bus, 4 Timers
Sensors	n/a	n/a	n/a	Barometer, Accelerometer, Gyro, Temperature

Table 2 - BeagleBone compare board specs

1.2.4. The Raspberry Pi Family

The Raspberry Pi is a credit card sized single-board computer developed in the UK by the Raspberry Pi Foundation with the intention of stimulating the teaching of basic computer science in schools (Molloy, 2016). The first Raspberry Pi has a Broadcom BCM2835 system on chip (SoC), which includes an ARM1176JZF-S 700 MHz processor, VideoCore IV GPU, and was originally shipped with 256 megabytes of RAM, later upgraded to 512 MB and on the present the boards already have 1024 MB. It does not include a built-in hard disk or solid-state drive, but uses an SD card for booting and persistent storage.

	Raspberry Pi 1				Raspberry Pi 2	Raspberry Pi 3	Compute Module	Raspberry Pi Zero	
Model	Model A	Model A+	Model B	Model B +	Model B	Model B	N/A	PCB v1.2	PCB v1.3
Architecture	ARMv6 (32 bit)				ARMv7 (32-bit)	ARMv8 (64/32-bit)	ARMv6 (32-bit)		
SoC	Broadcom BCM2835				Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2835		
CPU	700 MHz single-core ARM1176JZF-S				900MHz 32-bit quad-core ARM Cortex-A7	1.2 GHz 64-bit quad-core ARM Cortex-A53	700 MHz single-core ARM1176JZF-S	1 GHz ARM1176JZF-S single-core	
GPU	Broadcom VideoCore IV @ 250 MHz (BCM2837: 3D part of GPU @ 300MHz, video part of GPU @ 400 MHz) OpenGL ES 2.0 (BCM2835, BCM2836: 24 GFLOPS / BCM2837: 28.8 GFLOPS)								
Memory (SDRAM)	256MB	512 MB (shared with GPU). Older boards had 256MB (shared with GPU)			1 GB (shared with GPU)		512 MB (shared with GPU)		
USB 2.0 ports	1 (direct from BCM2835 chip)		2 (via the on-board 3-port USB hub)	4 (via the on-board 5-port USB hub)			1 (direct from BCM2835 chip)	1 Micro-USB (direct from BCM2835 chip)	

Table 3 - Raspberry Pi compare board specs (part. 1)

	Raspberry Pi 1				Raspberry Pi 2	Raspberry Pi 3	Compute Module	Raspberry Pi Zero	
Model	Model A	Model A+	Model B	Model B +	Model B	Model B	N/A	PCB v1.2	PCB v1.3
Video Input	15-pin MIPI camera interface (CSI) connector, used with the Raspberry Pi camera or Raspberry Pi NoIR camera						2x MIPI camera interface (CSI)	None	MIPI camera interface (CSI) (rev 1.3)
Video Output	HDMI (rev 1.3) composite video (RCA jack)	HDMI (rev 1.3), composite video (3.5 mm TRRS jack)	HDMI (rev 1.3), composite video (RCA jack)	HDMI (rev 1.3), composite video (3.5 mm TRRS jack)			HDMI, 2x MIPI display interface (DSI) for raw LCD panels, composite video	Mini-HDMI, 1080p60, composite video via GPIO	
Audio inputs	As of revision 2 boards via I^{2}S								
Audio outputs	Analog via 3.5 mm phone jack; digital via HDMI and, as of revision 2 boards, I^{2}S						Analog, HDMI, I^{2}S	Mini-HDMI, stereo audio through PWM on GPIO	
On-board storage	SD, MMC, SDIO card slot (3.3 V with card power only)	MicroSDHC slot	SD, MMC, SDIO card slot	MicroSDHC slot			4 GB eMMC flash memory chip	MicroSDHC	

Table 4 - Raspberry Pi compare board specs (part. 2)

	Raspberry Pi 1				Raspberry Pi 2	Raspberry Pi 3	Compute Module	Raspberry Pi Zero	
Model	Model A	Model A+	Model B	Model B +	Model B	Model B	N/A	PCB v1.2	PCB v1.3
On-board network	None		10/100 Mbit/s Ethernet (8P8C) USB adapter on the USB hub			10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1		None	
Low-level peripherals	8x GPIO plus the following which can also be used as GPIO: UART, I^{2}C bus, SPI bus with two chip selects, I^{2}S audio +3.3 V, +5 V, ground	17x GPIO plus the same specific functions, and HAT ID bus	8x GPIO plus the following, which can also be used as GPIO: UART, I^{2}C bus, SPI bus with two chip selects, I^{2}S audio +3.3 V, +5 V, ground. An additional 4x GPIO are available on the P5 pad if the user is willing to make solder connections	17x GPIO plus the same specific functions, and HAT ID bus			46x GPIO, some of which can be used for specific functions including I^{2}C, SPI, UART, PCM, PWM	40x GPIO ("unpopulated header")	

Table 5 - Raspberry Pi compare board specs (part. 3)

	Raspberry Pi 1				Raspberry Pi 2	Raspberry Pi 3	Compute Module	Raspberry Pi Zero	
Model	Model A	Model A+	Model B	Model B +	Model B	Model B	N/A	PCB v1.2	PCB v1.3
Power Ratings	300mA (1.5 W)	200 mA (1 W)	700 mA (3.5 W)	600 mA (3.0 W)	800 mA (4.0 W)		200 mA (1 W)	~160 mA (0.8 W)	
Power Source	5 V via MicroUSB or GPIO header								

Table 6 - Raspberry Pi compare board specs (part. 4)

1.2.5. Tactile Sensing Technologies

The "sense of touch" in humans comprises two main sub modalities: cutaneous and kinesthetic (neural inputs). Cutaneous sense receives sensory inputs from the receptors embedded in the skin and kinesthetic sense receives sensory inputs from the receptors located in muscles, tendons and joints. Cutaneous system involves physical contact with the stimulation and provides the awareness of the stimulation of the outer surface of body by means and receptors in the skin associated somatosensory area of central nervous system. The kinesthetic system provides humans information about the static and dynamic body posture on the basis of afferent information originating from the muscles, joints and skin. Human tactile sensing has generally served as a reference point for tactile sensing in robotics. Even though human tactile sensing has been a reference point for robotic tactile sensing, the way tactile sensing is defined in robotics falls short of what it means in humans. Most of the times, the robotic tactile sensing has been associated with the detection and measurement of forces in a predetermined area only. The tactile or cutaneous sensing is associated with the detection and measurement of contact parameters, which can be mechanical stimulation (force, stress, roughness etc.), temperature, moistness etc. The definition of a pressure sensor by Lee et al. (Almassri et al., 2014) is a device that measures a physical quantity and converts it into a signal which can be read by an observer or an instrumented. There are various types of sensors: thermal sensor, electromagnetic sensor, pressure sensor, mechanical sensor and others. In the case of pressure is sensed by mechanical elements such as plates, shells, and tubes that are designed and constructed to deflect when pressure is applied (Fässler, 2010).

Technology	Type	Advantages	Disadvantages
Mechanical	Whisker / Antenna	Simple; Robust; Can measure touch location	Bad spatial resolution: Can not measure force
	Mechanical displacement	Simple; Robust; Inexpensive	Limited spatial resolution
	Pneumatic touch sensor / Foil switches	Simple; Robust; Can measure touch location; Inexpensive	Can not measure force
	Digital tactile sensor array	No analog to digital conversion	Prone to damage
Capacitive		Good sensitivity; Moderate hysteresis; Wide dynamic range; Linear response; Robust	Complex circuitry; Susceptible to noise; Limited spatial resolution; some dielectrics are temperature sensitive
Strain Gauges	Metal Strain Gauges	More robust than semiconductor strain gauges	Temperature dependence; Small k-factor compared to semiconductor strain gauges
	Semiconductor strain gauge	Very linear response; Low hysteresis; Low creep; Large k-factor	Vulnerable to overload; Can not be shaped; Temperature dependence
Piezoresistive	Conductive elastomers	Shapeable; Good gripping surface	Creep; Memory; Hysteresis; Temperature dependence
	Carbon felt and Carbon fibers	Shapeable; Withstand very high temperatures; Withstand high overloads	Sensor noise at low loads; Not suited for miniature sensors
Piezoelectric		Wide dynamic range; Durability; Good mechanical material properties	Fragility of electrical junctions; Inherently dynamic; Good solutions are complex; Difficulty of separating pyroelectric / piezoelectric effects

Table 7 - General advantages and disadvantages of different sensor technologies (part.1)

Technology	Type	Advantages	Disadvantages
Pyroelectric		Wide dynamic range; Durability; Good mechanical material properties	Inherently dynamic; Good solutions are complex; Difficulty of separating pyroelectric / piezoelectric effects
Optical	Frustrated internal reflection	Shapeable; Very high resolution tactile images	Bulky; Complex construction
	Opto-mechanical	Good repeatability	Creep; Memory; Hysteresis; Temperature dependence
	Fiber-optic	Lower noise than the above; Flexible	Complex construction
	Photoelasticity	Good results in terms of linearity, Hysteresis, Creep and memory	Complicated optic system; Not shapeable
	Tracking of optical markers	No interconnects to break	Requires PV for computing applied forces; Hard to customize
Magnetic	Hall effect	Wide dynamic range; Low hysteresis; Linear response; Robust	Measure field in only 1 direction
	Magnetoelastic	Simpler then hall effect; Measures field in 2 directions; Wide dynamic range; Low hysteresis; Linear response; Robust	Susceptibility to stray fields and noise
Ultrasonic		Wide dynamic range; Good spatial resolution	Can not measure when touching material has similar acoustic impedance as skin material
Electrochemical			No steady state response; Bad spatial resolution; Low sensitivity

Table 8 - General advantages and disadvantages of different sensor technologies (part.2)

1.2.6. Inertial Measurement Unit

An Inertial measurement unit, or IMU, is the main component of inertial guidance systems used in air space, and watercraft, including guided missiles. An IMU works by sensing motion including the type, rate, and direction of that motion using a combination of accelerometers and gyroscopes. Accelerometers are placed such that their measuring axes are orthogonal to each other. An IMU works by detecting the current rate of acceleration, as well as it changes in rotational attributes, including pitch, roll and yaw (Hazry, Sofian, & Azfar, 2009), ("A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. « Starlino Electronics", 2009), (Woodman, 2007).

IMUs available in the market now are in various types and shape. The IMU can be selected from its degrees of freedom (DOF). Actually is possible encounter boards with three, six, nine and ten DOF. For three DOF, sensors configurations are two accelerometers and a gyroscope that measures yaw. For six DOF, all axes for accelerometer and gyroscope for measurement are available (pitch and roll). For nine DOF, all axes are measure for accelerometer, gyroscope and magnetometer (pitch, roll and yaw). For ten DOF, all axes for accelerometer and gyroscope for measurement are available (pitch, roll and yaw), but the IMU have a barometer too (Performance, n.d.).

Degrees of Freedom	Function
3 DOF	Accel/gyro
6 DOF	Accel/gyro
9 DOF	Accel/gyro/magn
10 DOF	Accel/gyro/magn/baro

Table 9 - IMU configurations DOF

1.2.7. Communications

A Computer network is a collection of computers and devices interconnected by communications channels that facilitate communications and allows sharing of resources and information among interconnected devices. Networks may be classified according to a wide variety of characteristics such as medium used to transport data.

Types of Networks

- Local Area Networks (LANs)

- Wide Area Networks (WANs)
- Wireless Local Area Networks (WLANs)

Ethernet is the most popular physical layer LAN technology in use today. It defines the number of conductors that are required for a connection, the performance is an important point on networks because the data transmission and communication needs to be always operational. A standard Ethernet network can transmit data at a rate up to 10 Megabits per second (10 Mbps). Other LAN types include Token Ring, Fast Ethernet, Gigabit Ethernet, Fiber Distributed Data Interface (FDDI), Asynchronous Transfer Mode (ATM) and LocalTalk. Ethernet is popular because it strikes a good balance between speed, cost and ease of installation.

Name	IEEE Standard	Data Rate	Media Type	Maximum Distance
Ethernet	802.3	10 Mbps	10Base-T	100 meters
Fast Ethernet/ 100Base-T	802.3u	100 Mbps	100Base-TX	100 meters
			100Base-FX	2000 meters
Gigabit Ethernet/ GigE	802.3z	1000 Mbps	1000Base-T	100 meters
			1000Base-SX	275/550 meters
			1000Base-LX	550/5000 meters
10 Gigabit Ethernet	IEEE 802.3ae	10 Gbps	10GBase-SR	300 meters
			10GBase-LX4	300m MMF/ 10km SMF
			10GBase-LR/ER	10km/40km
			10GBase-SW/LW/EW	300m/10km/40km

Table 10 - LAN Technology Specifications

Token Ring is another form of network configuration. It differs from Ethernet in that all messages are transferred in one direction along the ring at all times. Token Ring networks sequentially pass a "token" to each connected device. When the token arrives at a particular computer or device, the recipient is allowed to transmit data onto the network. Since only one device may be transmitting at any given time, no data collisions occur.

Wireless LANs, or WLANs, use radio frequency (RF) technology to transmit and receive data over the air. WLANs give users mobility as they allow connection to a local area network without having to be physically connected by a cable. With mobility, WLANs give flexibility and increase productivity, appealing to both entrepreneurs and to home users. The Institute for Electrical and Electronic Engineers (IEEE) developed the 802.11 ("Performance of Wireless Networks: WiFi - High Performance Browser Networking (O'Reilly)", 2016) specification for wireless LAN technology. 802.11 specifies over-the-air interface between a wireless client and a base station. WLAN 802.11 standards also have security protocols (Wired Equivalent Privacy - WEP and Wi-Fi Protected Access - WPA) that were developed to provide the same level of security as that of a wired LAN ("Ethernet Tutorial - Part I: Networking Basics | Lantronix", 2016).

Specification	Data Rate	Modulation Scheme	Security
802.11	1 or 2 Mbps in the 2.4 GHz band	FHSS, DSSS	WEP and WPA
802.11a	54 Mbps in the 5 GHz band	OFDM	WEP and WPA
802.11b/High Rate/Wi-Fi	11 Mbps (with a fallback to 5.5, 2, and 1 Mbps) in the 2.4 GHz band	DSSS with CCK	WEP and WPA
802.11g/Wi-Fi	54 Mbps in the 2.4 GHz band	OFDM when above 20Mbps, DSSS with CCK when below 20Mbps	WEP and WPA

Table 11 - Wireless Protocols

In terms of the cellular networks, the data is transmitted not to a central hub in a small network of devices (as it is with Wi-Fi) or even directly from device to device (as it with Bluetooth), but through a global network of transmitters and receivers. A typical cellular network can be envisioned as a mesh of hexagonal cells, as shown in fig. 1-2, each with its own base station at the center. The cells slightly overlap at the edges to ensure that users always remain within range of a base station (Miller, 2013).

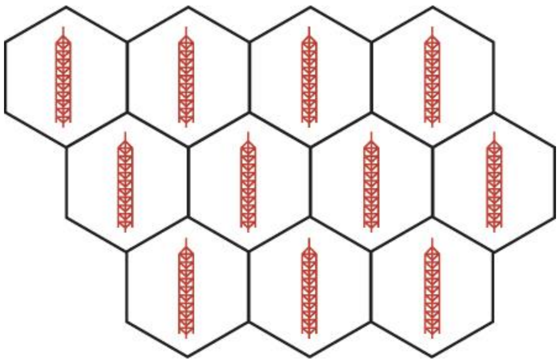


Figure 1-2 - Cells in a cellular network

The base station at the center of each group of cells functions as the hub for those cells not of the entire network, but of that individual piece of the network. RF signals are transmitted by an individual phone and received by the base station, where they are then re-transmitted from the base station to another mobile phone. Transmitting and receiving are done over two slightly frequencies. Base stations are connected to one another via central switching centers which track calls and transfer them from one base station to another as callers move between cells; the handover is (ideally) seamless and unnoticeable. Each base station is also connected to the main telephone network, and can thus relay mobile calls to landline phones (Pearson, 2011), (Pereira & Sousa, 2004), (Akyildiz, Gutierrez-Estevez, & Reyes, 2010), (Jary, 2014).

Symbol	Standard	Full Name	Maximum Download Speed (Theoretical)	Maximum Upload Speed (Theoretical)
2G	GSM	Global System for Mobile Communications	14.4 Kbits/s	14.4 Kbits/s
G	GPRS	General Packet Radio Service	53.6 Kbits/s	28.8 Kbits/s
E	EDGE	Enhanced Data rates for GSM Evolution	271.6 Kbits/s	108.8 Kbits/s
3G	UMTS	Universal Mobile Telecommunications System	384 Kbits/s	128 Kbits/s
H	HSPA	High-Speed Packet Access	7.2 Mbits/s	3.6 Mbits/s

Table 12 - Cellular Technologies (part.1)

Symbol	Standard	Full Name	Maximum Download Speed (Theoretical)	Maximum Upload Speed (Theoretical)
H+	HSPA+	Evolved High-Speed Packet Access – Release 6	14.4 Mbits/s	5.76 Mbits/s
H+	HSPA+	Evolved High-Speed Packet Access – Release 7	21.1 Mbits/s or 28 Mbits/s	11.5 Mbits/s
H+	HSPA+	Evolved High-Speed Packet Access – Release 8	42.2 Mbits/s	11.5 Mbits/s
H+	HSPA+	Evolved High-Speed Packet Access – Release 9	84.4 Mbits/s	11.5 Mbits/s
H+	HSPA+	Evolved High-Speed Packet Access – Release 10	168.8 Mbits/s	23 Mbits/s
4G	LTE	Long Term Evolution	100 Mbits/s	50 Mbits/s
4G	LTE-A	Long Term Evolution - Advanced	1 Gbits/s	500 Mbits/s

Table 13 - Cellular Technologies (part.2)

1.2.8. Machine-to-Machine (M2M)

The term M2M communication describes devices that are connected to the Internet, using a variety of fixed and wireless networks and communicate with each other and the wider world. The term is slightly erroneous through as it seems to assume there is no human in the equation, which quite often there is in one way or another (Oecd, 2012). So M2M communications is an emerging technology that envisions the interconnection of machines without the need of human interventions. The main concept lies in seamlessly connecting an autonomous and self-organizing network, of M2M-capable devices to a remote client, through wired or wireless network. The M2M is a combination of various heterogeneous electronic communication, and software technologies. An software application is usually employed at the remote client to process the collect data and provide the end user with a set of smart services and a practical interface. The penetration of M2M solutions for monitoring and remote control in a wide range of markets, including industrial automation, security and surveillance, smart metering, energy management, and transportation generates great business opportunities. The above challenges stress the imperative need for standardization of M2M communications (Kartsakli et al., 2014). The ability to connect new devices to the network lead to the evolution of the IoT. The Internet will be no longer just a network of computers, but will potentially involve trillions of smart things with embedded systems. IoT will greatly increase the size and scope of current Internet, providing new design opportunities and challenges (Zeng, Guo, & Cheng, 2011). In this scenario, Cloud Computing (Zhang, Cheng, & Boutaba, 2010) can be seen as a scalable infrastructure (Höfer & Karagiannis, 2011) that supports computing power, storage and software services.

1.2.9. Wireless Sensor Network

With the advances in Wireless Sensor Networks (WSN), the use of these networks for collect and interpret data in real time was facilitated. In the past they were typically used wired sensor networks but have always been expensive, due to installation and maintenance costs. But the large amount of research projects in this area allows for the existence of tiny hardware (and more accurate) devices with reduced cost/size. In the recent years the WSN are used in various monitoring applications, such as:

- One of the areas on the rise is the monitoring in automobiles (Tavares, Velez, & Ferro, 2008).
- Physiological Monitoring – consists of an array of sensors embedded into the fabric of the wearer to continuously monitor the physiological parameters (Pandian et al., 2008).
- Military – Sensing intruders on bases, detection of enemy units' movements on land/sea, chemical/biological threats and offering logistics in urban warfare (Mehndiratta & Bedi, 2013).

Wireless sensor networks consist of nodes with sensing, computation, and wireless communications capabilities. Many routing, power management, and data dissemination protocols have been specifically designed for WSNs where energy awareness is an essential design issue. The IEEE standard 802.15.4 and ZigBee protocol ("XBee ZigBee Addressing", 2016), has as objectives to provide a stable and secure communication, the low energy consumption, an easy installation, reduced maintenance effort and low cost. The ZigBee protocol was developed by the ZigBee Alliance, an organization composed by several companies (including, e.g., Logitech, Intel, LG, Cisco, Sony, Samsung ("Our Members | The ZigBee Alliance", 2016). The ZigBee specifications supports inexpensive and robust networking in environments with a very large number of nodes. It should be noted that, although similar, ZigBee, Bluetooth, and WLAN (Wi-Fi) are designed for different purposes and different applications (Huang, Hsieh, & Sandnes, 2008).

	ZigBee (802.15.4)	Bluetooth (802.15.1)	Wi-Fi (802.11)
Data rate	20-250 kbps	1 Mbps	11 & 54Mbps
Range	10-100 meters	10 meters	50-100 meters
Frequency band	868 MHz (Europe) 900-928 MHz (NA), 2.4GHZ (worldwide)	2.4GHz	2.4 and 5 GHz
Transmit power	0.5, 1, or 3mW	1, 2.5 or 100mW	100mW
Nodes per Network	256+	8	Unlimited (Depending on applications))
Topology	Ad hoc, peer-to-peer, star, mesh	Ad hoc, infrastructure	Ad hoc, very small networks
Complexity (Device and application impact)	Low	High	High
Power Consumption	(Very) Low	High	Medium

Table 14 - ZigBee, Bluetooth, and Wi-Fi characteristics

These advances include the development of communication standards such as IEEE 802.15.4/ZigBee like described above. This sensor network support small power consumption and node expansion compared to other networks standards for WSN (García-hernández, Ibargüengoytia-gonzález, García-hernández, & Pérez-díaz, 2007).

ZigBee networks can have one of three different topologies. These topologies are summarized below:

Star topology fig. 1-3 is the simplest and most limited topology available. All devices connect to a single Coordinator node and all communication goes via coordinator.

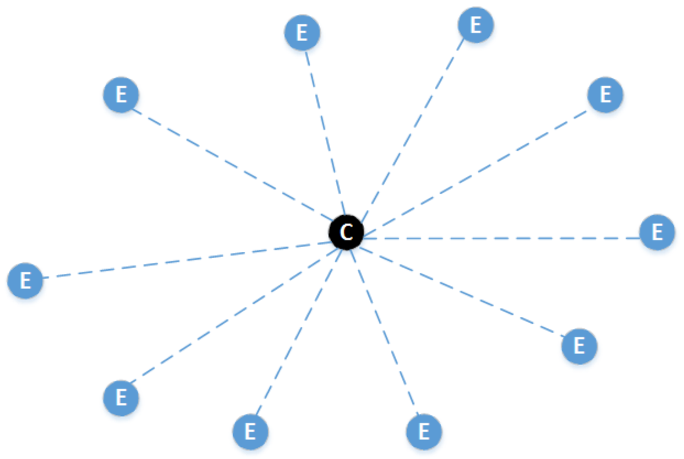


Figure 1-3 - WSN Star topology

Tree topology fig. 1-4 the coordinator forms the root node of a tree of child nodes. Direct communication can only occur between a child node and its parent, but all nodes can communicate together by traversing up the tree to a common ancestor and then down to the largest node. In this topology routers are able to extend the range of the network, but if a router fails there is no alternative route and portions of the network can become disconnected.

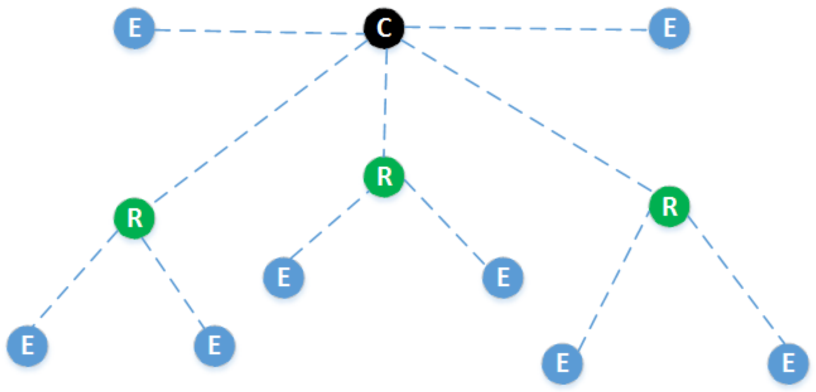


Figure 1-4 - WSN Tree topology

Mesh topology fig. 1-5 is one of the most flexible offered by ZigBee. It is similar to the tree topology but without the following the rigid tree structure and a router can communicate directly with any other router or the coordinator if it is in the range. This means that is possible to have many different routes through the network to certain node (Hillman, n.d.).

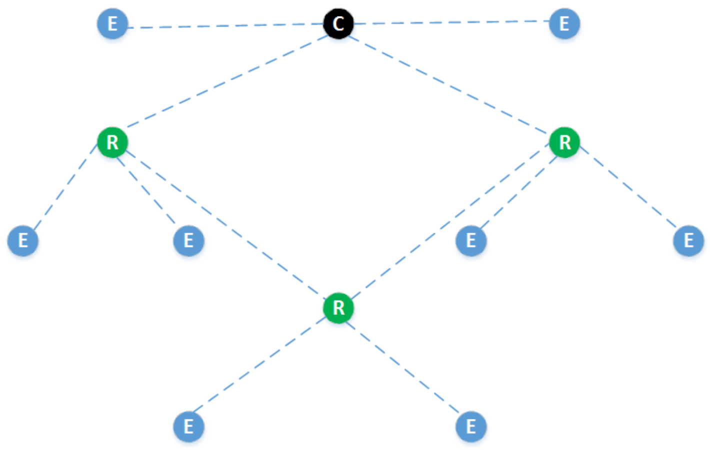


Figure 1-5 - WSN Mesh topology

1.2.10. Joining M2M communication, WSN and Cloud

This architecture allows to store and process the sensor data in a more accessible form, available timely, and cost-effective. This concept can be called "Sensor-Cloud" (Ahmed & Gregory, 2011), (Yuriyama & Kushida, 2010), that it allows easier integration with new mobile devices like tablets or smartphones through customized mobile applications that collect data from cloud and process all sensor data. The sensor-cloud infrastructure fig. 1-6 virtualizes a physical sensor as virtual sensor on the cloud computing. Dynamic grouped virtual sensors on cloud computing can be automatic provisioned when users need them. With the development of hardware limitations (in terms of size), and in pursuit of a better performance and enhancing greater computing capability, people turn to find other techniques to achieve these goals. Therefore, the concept of "cloud" was born. In fact, as early as the Internet appeared, the "cloud" has already existed silently providing for us some services (Chung, Yu, & Huang, 2013).

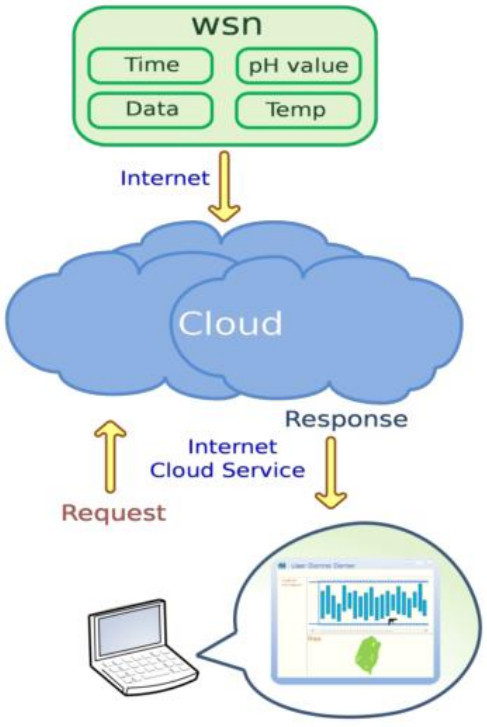


Figure 1-6 - Sensor cloud infrastructure

Like in above image the concept sensor-cloud can be seen, that is, the values/data are sent to the server and the same data are accessed remotely by clients through computers, smartphones or tablets, handling sensor data efficiently.

1.2.11. Mobile Applications

The growth of information technology and the easy access to it allowed currently several thousands of applications are released to the market every day. Mobile communication is so integrated into our lives that many people feel uncomfortable without a smartphone, a smartphone is a piece of technology multifunctional device that not only communicates, but helps to learn, earn, and have fun. This is made possible by the development of mobile applications. Some of these apps are related with the use of bicycles and cycling, but most only collect information via mobile phone modules. The difference through this work is the use WSN that can in real time detect all interactions that a biker has with his bike and through M2M communications send this data to the cloud, to be analyzed and processed in the mobile application.

According with the website Cycling Weekly (Wynn & Elton-Walters, 2016) these are the best applications that a biker could use:

Cyclemeter: This application fig. 1-7 uses the GPS functions of Apple devices to create a host of statistics to help a biker log and improve cycling performance ("Abvio | Cyclemeter", 2016). This application has the capability to record speed, time, distance and has an extensive array of workouts to follow, making it a virtual training partner.

Courtesy of Abvio app.

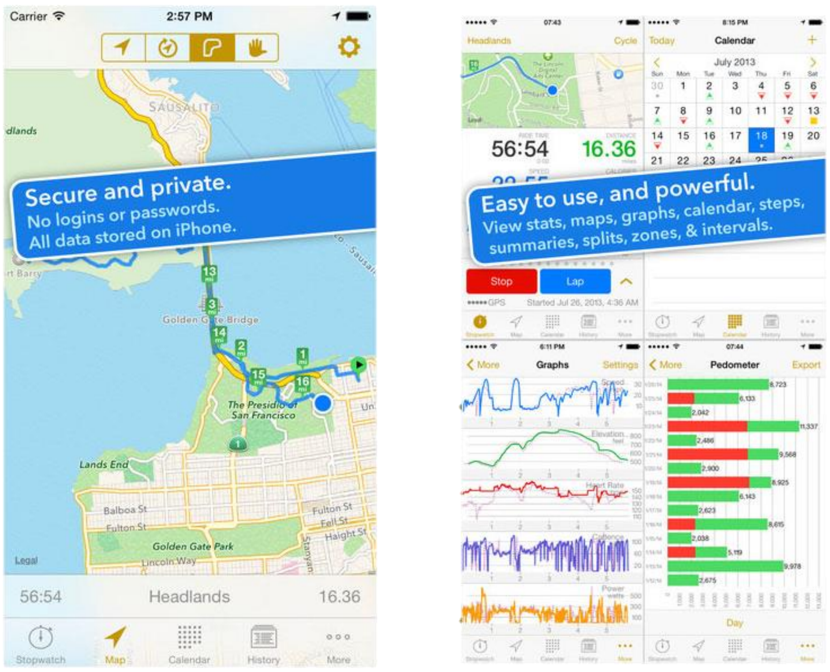
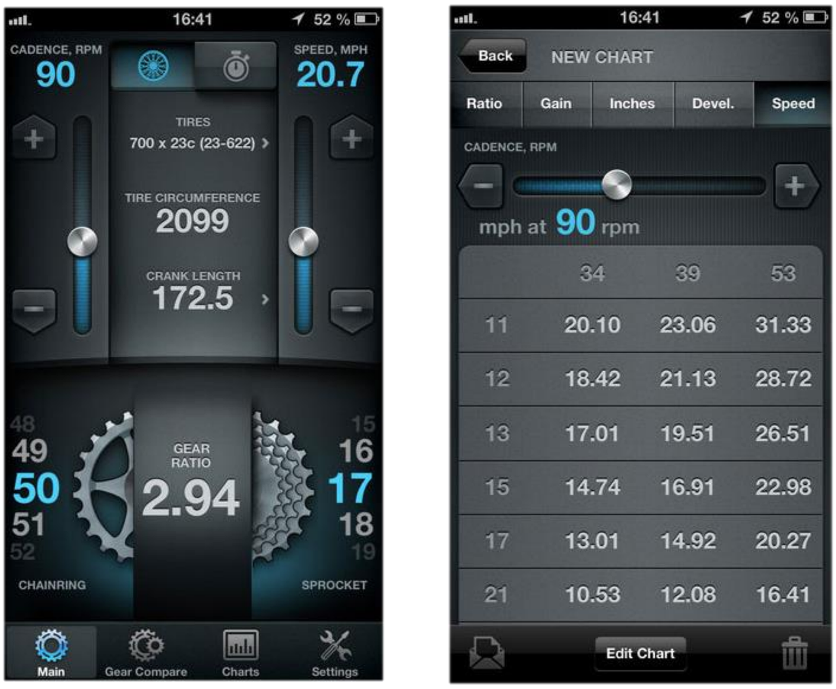


Figure 1-7 – Cyclemeter application

Bike Gear Calculator: This is a little different application fig. 1-8 in this case the application is used for every user compare gear ratios on bike to optimize his setup ("Bike Gears, Bike Gear Calculator application for iPhone and iPod Touch.", 2016).

Courtesy of Bike Gear Calculator Iphone application.



	34	39	53
11	20.10	23.06	31.33
12	18.42	21.13	28.72
13	17.01	19.51	26.51
15	14.74	16.91	22.98
17	13.01	14.92	20.27
21	10.53	12.08	16.41

Figure 1-8 - Bike Gear aplication

Strava: One of the most application used and consequently most popular is Strava fig. 1-9, this GPS cycling app offer an array of handy ride logging functions which are then uploaded to our online account profile. The app keeps track of ours ride stats as travels, including speed, time and distance. At the end of the ride the biker could view further some stats about calories burned and elevation ridden ("Strava | Run and Cycling Tracking on the Social Network for Athletes", 2016).

Courtesy of Strava, Inc.

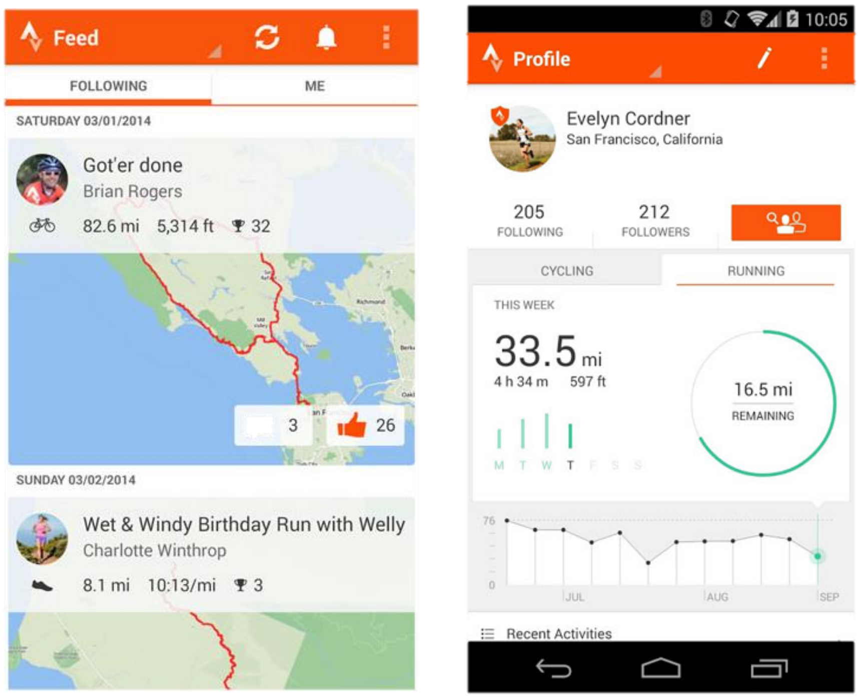
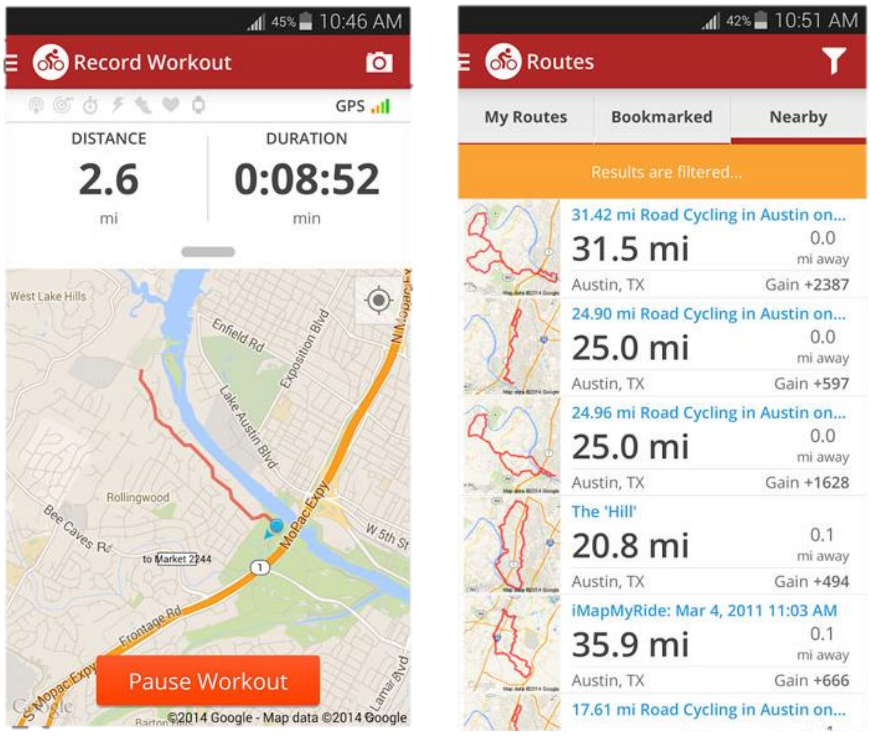


Figure 1-9 - Stava application

Map My Ride: Another GPS cycling application fig. 1-10 that records a host of data from our ride, including distance, speed, elevation and a detailed route. This app has the capability to upload all data to a site for detailed analysis and sharing with others users ("Bike Maps, Cycling Workout, Biking Routes | MapMyRide", 2016).

Courtesy of MapMyFitness, Inc.

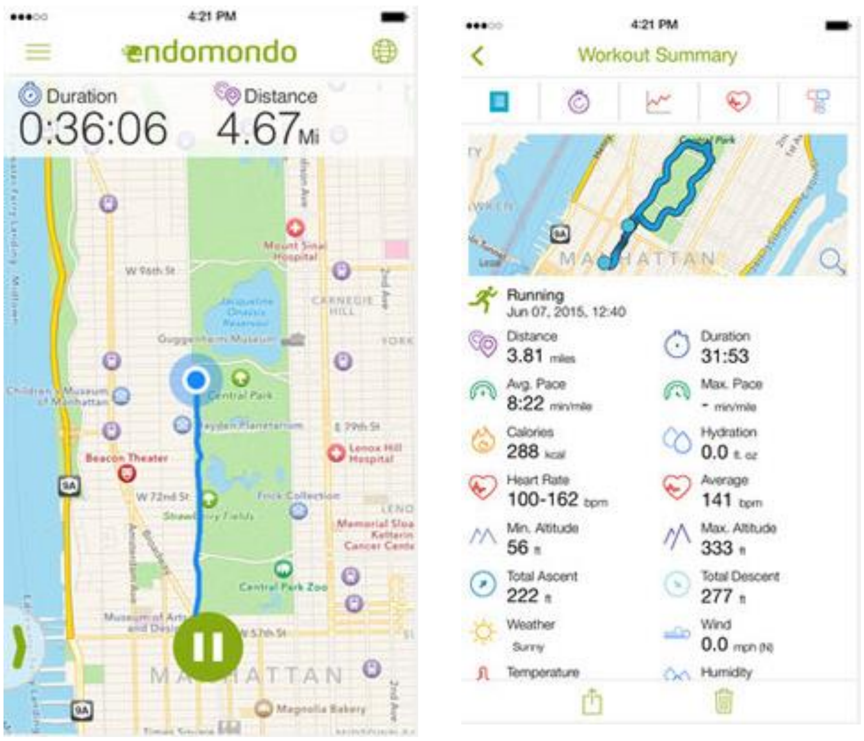


31.42 mi Road Cycling in Austin on... — 31.5 mi — 0.0 mi away — Austin, TX — Gain +2387
24.90 mi Road Cycling in Austin on... — 25.0 mi — 0.0 mi away — Austin, TX — Gain +597
24.96 mi Road Cycling in Austin on... — 25.0 mi — 0.0 mi away — Austin, TX — Gain +1628
The 'Hill' — 20.8 mi — 0.1 mi away — Austin, TX — Gain +494
iMapMyRide: Mar 4, 2011 11:03 AM — 35.9 mi — 0.1 mi away — Austin, TX — Gain +666
17.61 mi Road Cycling in Austin on...

Figure 1-10 - Map My ride application

Endomodo: Is an application fig. 1-11 designed to be a personal trainer and as such features “audio encouragement” to motivate the athletes during the exercise, rather than just tracking and logging app ("Endomondo", 2016).

Courtesy of UNDER ARMOUR CONNECTEDFITNESS.



Running
Jun 07, 2015, 12:40

Distance 3.81 miles	Duration 31:53
Avg. Pace 8:22 min/mile	Max. Pace - min/mile
Calories 288 kcal	Hydration 0.0 fl. oz
Heart Rate 100-162 bpm	Average 141 bpm
Min. Altitude 56 ft	Max. Altitude 333 ft
Total Ascent 222 ft	Total Descent 277 ft
Weather Sunny	Wind 0.0 mph (N)
Temperature	Humidity

Figure 1-11 - Endomodo application

1.2.12. Mountain Bike

There are an enormous number of sports that nowadays can practice with a bicycle. For this reason, many bicycles can be produced with some aspects that the athletes can specify to the manufactures. Cycling is practice with bicycles normally produced of fiber-carbon and with tires very thin, to improve his weight, lower rolling resistance and grip while cornering. In the case of the sports practiced on the mountain bikes the weight is put aside and the concern is more with the strength and safety of the bikers. The first thing to know is that mountain biking is a sport or recreational activity that consists of a person riding over a rough terrain, using a specially adapted mountain bike. The first mountain bike was a cruiser bicycle that was modified to enable cyclists to freewheel down mountain bike trails (STEYN, VAN NIEKERK, & JACOBS, 2014). The sport became popular in the 1970s in Marin Country, California, USA. A bicycle frame is the main part of a bicycle, onto which the wheels and all other components are attached fig. 1-12. The geometry of a mountain bike varies based on the angle of the seat post and the head tube measured from the horizontal. Mountain bike frames are manufactured using materials such as carbon steel, steel alloys, aluminium alloys, titanium and carbon fiber ("HISTORY | The Marin Museum of Bicycling and Mountain Bike Hall of Fame", 2016).

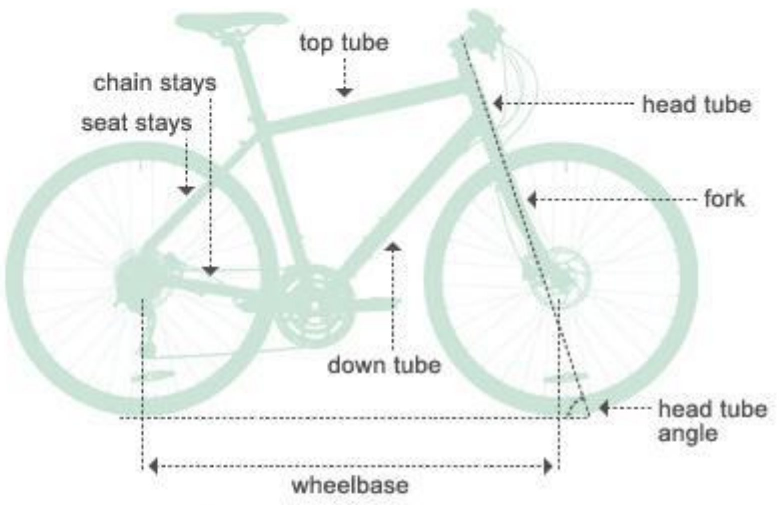


Figure 1-12 – Safety Bike struture

The invention of the Wheel is one of the most significant advances in history. Rolling resistance between the Wheel and road surface is a major factor in the performance of any vehicle. The terrain surface has a major impact on the rolling speed of a wheel and the overall performance of the vehicle (Jackson et al., 2011), (Steyn & Warnich, 2014).

1.3. Objectives

The main objective with this work is to build and develop a monitoring system associated athlete of mountain Bike, using a distributed WSN on his equipment and bicycle. This equipment should not interfere with his workout, should contribute to achieve a better performance.

The system included historically information from the athletes, with the objective to each one achieve better performance.

The following four topics describe the developed system very briefly:

1. Wireless Sensor Network: Creation of wireless sensor network that can capture the interactions of bicycle and biker. This implies the choice of sensors and his assembly.
2. Developing acquisition, processing application: On the microcontrollers is important to acquire the information from the sensors, this information needs to be locally processed and sent to the database.
3. Database: Creation and installation of mountainBikeDB, with the supposed mechanism to read and write to it.
4. Mobile Application: Developing of an Android application for tablet or smartphone that could provide the visualization of all metrics calculated using the training data obtained during practice.

1.4. Related Work

Cycling it's a complex physical activity that involves a set of movements. The cycling also provides a framework to a variety of complex problems in system dynamics and control. These include multi-body dynamics, nonlinear and linear system descriptions, human control, system simulation and instrumentation. The bicycle with a human comprises a human vehicle with dynamic behavior. In contrast with an automobile or aircraft, the pilot of a bicycle comprises 80%~90% of the overall system mass and thus the motion of the rider is not negligible. These movements can be analyzed using different techniques, such as instrumented bicycle or motion capture techniques ("Bicycle dynamics, control and handling — Sports Biomechanics Lab", 2016). These two techniques highlight the complexity of the study. The first one it's related with the use of sensors on the bike or in the biker's equipment like pressure sensors and GPS. The second technique could be related with the movement of the biker related with the bicycle, that is, analyze the influence of the position of the upper body and the legs while the bicycle is on movement. These two techniques allow the creation of models that describe the movement, the points of pressure (Lie & Sung, 2010) and speed that a biker could be doing while training on certain track. These data could be used to provide a track specific performance report.

1.5. Methods of Analyze

Like in football specially in pre-season many of the athletes of this sport use some equipment's to track is own training, this equipment is produced by the GPSports company ("Home - GPS Tracking Systems for Elite Sports", 2016), this equipment incorporates advanced GPS tracking with heart rate and accelerometer monitoring. In the case of cycling the heart-rate monitor is one piece of technology most used to improve speed, fitness and body composition in the training of bikers (cyclists, 2016). On the University of California, they are using some instrumented bikes to study the bicycle dynamics, control and handling ("Instrumented Bicycle — Sports Biomechanics Lab", 2016). They are using some sensors on the bicycle to collect information about some variables related with force and accelerations produced. This type of study allows the creation of models that describe all the movements and balance that a biker produce with his bicycle. As well the research related with braking on a bicycle (Sundström, Bäckström, Carlsson, & Tinnsten, 2015), (Beck, 2009).

Chapter 2 - System Description

2. Overview

The architecture of the system has three blocks fig. 2-1. At first block is represented the bicycle and the biker, so it is easy to assume that wireless sensor network was placed, on the sports equipment (i.e., gloves, shoes, chest trap and bicycle frame).

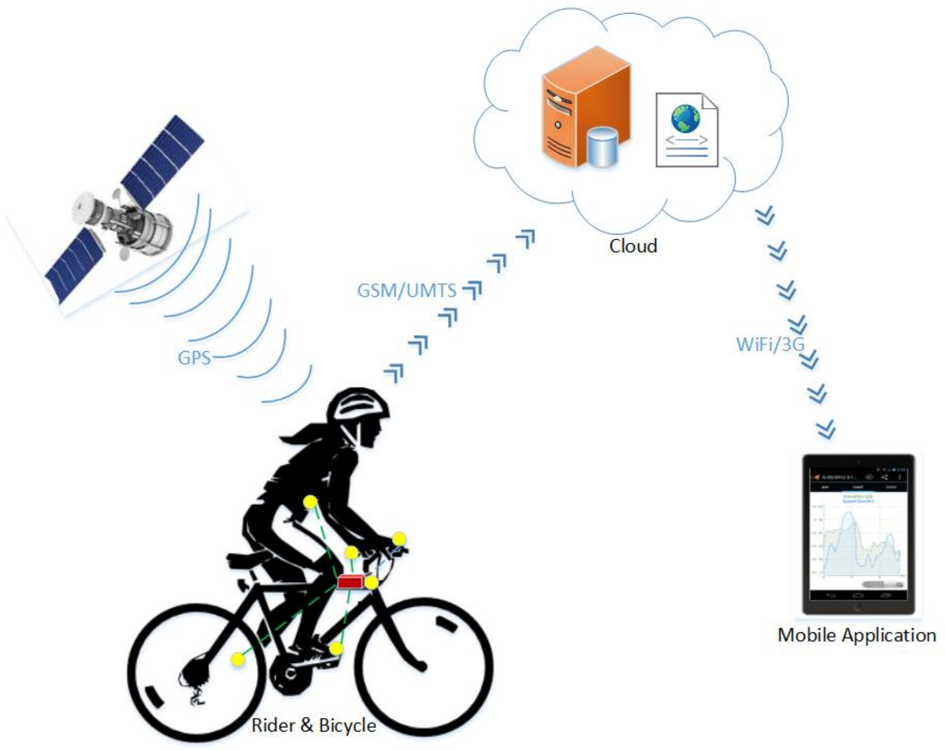


Figure 2-1 - System Architecture

Each of the end-nodes has a microcontroller ATmega328p that will read the behavior and the interaction of the biker with his bicycle. These nodes will make the acquisition and processing of primary data and then send it to coordinator ATmega2560, that stores the information on a local database and send it to the cloud. The second block is the server (cloud), which receives the information and makes it available to be accessed later. The third block, represents a mobile application that access the information in the cloud, interprets and correlates all biker information and allows data visualization with a friendly graphical user interface. In the above figure is presented some yellow dots and a parallelepiped, this is to illustrate where the end-nodes and coordinator where placed. They are based on the microcontrollers platform shown on fig. 2-2.

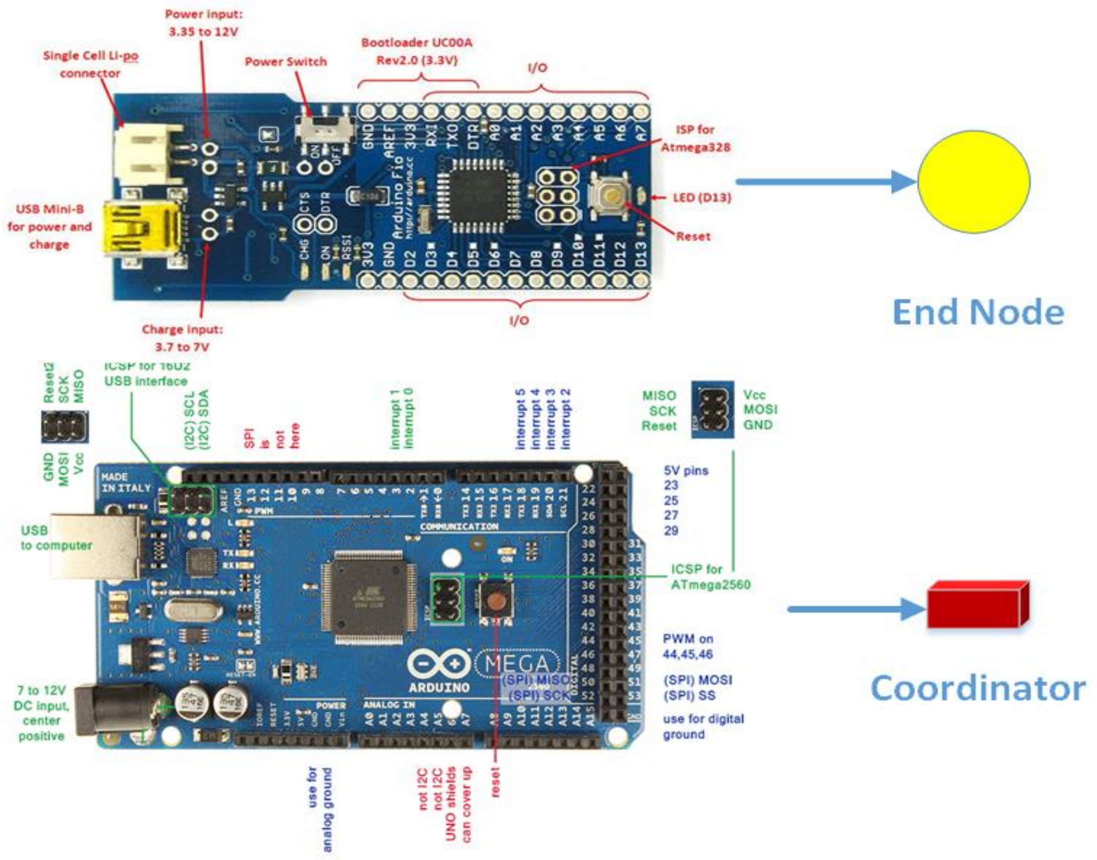


Figure 2-2 - Representation Coordinator & End-node

2.1.Hardware Components

The Hardware involved in the WSN is responsible for the acquisitions, processing and sending of data to the server. So, in the WSN are the force sensors, inertial measurement boards, shields to attach to the microcontrollers and the microcontrollers himself.

2.1.1. Force Sensors

Force Sensors from FlexiForce® ("FlexiForce A201 Sensor", 2014), act as a force sensing resistor in an electric circuit, when no pressure is applied to the sensor the resistance is very high, otherwise the resistance decrease (Manual, 2016). The force sensor is materialized by A201 FlexiForce® thin film piezoresistor fig. 2-3 included in a voltage divider implementation.

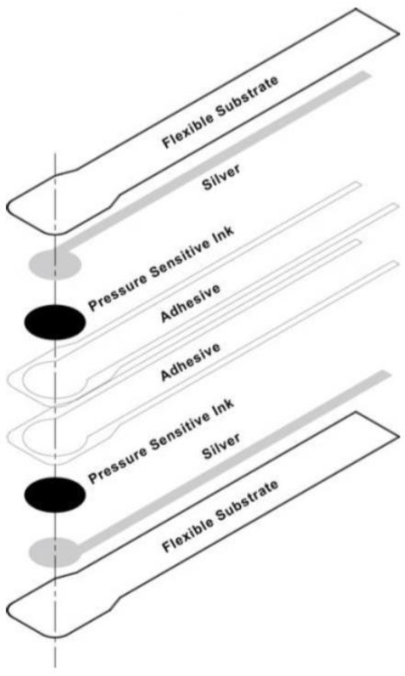


Figure 2-3 - FlexiForce a201 Composition

The FlexiForce® force sensor is an ultra-thin, flexible printed circuit. The standard A201 force sensor is constructed of two layers of substrate (polyester) film. On each layer, a conductive material (silver) is applied, followed by a layer of (Pressure-Sensitive Ink). Adhesive is then used to laminate the two layers of substrate together to form the force sensor. The active sensing area is defined by the (silver) circle on top of the (Pressure-Sensitive Ink). Silver extends from the sensing area to the connectors at the other end of the sensor, forming the conductive leads. A201 force sensors are terminated with male square pins, allowing them to be easily incorporated into a circuit. These sensors are available in three force ranges Low 4.4N, Medium 111N and High 445N. Foil sensors such as produced by FlexiForce® are also commonly used in commercial applications. One example are PlayStation controllers which have buttons that do not only have an on-off function but also a scaled input according to the applied force on the button.

Performance

- Linearity (Error): <±3%
- Repeatability: <±2.5% of Full Scale
- Hysteresis: <4.5% of Full Scale
- Drift: <5% per Logarithmic Time Scale

- Response Time: <5µsec
- Operating Temperature: (-9ºC – 60ºC)

In this work the sensor used is the Sensor HT201-L (Low: 0-30lb [133N] force range). To know the real force, applied on sensor it's needed to execute some calibration. In the fig. 2-4, the graphs show the resistance curve and the conductance curve (1/R).

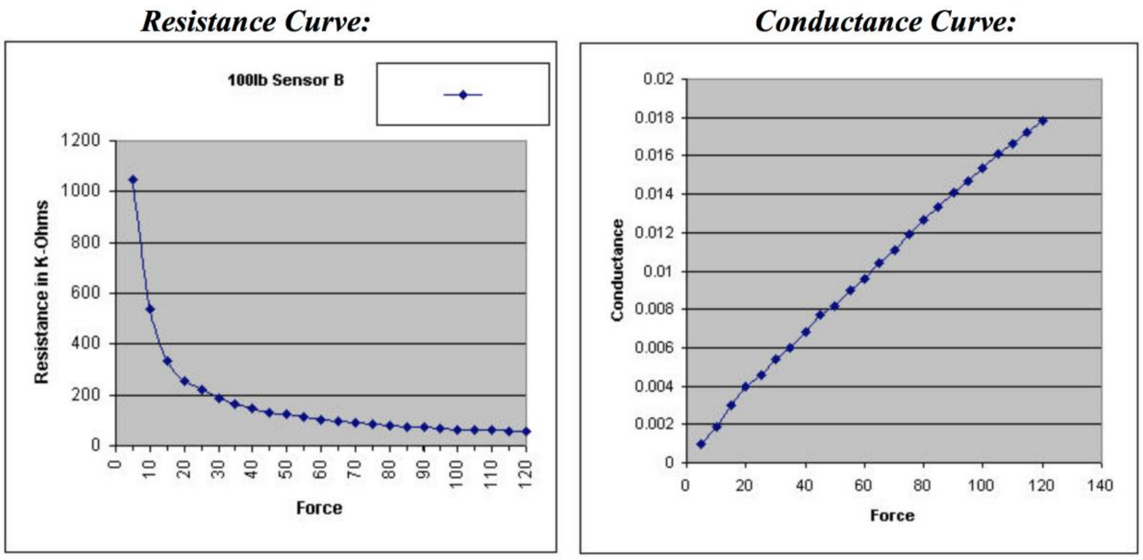


Figure 2-4 - Force versus Conductance

In the project these sensors are in the biker shoes and gloves, these ten sensors acquire the interaction described. The following conditioning circuit fig. 2-5 is used:

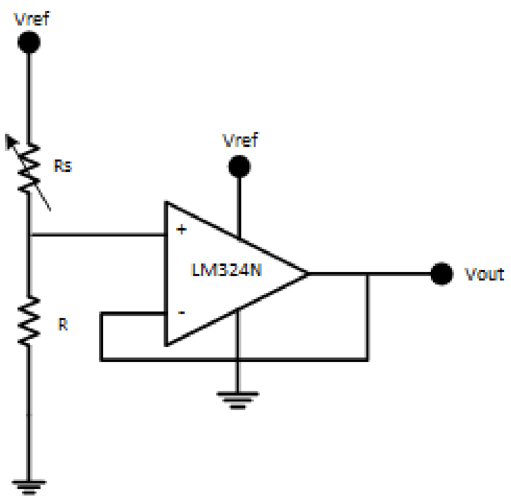


Figure 2-5 - Conditioning circuit for force sensors

The equation of the output voltage is represented as follows:

$$Vout = \frac{R}{Rs + R} \times Vref \qquad (1)$$

where the parameters in equation (1) are defined as follows:

Vout: is the output voltage [V];

Rs: is the variable resistance, force sensor [Ω];

R: is the reference resistor [Ω];

Vref: is the reference voltage [V];

This condition circuit is used because the acquisition module only reads voltage values. It's important to say that in our circuit Vref is 3.3V and the R=1MΩ.

For the conversion of the ADC values from the sensors for Newtons (N) in the side of the application is used the following formulas:

$$gain = \frac{KnowForce}{ADVValue} \approx \frac{1500g}{172} \approx 8.72 \qquad (2)$$

In order to obtain a calibration value is necessary to apply a known force (in this case 1.5Kg) and read the ADC value, after that is divided the force value by the ADC value read.

$$weight\ (in\ Newton) = gain \times (ADC_{value}) \times 0.001 \times 9.80665002864 \qquad (3)$$

Then the equation (3) is applied to obtain the force applied in each force sensor.

On fig. 2-6 is possible to observe the location if a biker applied a force of approximately 14.53N equivalent to 1482 grams on the pressure sensor.

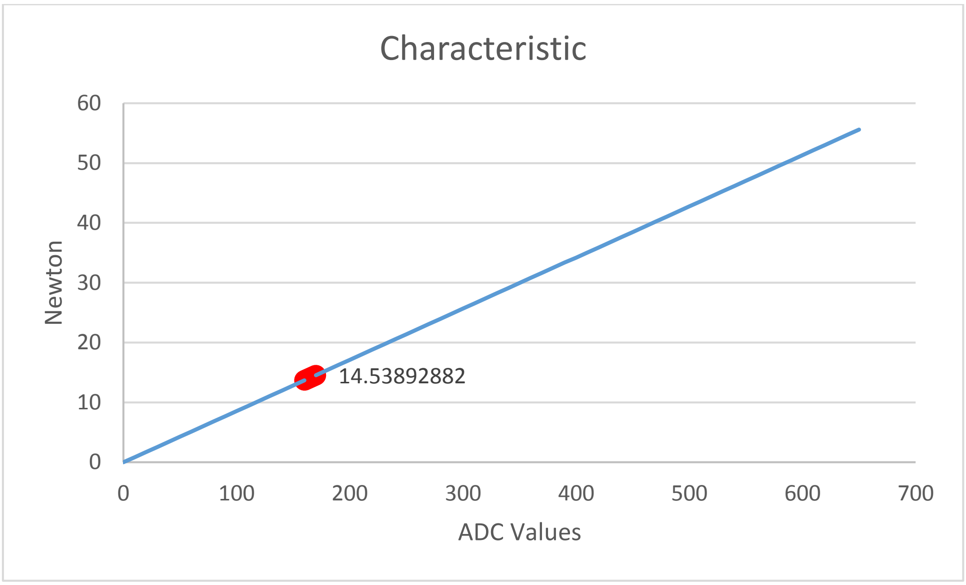


Figure 2-6 – Characteristic of the sensor

2.1.2. Inertial Measurement Unit

IMU, is an inertial measurement unit that packs an L3GD20H 3-axis gyro and an LSM303D 3-axis accelerometer and 3-axis magnetometer onto a tiny 0.8x0.5 board. This board fig. 2-7 and fig. 2-8, allows to calculate some angles between the bicycle and the rider as well as the direction of the movement ("Pololu - MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carrier)", 2016).

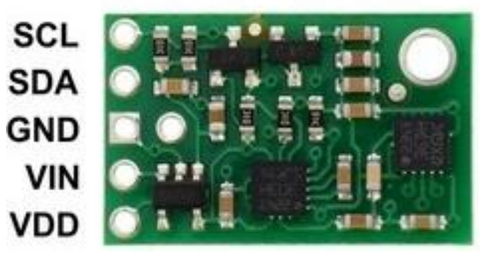


Figure 2-7 - MinIMU-9 v3 Gyro, Accelerometer, and Compass

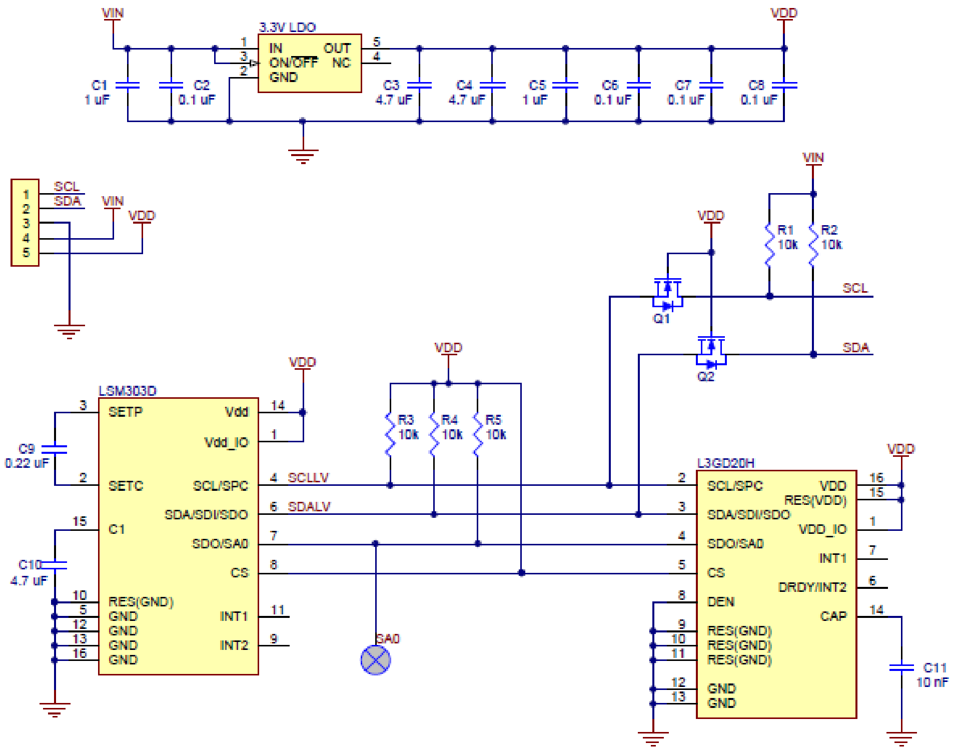


Figure 2-8 - IMU schematic

The nine independent rotation, acceleration and magnetic readings (known as 9 Degrees of Freedom) provides all the data needed to make an altitude and heading reference system (AHRS). These sensors, combined with a built in processor create an inertial sensor system fully capable to measure the altitude of objects in 3D space. The accelerometers measure proper acceleration – the rate at which the velocity of an object is changing. They measure the static (gravity) or dynamic (motion vibration) acceleration forces of a given object. The ideal accelerometer in AHRS provides a long term stability, low vibration error and reliability. Magnetometers are used in AHRS to measure the direction of the magnetic field at a point in space. In case of gyroscope the AHRS demand very precise sensors, the gyros are used as the primary source of orientation information. The quality of these devices has big impacts in overall performance of the inertial sensor system (Honglong, Liang, Wei, Guangmin, & Weizheng, 2008). This technology provides good accuracy and reliability, it is not conducive to a MEMS-based AHRS due to its larger size and greater power requirements.

This sensor is used on the chest of the rider to measure the upper body motion and in the bicycle frame to record the oscillations on the three plans of motion (Prayudi & Kim, 2012). Each of the three sensors acts as a slave device on the same I^{2}C bus.

The orientation of the bicycle/rider fig. 2-9 is often described by three consecutive rotations, whose order is important. The angular rotations are called the Euler angles. The orientation of the body frame with respect to the fixed earth frame was determined in the following manner.

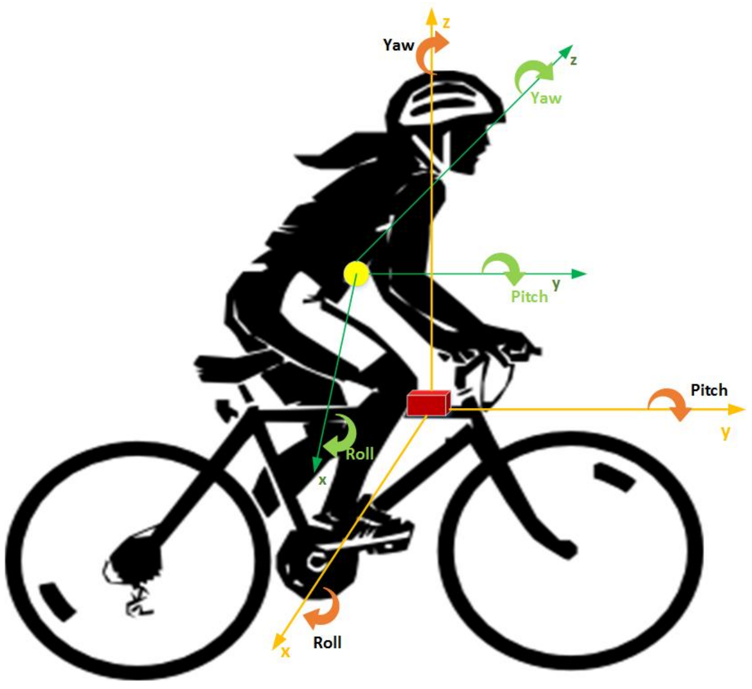


Figure 2-9 - Representation IMU axis

2.1.3. Radio Frequency Identification

Principle of radio frequency identification (RFID) is based on using of wireless non-contact radio frequency electromagnetic fields of data transmission for the purpose of automatic identification and observation of RFID tags which are situated in objects. RFID uses a passive device (called an RFID tag) to communicate data using radio frequency (RF) through electromagnetic induction (Dobrnjac, 2016).

RFID tags contain electronically stored information, which is receive on initiative of RFID readers and aerials from the tag. RFID systems differ in numerous aspects: working frequency and readings distance, type and capacity of tag memory, target and insurance of data. The tags can be classified in three type's passive, active or semi passive.

Passive RFID tag not includes own energy source and it is dependent on the power supply of the aerial sensor. The electromagnetic field serves as the energy source for RFID tag as communication channel in the line of sensor to RFID tag. The purpose of passive tags is identification of objects at which the transfer of pulse is realized directly in tag.

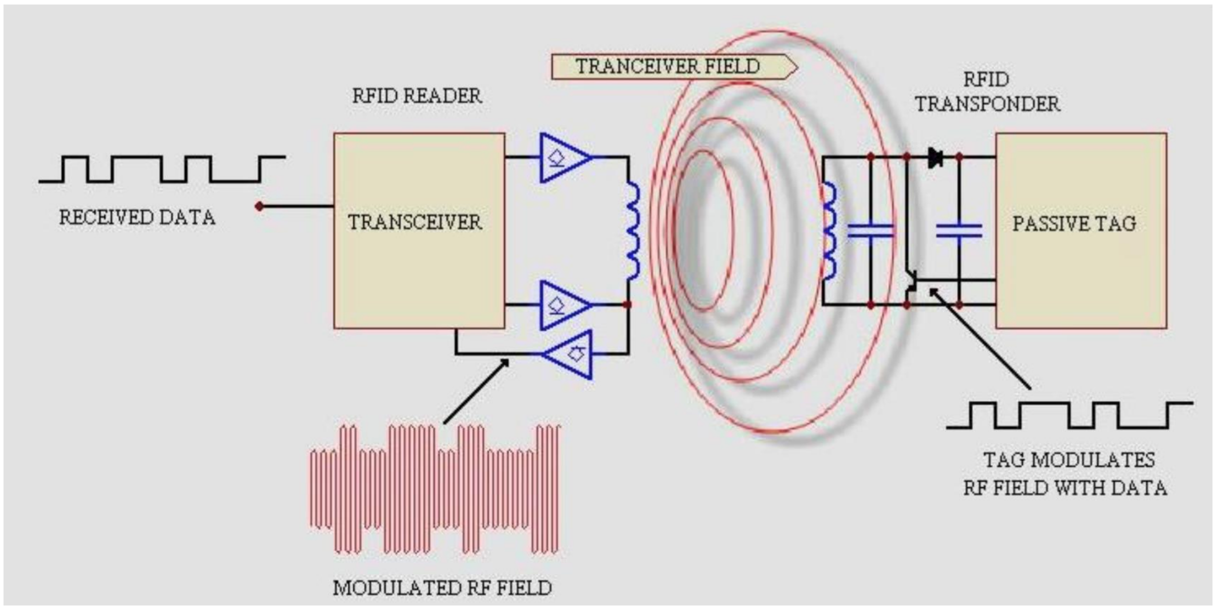
Active RFID tag not serves only for identification of objects but also for further functions as temperature measurement, pressure measurement etc. This tags can be independent on sensor and it can contain the sensors for measurement of physical quantities. Often, it is able to visually and acoustically to communicate with user. It means that it receives and emits data at the same time.

Semi Passive RFID tag use batteries and rely on the RFID reader signal to communicate. This allows them to provide a very long range of readability, making them an effective way to tracking costly items over great distances. Since Semi Passive RFID are batteries operated, they have the ability to monitor inputs from sensors as well, allowing them to control outputs which results in the ability to activate/deactivate such as alarms and thermostats ("Radio Frequency Identification Tags - Semi Passive (Battery Assisted Backscatter) - Available Technologies - PNNL", 2016).

Frequency band	Frequency range	The most common using frequencies in RFID system
Low frequency (LF)	100 kHz – 500 kHz	125 kHz, 134.2 kHz
High frequency (HF)	10 MHz – 15 MHz	13.56 MHz
Ultra high frequency (UHF)	400 MHz – 950 MHz	866 MHz – Europe; 915 MHz - USA
Microwaves (μW)	2.4 GHz – 6.8 GHz	2.45 GHz, 3.0 GHz

Table 15 - Frequency bands of RFID system

It is used the sensor MFRC522 fig. 2-10, that is a highly integrated reader/writer IC for contactless communication at 13.56MHz. The MFRC522 reader supports ISO/IEC 14443 A/MIFAR and NTAG (Semiconductors, 2016). In the development of this work we faced a problem that was how in the database the users/bikers identified is individual training and number of workouts performed. This is the solution that we get, so to biker could perform a training need to have a tag to identify in the coordinator. With this solution in database the biker will have all his trainings available and thus can compare them. This solution (using RFID) was adopted, because can simply connect the RFID sensor to the coordinator and the energy that the sensor consumes is very low. Every user that use the system need to be registered in a file on the SD card.



Transceiver **Passive Tag**

Figure 2-10 - Sensor RFID-RC522

2.1.4. GPS

In order to get a localization information of the biker and the preformed route the coordinator node includes a GPS receiver. The GPS shield fig. 2-11 and fig. 2-12 collect the altimetry and speed of the athlete and have the ability to save data in SD card, this one is useful to log the information, local DB. The GPS system operates independently of any telephonic or internet reception.

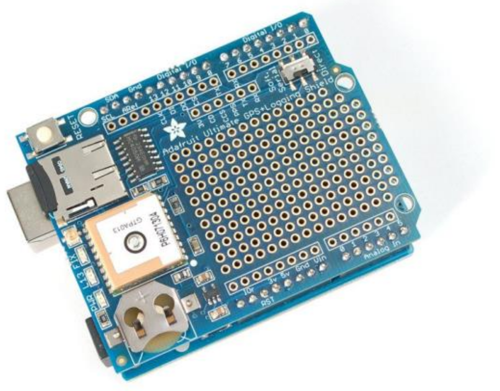


Figure 2-11 - Adafruit Ultimate GPS Logger Shield

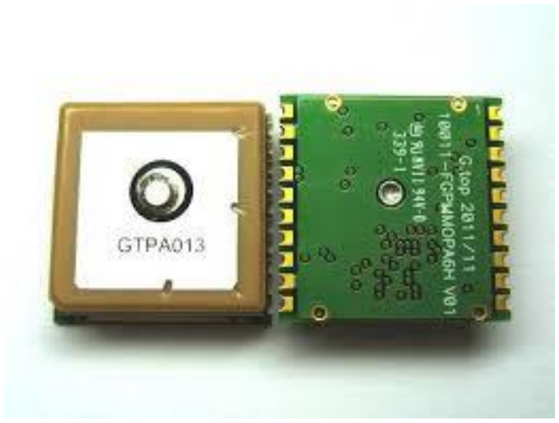


Figure 2-12 - GPS module GlobalTop PA6H

The shield fig. 2-11 uses an embedded GPS module is a GlobalTop PA6H with Mediatek MT3339 fig. 2-12 chipset that achieves the industry highest level of sensitivity (-165dBm) and instant Time-to-First Fix (TTFF) with lowest power consumption for precise GPS signal processing to give the ultra-precise positioning under low receptive, high velocity conditions (GlobalTop Technology Inc., 2011). FGPMMOPA6H is excellent low power consumption characteristics (acquisition 82mW, tracking 66mW), power sensitive devices, especially portable applications.

2.1.5. Arduino Fio

The Arduino Fio fig. 2-13 is a microcontroller board based on the ATemega328P runs at 3.3V and 8MHz. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 8 analog inputs, an on-board resonator, a reset button, and holes for mounting pin headers. It has connections for a Lithium Polymer battery. An XBee socket is available on the bottom of the board. The Arduino Fio is intended for wireless applications ("Arduino - ArduinoBoardFio", 2016).

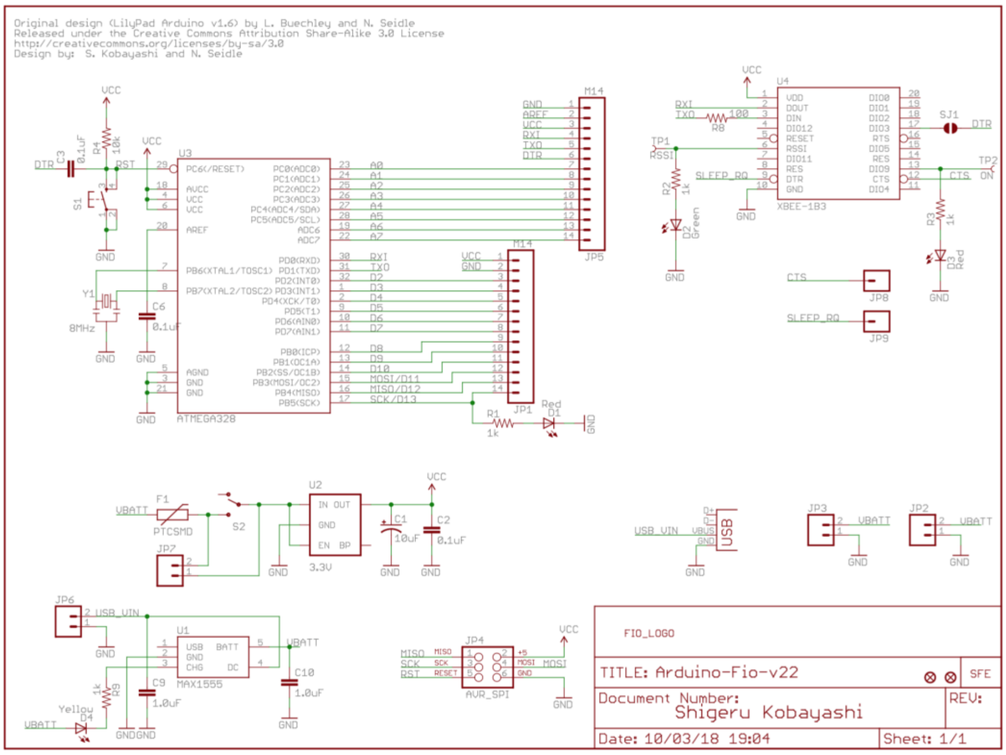

Original design (LilyPad Arduino v1.6) by L. Buechley and N. Seidle
Released under the Creative Commons Attribution Share-Alike 3.0 License
http://creativecommons.org/licenses/by-sa/3.0
Design by: S. Kobayashi and N. Seidle
ATMEGA328
XBEE-1B3
MAX1555
AVR_SPI
TITLE: Arduino-Fio-v22
Document Number:
Shigeru Kobayashi
Date: 10/03/18 19:04
Sheet: 1/1
REV:
SFE
FIO_LOGO

Figure 2-13 - Arduino Fio Schematic

On this project we develop and implement a six node wireless network based on Arduino Fio processing platform, they work as end-nodes and are responsible for the acquisition of sensor data to the network. In the following images shows the mounted equipment and used in the project (end-nodes).

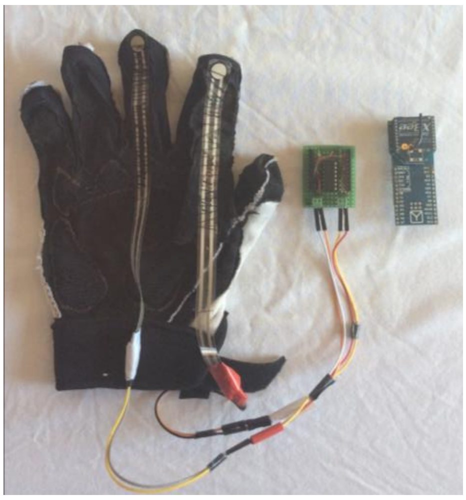


Figure 2-14 - Hand end-node

Here is presented one of the gloves fig. 2-14 of the WSN, on the left is visible the sensors from FlexiForce® stitched to the Gloves. In the middle is the condition circuit created in a board (using a Low Power Quad Operational Amplifier) (LM324-N) ("LM324-N | Operational Amplifiers | Amplifier | Description & parametrics", 2016), and two resistances. On the right side is presented the microcontroller Arduino Fio and attached to them is the XBee.

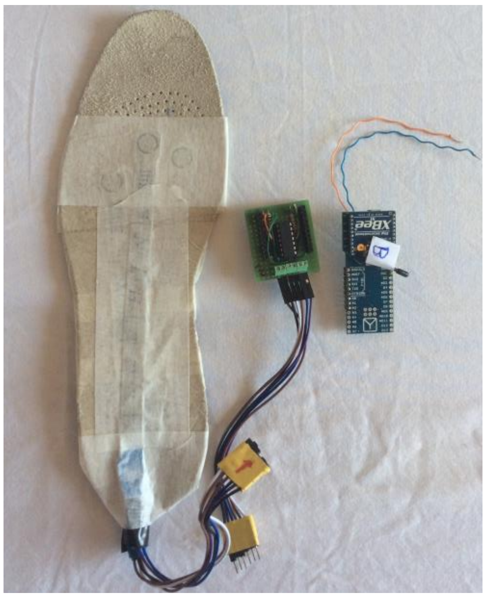


Figure 2-15 - Feet end-node

Like on gloves we have the same connections and the same circuits the main different is that we use three sensors from FlexiForce® fig. 2-15. Using one more sensor we pretend to obtain more accurate values of the force applied to the pedals of the bicycle.

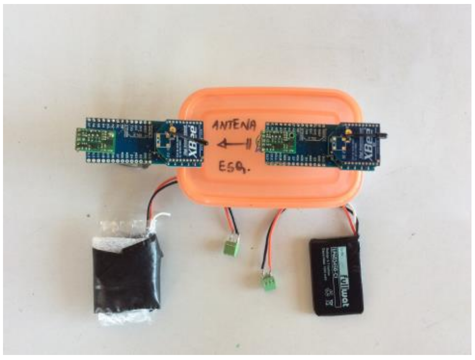


Figure 2-16 - Body and Bicycle end-node

This equipment fig. 2-16 is responsible for the measure of the variables pitch, roll and yaw. Like in the above figures the microcontroller used is the Arduino Fio and attached to them we have a XBee module and a IMU board from Pololu.

The acquisition and primary processing of sensor data is made by the microcontrollers. The microcontroller is programmed using Arduino IDE ("Arduino - Software", 2016), using a C compiler language and Arduino libraries. Depending of the type of sensors and the connections they require. On the microcontrollers the data is collected and processed to be ready to send for the coordinator.

In the case of the force sensors used in the gloves and shoes will be present in schematic way, all the connections were made to guarantee that the data is collected.

The schematic circuit represented is used to get the values from the force sensors, this specially circuit is for the gloves. The difference between this circuit and the circuit used in shoes is the insertion of a new resistor a utilization of another AMPOP and a new force sensor. Looking to the schematic fig. 2-17. Battery is a lithium polymer that powers the circuit, Sensor 1 and Sensor 2 are the FlexiForce® sensors presented above, these are connected to the conditioning circuit.

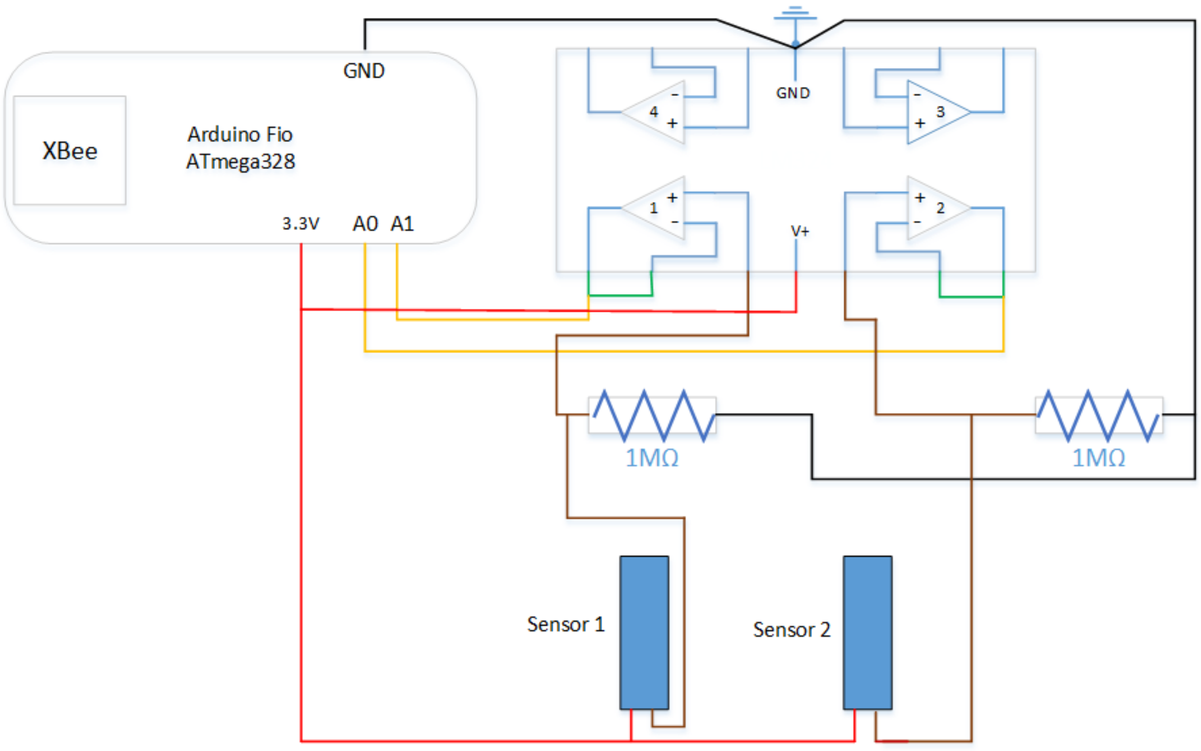
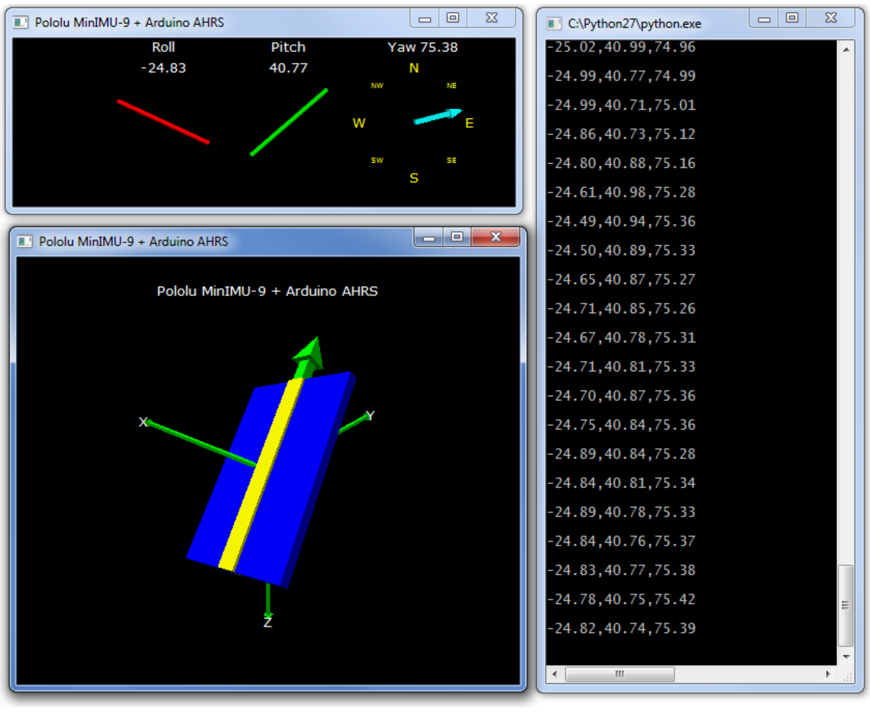


Figure 2-17 - Arduino Fio Force Sensors Gloves Schematic representation

In the case of IMU fig. 2-19, the connection is made by I^2C protocol. This communication protocol requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 slave devices. Also unlike SPI, I^2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus. Most I^{2}C devices can communicate at 100KHz or 400KHz. Communication via I^{2}C is more complex than with UART or SPI solution. The signaling must adhere to a certain protocol for the devices on the bus recognize it as valid I^{2}C communications. Fortunately, most devices take care all the fiddly details ("I2C - learn.sparkfun.com", 2016). The program that we develop was based on Arduino program provided by Pololu's fig. 2-18, this program allows the Arduino calculating estimated roll, pitch and yaw values from the sensor, making use of Arduino and Pololu's LSM303 and L3G libraries.

Courtesy of Pololu Corporation.


Pololu MinIMU-9 + Arduino AHRS
Roll
-24.83
Pitch
40.77
Yaw 75.38
N
NW
NE
W
E
SW
SE
S
Pololu MinIMU-9 + Arduino AHRS
Pololu MinIMU-9 + Arduino AHRS
X
Y
Z
C:\Python27\python.exe

```
-25.02,40.99,74.96
-24.99,40.77,74.99
-24.99,40.71,75.01
-24.86,40.73,75.12
-24.80,40.88,75.16
-24.61,40.98,75.28
-24.49,40.94,75.36
-24.50,40.89,75.33
-24.65,40.87,75.27
-24.71,40.85,75.26
-24.67,40.78,75.31
-24.71,40.81,75.33
-24.70,40.87,75.36
-24.75,40.84,75.36
-24.89,40.84,75.28
-24.84,40.81,75.34
-24.89,40.78,75.33
-24.84,40.76,75.37
-24.83,40.77,75.38
-24.78,40.75,75.42
-24.82,40.74,75.39
```

Figure 2-18 - AHRS System designed by Pololu

LSM303 library makes it simple to read the raw gyro data. In case of raw accelerometer and magnetometer data is possible to get them using the L3G.

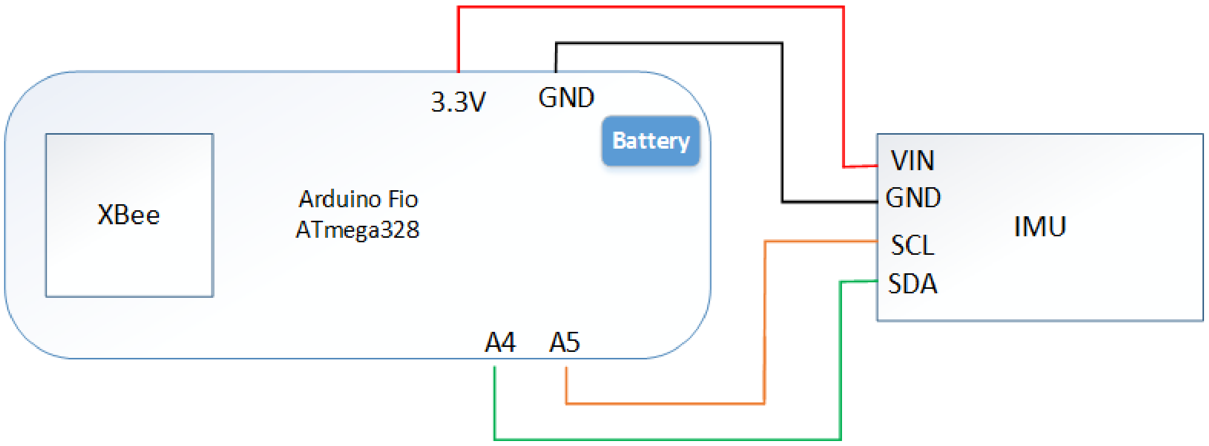


Figure 2-19 - Arduino Fio IMU Schematic representation

2.1.6. Arduino Mega 2560

The Arduino Mega fig. 2-20 is a microcontroller board based on the ATmega1280. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16MHz crystal oscillator, a USB connection, a power jack, an ICSP header is provided with a reset button in case of something goes wrong. The mega is compatible with most shields designed for the Arduino ("Arduino - ArduinoBoardMega", 2016). Arduino Mega is theoretically equivalent to packed four Arduinos Uno’s into one board. The Mega supports a whopping 256 kB of flash program space. He will receive all the information from another end-nodes.

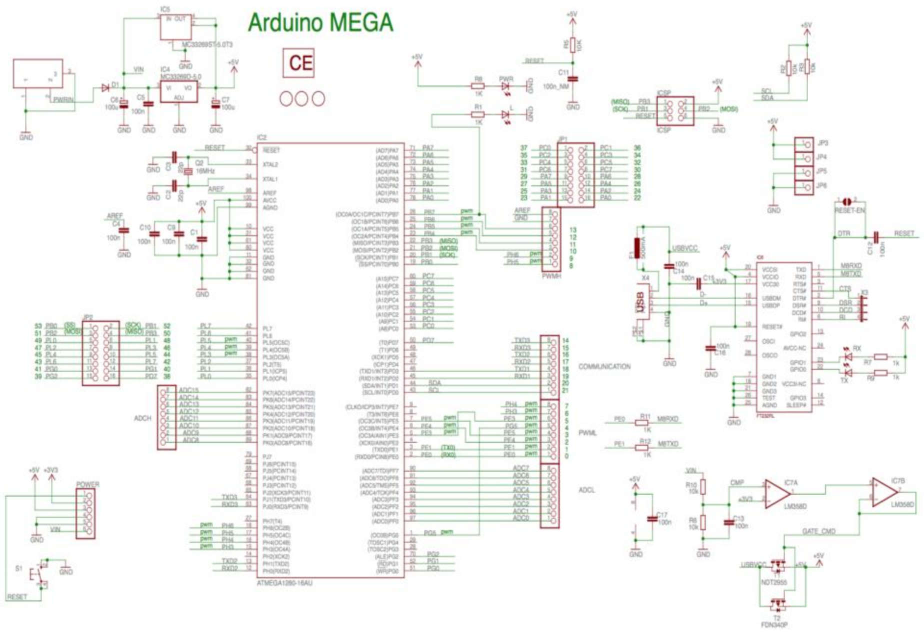


Figure 2-20 - Arduino Mega Schematic

He works as coordinator and could find it on the frame of the bicycle. On the fig. 2-21 it can be seen that is described the connection of three shields and one sensor.

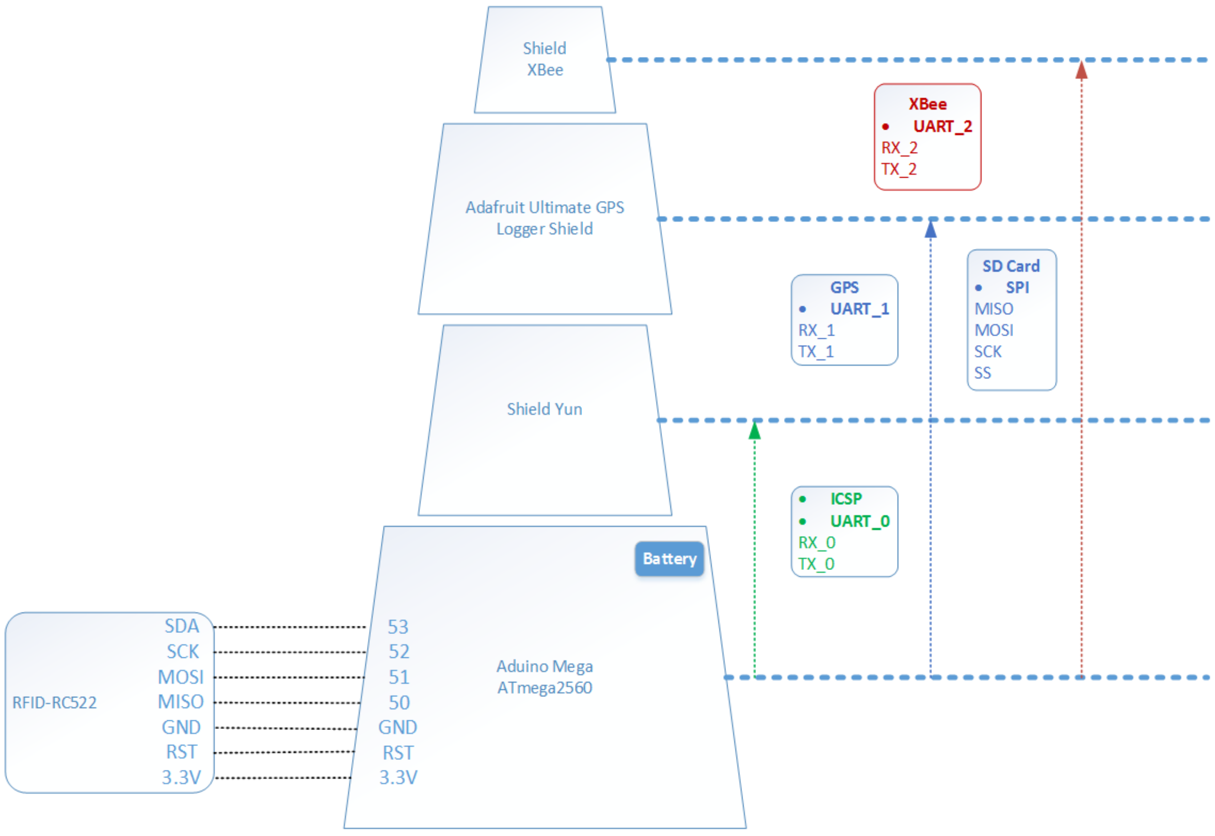


Figure 2-21 - Coordinator schematic representation

Starting from the bottom of the scheme platform based on ATMEGA2560 model we have the microcontroller, attachment to them the RFID sensor that permits to each user identify itself regarding in the coordinator. In the case of the first shield, shield YUN fig. 2-22 is one of the most important, it is responsible with the communication via 3G/4G with the remote database, this is, the data is processed and when it's ready is sent to database through a wireless connection. Yun Shield runs Open Source OpenWrt system (Same system as runs in Arduino Yun) and is fully compatible with Arduino IDE ("Yun Shield", 2016).

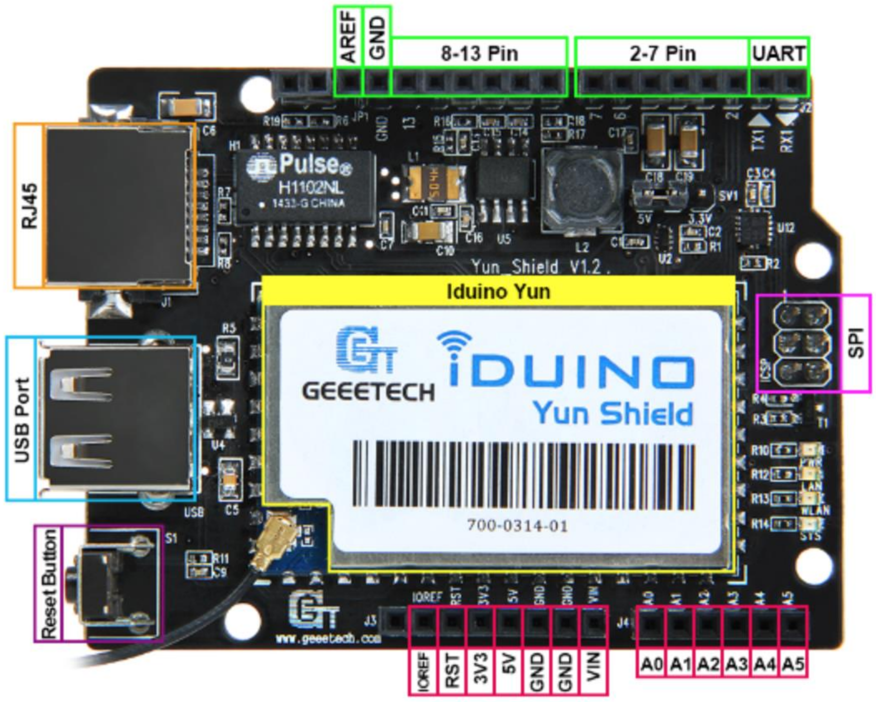


Figure 2-22 - Yun Shield

One of the problems of this demanding work was the sending of the collected data form the sensors to the cloud, using this shield the communication is possible. This shield uses two protocols of communication with the microcontroller ICSP and UART.

The shield Yun and Arduino Mega (together) is similar to the Leonardo ("Arduino - ArduinoBoardLeonardo", 2016) in that the ATmega16u2 has built-in USB communication, eliminating the need for a secondary processor. This allows the shield Yun to appear connected to a computer as mouse and keyboard, in addition to a virtual (CDC) serial / COM port.

In the Mega2560, the UART between mega2560 and mega16u2 will influence the Bridge feature with the Iduino Yun Shield. So we have to disconnect it by setting mega16u2 into reset mode. Made this little intervention on Arduino mega, the utilization of the header In-Circuit Serial Programming (ICSP) is fundamental because it will possible to develop the applications through a Wi-Fi connection. Eliminating the need of the USB cable connected to the microcontroller, for the insertion of new sketch's.

A universal asynchronous receiver/transmitter (UART) is a block of circuit responsible for implementing serial communication. In our case the Arduino Mega – built on an ATmega2560 has a whopping four UARTs ("Serial Communication - learn.sparkfun.com", 2016).

Moving for the next shield we have the Adafruit Ultimate GPS Logger Shield, in this one the communication protocols used is SPI and UART.

In case of SD card the SPI protocol is needed, the main difference between I^2C and SPI is that SPI is a synchronous solution. Because that it uses separate lines for data and "clock" that keeps both sides in perfect sync. The clock is an oscilatory signal that tells the receiver exactly when the sample the bits on the data line.

In SPI fig. 2-23, only one side generates the clock signal (usually CLK or SCK for Serial Clock). The side that generates the clock is called the "master", and the other side is called the "slaved". There is always only one master (which is almost always the microcontroller), but there can be multiple slaves.

When data is sent from the master to a slave, it's send on a data line called MOSI, for "Master Out / Slave In". If the slave needs to send a response back to the master, the master will continue to generate a prearranged number of clock cycles, and the slave will put the data onto a third data line called MISO, for "Master In / Slave Out". "Prearranged" because the master always generates the clock signal, it must know in advance when a slave needs to return data how much data will be returned. SPI is a "full duplex" (has separate send and receive lines, and the information could be transmitted simultaneously in both directions) (Mohammadi, 2016).

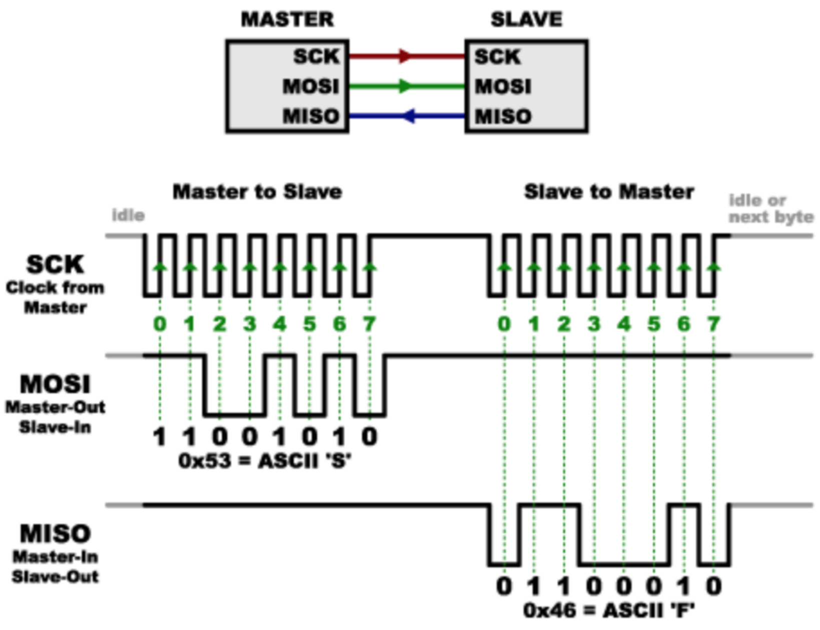


Figure 2-23 - Representation of SPI protocol

For the GPS the protocol used is UART, in this case UART_1 (TX_1 and RX_1).

The last but not the least, it’s the shield for the XBee that use UART_2 (TX_2 and RX_2), and is responsible for the reception of all data from the WSN. Because the microcontroller works as a coordinator

Chapter 3 - Embedded Software

3. Arduino Fio IMU Sensor

It was developed embedded software for microcontrollers, which is responsible of the acquisition, and processing of the data delivered by the sensors.

All the microcontrollers are programmed using C language. Several Arduino libraries were used to deal with shields and the sensor boards.

In our AHRS system is used the Direct Cosine Matrix Algorithm (DCM), it calculates the orientation of a rigid body in respect to the rotation of the earth by using rotation matrices. Rotation matrices have the advantage of being a natural fit to control and navigation. A rotation matrix describes the orientation of one coordinate system with respect to another.

$$R = \begin{array}{cccc} xb & yb & zb & \\ r_{xx} & r_{xy} & r_{xz} & xe \\ r_{yx} & r_{yy} & r_{yz} & ye \\ r_{zx} & r_{zy} & r_{zz} & ze \end{array} \qquad R^{-1} = \begin{array}{cccc} xe & ye & ze & \\ r_{xx} & r_{yx} & r_{zx} & xb \\ r_{xy} & r_{yy} & r_{zy} & yb \\ r_{xz} & r_{yz} & r_{zz} & zb \end{array}$$

The columns of the matrix are the unit vectors in one system as seen in another system. A vector in one system can be transformed into the other system by multiplying it by the rotation matrix. The transformation in the reverse direction is accomplished with the inverse of the rotation matrix, which turns out to be equal to its transpose. Certain types of vectors (directions, velocity, acceleration and translation) can be transformed between rotated reference frames with 3x3 matrix (Lima & Torres, 2012).

A_p= a vector A measured in the frame of reference of the plane;

A_G= a vector A measured in the frame of reference of the ground;

$$R = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} = rotation\,matrix \tag{4}$$

$$A_G = RA_p$$

The rotation matrices are related to the Euler angles, which describe the three consecutive rotations needed to describe the orientation. So the relation between the direction cosine matrix and Euler angles is:

(5)

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} X'' \\ Y'' \\ Z'' \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X''' \\ Y''' \\ Z''' \end{bmatrix} \tag{7}$$

The coordination system shown in fig. 2-9 is typically used in flight simulators with the origin located at the aircraft centroid, with y-axis pointing forward, the x-axis off the right and the z-axis pointing up. Note that orders matters (5)->(6)->(7). Combine all three transformations (roll, pitch, and yaw).

$$R = \begin{bmatrix} cos\theta cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ cos\theta sin\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & cos\psi sin\theta sin\phi - sin\psi cos\phi \\ -sin\theta & sin\psi cos\theta & cos\psi cos\theta \end{bmatrix} \tag{8}$$

Equation (4) and (8) express how to rotate a vector measured in the frame of reference of the plane to the frame of reference of the ground fig. 2-10.

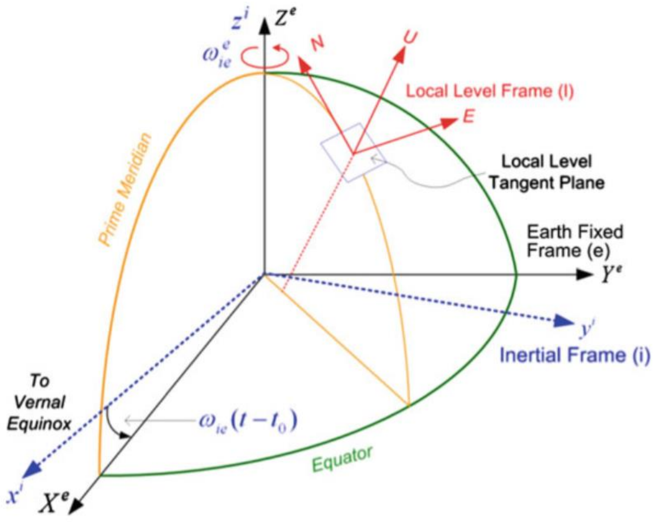


Figure 3-1 Vector measured in the frame of reference of the plane to the frame of reference of the ground

Equation (2) is expressed in terms of direction cosines. Equation (8) is expressed in terms of Euler angles (Zhao Lin, Xia, Liu, & Cheng, 2007). The following equations permit to calculate the Pitch, Roll and Yaw (Ran & Cheng, 2016).

$$Pitch \rightarrow \psi = -\operatorname{asin}(r_{zx}) \tag{9}$$

$$Roll \rightarrow \theta = atan2(r_{zy}, r_{zz})$$

$$Yaw \rightarrow \phi = atan2(r_{yx}, r_{xx})$$

```
void Euler_angles(void){    //Equation (9)
  pitch = -asin(DCM_Matrix[2][0]);
  roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
  yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}
```

DCM algorithm is represented on fig. 3-2.

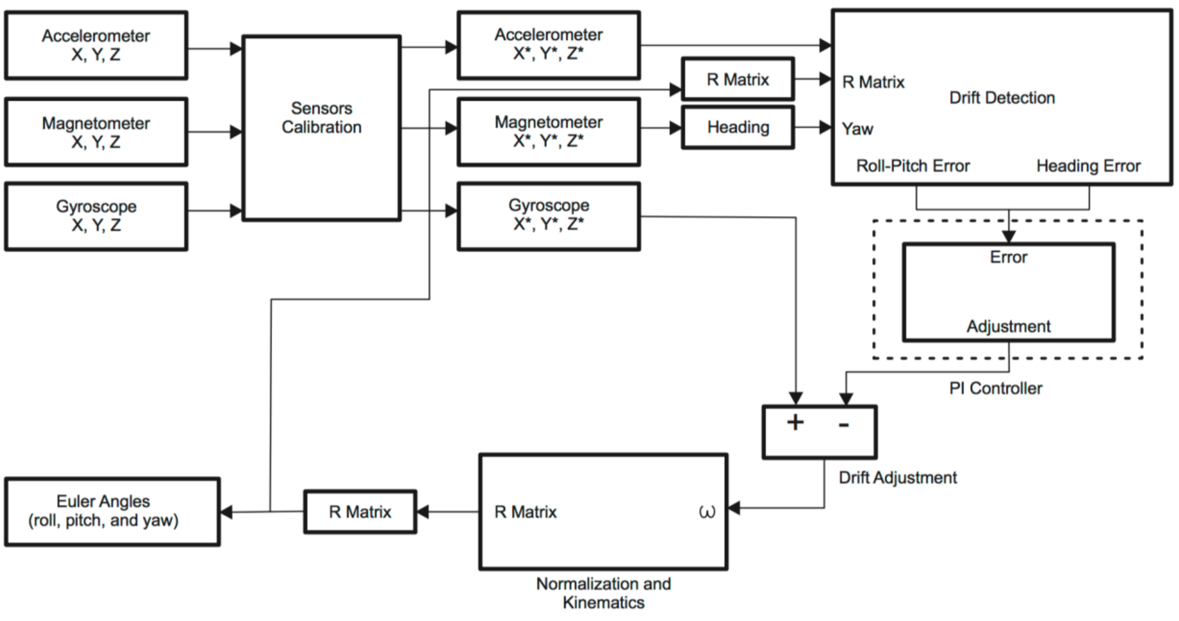


Figure 3-2 - Direct Cosine Matrix Algorithm Overview

Gyroscope readings have different offsets depending on which direction the gyroscope is facing, when these readings are integrated over time it causes the integral result to drift. Before apply equation (9), it is essential to correct this problem (drift), and the process of enforcing the orthogonally conditions "renormalization" is used. First compute the dot product of X and Y rows of the matrix, which is supposed to be zero, so the result is a measure of how much the X and Y rows are rotating toward each other.

$$X = \begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} \quad Y = \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} \tag{10}$$

$$error = X.Y = X^T Y = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \end{bmatrix} \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix}$$

We apportion half of the error each to the X and Y rows, and approximately rotate the X and Y rows in the opposite direction by cross coupling.

$$X = \begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} = X_{orthogonal} = X - \frac{error}{2} Y$$

(11)

$$Y = \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} = Y_{orthogonal} = Y - \frac{error}{2} X$$

Orthogonal error is greatly reduced by substituting equation (11) into (10), keeping in mind that the magnitude of each row and column of the R matrix is approximately equal to one. The next step is to adjust the Z row of the matrix to be orthogonal to X and Y row. The way to do that is to simply set the Z row to be the cross product of the X and Y rows. (12)

$$\begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} = Z_{orthogonal} = X_{orthogonal} \times Y_{orthogonal}$$

The last step in the renormalization process is to scale the rows of the R matrix to assure that each has a magnitude equal to one. The resulting magnitude adjustment equations for the row vectors are:

$$X_{normalized} = \frac{1}{2}(3 - X_{orthogonal} . X_{orthogonal}) X_{orthogonal}$$

$$Y_{normalized} = \frac{1}{2}(3 - Y_{orthogonal} . Y_{orthogonal}) Y_{orthogonal}$$

$$Z_{normalized} = \frac{1}{2}(3 - Z_{orthogonal} . Z_{orthogonal}) Z_{orthogonal}$$

(13)

```
void Normalize(void){
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5; //eq.11

  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq.11
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq.11

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//eq.11
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//eq.11

  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x b //eq.12

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //eq.13
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //eq.13
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //eq.13
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}
```

These equations state that to adjust the magnitude of each row vector to one, is necessary to subtract the dot product of the vector with itself (the square of the magnitude), subtract from three, multiply by ½ , and multiply each element of the vector by the result.

The accelerometer is not affected by drift, therefore, it can be used as an orientation reference in X and Y axis of the rigid body to compensate the roll-pitch error (gyro’s offset error). The magnetometer’s readings are used to calculate the heading of the rigid body, this one helps to compensate yaw error. The heading of the system used as the reference vector in the Z axis (yaw error), in addition to roll-pitch error calculated by the accelerometer, it allows the system to calculate the rotations correction matrix.

3.1.Data Communications

In order to communicate in the WSN (between end-nodes and coordinator) fig. 3-3 is used the ZigBee protocol, while the communication between coordinator and the database is made using Wi-Fi. The server connection is via the Ethernet protocol.

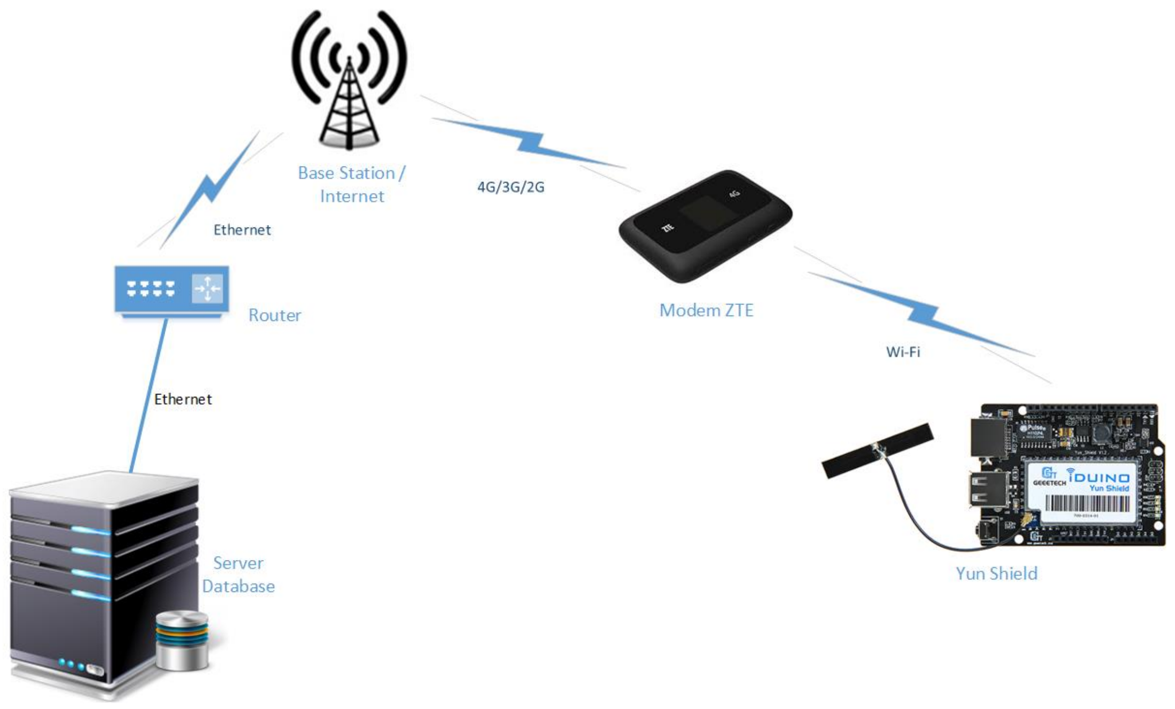


Figure 3-3 - Representation of communication between coordinator and server

3.2. ZigBee

The coordinator and every end-node have a simple XBee radio, this piece of electronic is based on the IEEE 802.15.4 standard designed for point-to-point and star communications. ZigBee over 802.15.4, defines specifications for low-rate Wireless Personal Area Network (WPAN) for supporting simple devices that consume minimal power and typically operate in the personal space of 10m. ZigBee provides a self-organized, multi-hop, and reliable mesh networking with long battery lifetime (Lee, Su, & Shen, 2007). In these networks it’s possible to have two types of devices a full-function device (FFD) or reduce-function device (RFD). A node can operate as either a FFD or RFD. An FFD can perform all the tasks that are defined by the ZigBee standard, and it operates in the full set of the IEEE 802.15.4 MAC layer. An RFD performs only a limited number of tasks (Elahi & Gschwender, 2010).

Coordinator: A coordinator is an FFD and responsible for overall network management. Each network has exactly one coordinator.

End device: An end device can be an RFD. An RFD operates within a limited set of IEEE 802.15.4 MAC layer, enabling it to consume less power. The end device (child) can be connected to a router or coordinator (parent). It also operates at low duty cycle power, meaning it consumes power only while transmitting information ("ZigBee", 2016). The end device performs the following functions:

- Joins or leaves a network;
- Transfers application packets;

In the project is used the star topology fig. 3-4 and fig. 3-5, in which the coordinator (center node) is placed in bicycle frame.

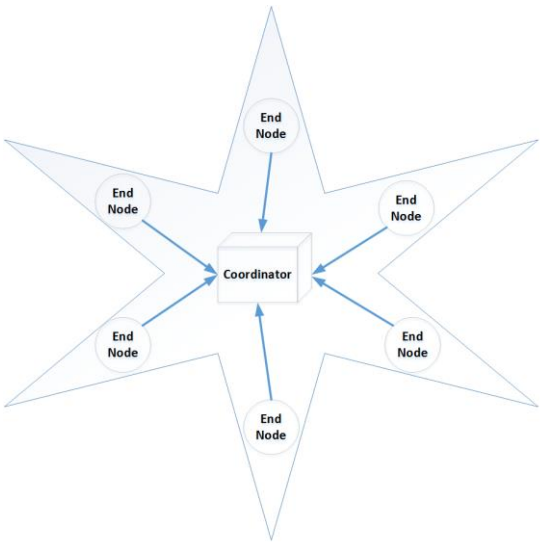


Figure 3-4 - WSN topology used

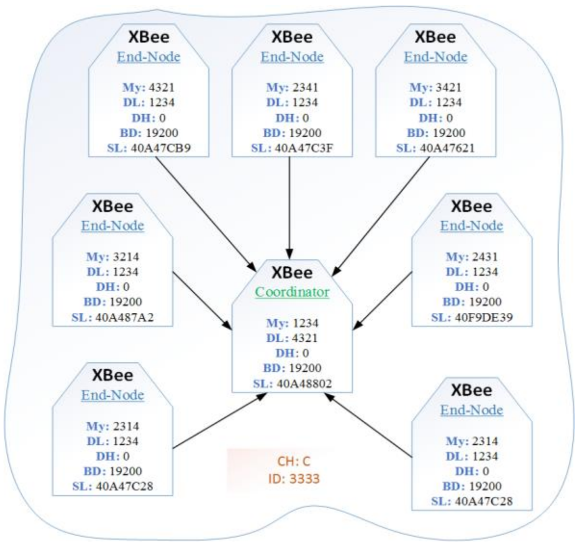

XBee End-Node My: 4321 DL: 1234 DH: 0 BD: 19200 SL: 40A47CB9
XBee End-Node My: 2341 DL: 1234 DH: 0 BD: 19200 SL: 40A47C3F
XBee End-Node My: 3421 DL: 1234 DH: 0 BD: 19200 SL: 40A47621
XBee End-Node My: 3214 DL: 1234 DH: 0 BD: 19200 SL: 40A487A2
XBee Coordinator My: 1234 DL: 4321 DH: 0 BD: 19200 SL: 40A48802
XBee End-Node My: 2431 DL: 1234 DH: 0 BD: 19200 SL: 40F9DE39
XBee End-Node My: 2314 DL: 1234 DH: 0 BD: 19200 SL: 40A47C28
CH: C
ID: 3333
XBee End-Node My: 2314 DL: 1234 DH: 0 BD: 19200 SL: 40A47C28

Figure 3-5 –Principal configurations

The main configurations made on each XBee module are: PAN ID to identify the network, the Destination Address (High and Low) to distinguish the network nodes and the baud rate. These configurations they are setup through the XCTU Software ("XCTU - Next Gen Configuration Platform for XBee/RF Solutions - Digi International", 2016). The star topology was adopted for the reason that nodes are closely to the coordinator and the end-nodes only transmit the data from the sensors.

3.3.Communication M2M

Like above is explain this protocol allows the connection without human interaction. The coordinator node is the gateway of WSN, the connection with the server in the cloud is done through a mobile internet modem. The modem used is a ZTE MF910 4G LTE Mobile Router is a new 4G Pocket Wi-Fi Hotspot which connects up to 10 Wi-Fi enabled devices to 4G Network. With 4G download speeds up to 150Mbps. Supports five LTE bands and tri band UMTS (3G UMTS and 2G GPRS/GSM network). MF910 ZTE could support 4G LTE Band 3/7/8/20, and this LTE category 4 (Cat4) mobile Wi-Fi Gateway with a built-in-battery that lasts for up to 8 hours use. This router is used for wireless Internet access using LTE, UMTS or GSM mobile telephony networks.

3.4. Wireless

Wireless fidelity (Wi-Fi) include IEEE 802.11 a/b/g standards for wireless local are networks (WLAN). It allows users to surf the internet at broadband speeds when connected to an access point (AP). The IEEE 802.11 architecture consists of several components that interact to provide a wireless LAN that supports station mobility transparently to upper layers. An IEEE 802.11 network can operate over one of the 14 channels defined for the 2.4GHz. Each channel is 22 MHz wide, and, since the overall ISM bandwidth is just above 80 MHz, the channels are partially overlapped. The modulation scheme is either a DSSS (direct sequence spread spectrum) for the lower bit rates, or an OFDM (orthogonal frequency division multiplexing) for the higher ones. This protocol is used by the Yun shield, and permits that information arrive to the database.

3.5. Ethernet

The term Ethernet refers to the family of LAN products covered by the IEEE 802.3 standard that defines what is commonly known as the CSMA/CD protocol. The connection to the server is made by twisted-pair cable 100MbpsFast Ethernet.

Chapter 4 - Cloud

4. Server

The Server fig. 4-1 in this work is responsible for the storage of every training performed by each biker. There is the place were the data is all saved for further analysis through a mobile application, this is, the application access to the webserver through PHP modules and present the data in the application with a user friendly view. The main objective is to centralize all the information in one place. This could be a problem if something went wrong with the server, but some mechanisms were created, like a local database and the possible in the mobile application to export each training.

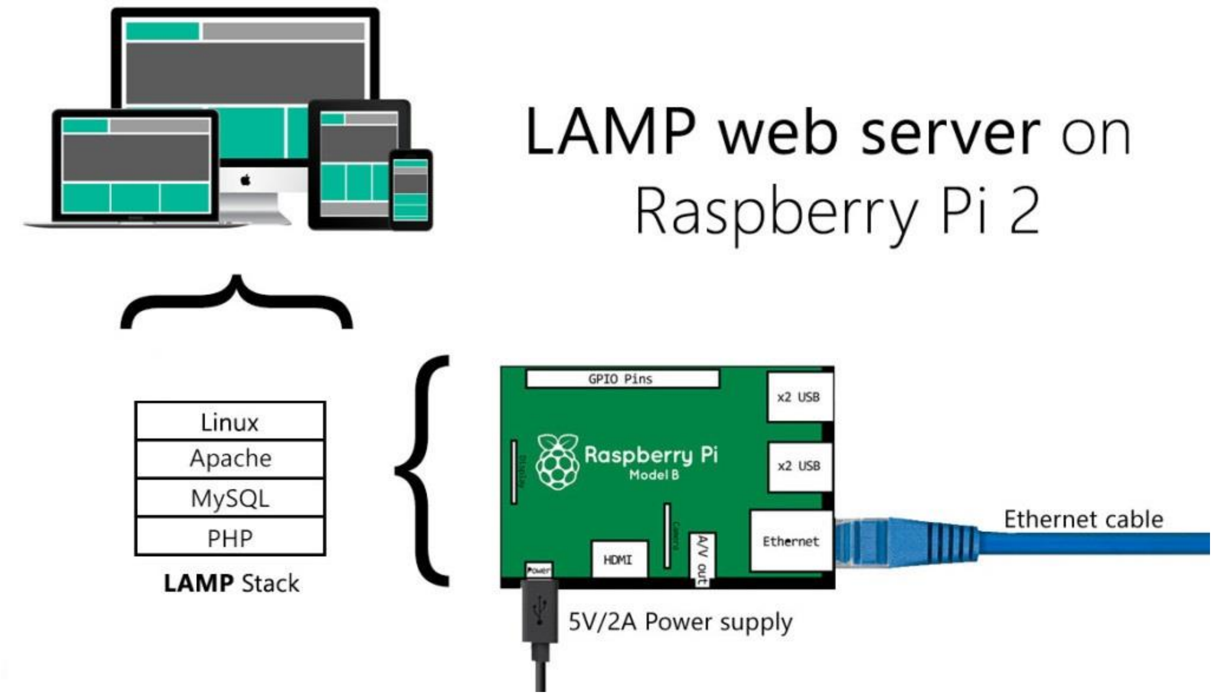


Figure 4-1 - LAMP Architecture system

The LAMP term is one of the most common configuration for webservers which standard for:

Linux – operation System;

Apache- webserver (http) software;

Mysql – database server;

PHP or Perl – programming languages;

The main reason to use the raspberry pi as a webserver is because he is dedicated network device, and there is an equipment very accessible in terms of costs and efficiency, making is job perfectly. The operating system and all mechanisms needed is freeware.

All the configuration was made at the command line, and is possible to access to the webserver through a secure shell (ssh) connection.

4.1.Database

One of the most important things in this work is the conception and the implementation of the database. Because the database needs to have the right design to support all data and to store with the best manner the information of each coach and biker. All structure of the database was made in the tool called MySQL Workbench ("MySQL :: MySQL Workbench Manual", 2016). This tool delivers visual options to the developer, providing data modeling, SQL development and comprehensive administrations tools for server configuration and user administration. MySQL is the most popular appliance to development databases. The model developed for object-relational database (ORD) is shown in the fig. 4-2.

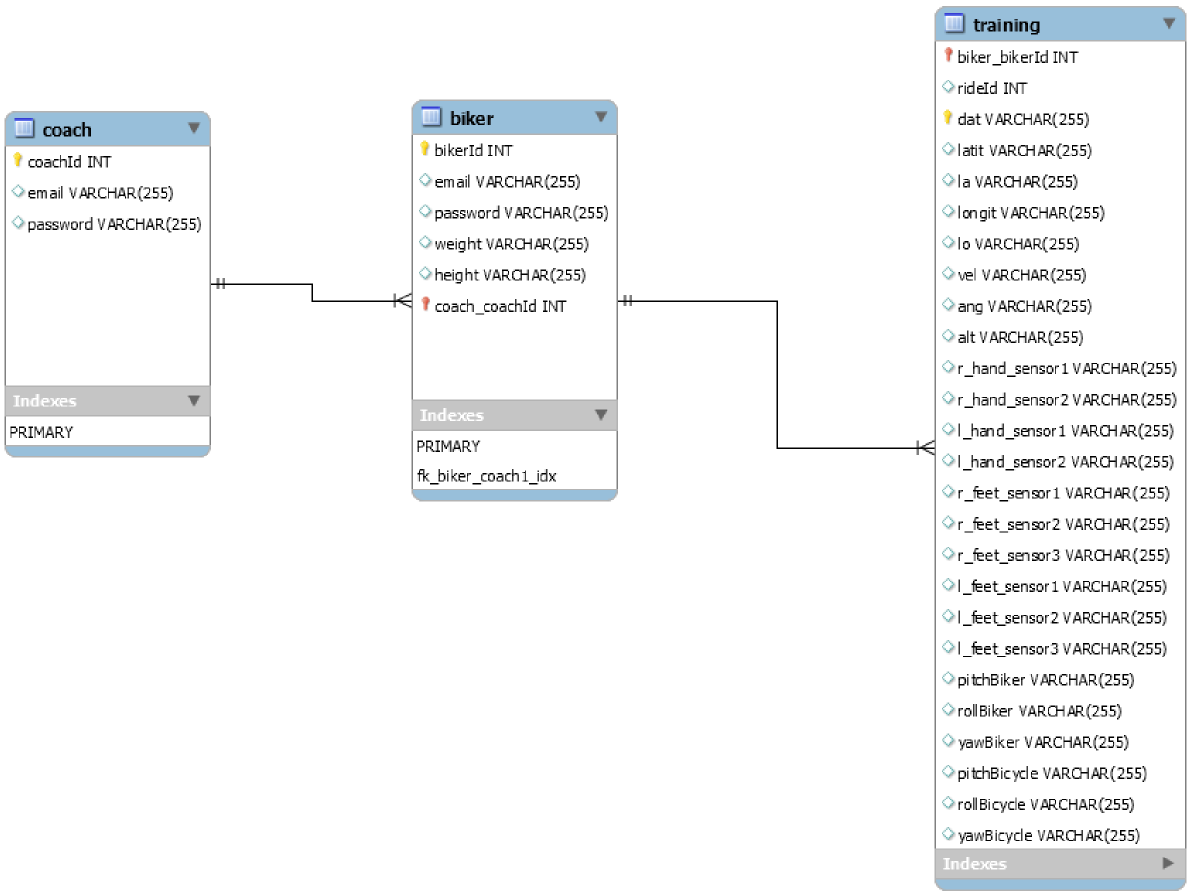


Figure 4-2 - Database Model Diagram

Like it was shown before the database is constituted with three main tables:

- Coach: There is the coach credentials. In this work the registration of the coach is made by the administrator of the database. At the beginning the registration was made in the application, but this way was not possible to control these same records. Therefore, it was decided disable this option. Only after that registration the Coach could access to the database. The coaches have the possibility to register is own bikers. The primary key is CoachId.

- Biker: In this table is presented the data that a coach need to fill in the registration of a biker, the coach defines the password and the username (in this case is the email like in the coach). Despite the biker not perform is own register, the coach may provide the username and password for the biker visualize is own trainings and his evolution. The primary key is the bikerId.

- Training: There is the place were each training is stored. The values collected by the sensors’ nodes are sent by the coordinator to the table. The primary key is the dat (date of precise moment of training).

Regard to relations every coach has at least one biker, and one biker need to have one coach. The biker may have the number of trainings that he desires.

4.2.PHP Modules

The usage of the php modules is necessary for the interaction with the application and with coordinator. This php files are housed in the cloud, every time is needed to get some information from database or to insert some information these files are called. In the coordinator is used two files:

- getRiderId.php: This file is important to know how many trainings already was performed. Let’s see a simple case one biker already perform two trainings, he wants to perform another one, this simple php file goes to the database and see the number of last training.
- sendData.php: Like name indicates this file will send all data collected from sensors.

```
<?php
	include("conn.php");


	$created_date = date("Y-m-d H:i:s");
	$biker_bikerId =$_GET['biker_bikerId'];
	$rideId=$_GET['rideId'];
	$dat=$_GET['dat'];
	$latit=$_GET['latit'];
	$la=$_GET['la'];
	$longit=$_GET['longit'];
	$lo=$_GET['lo'];
	$vel=$_GET['vel'];
	$ang=$_GET['ang'];
	$alt=$_GET['alt'];
	$r_hand_sensor1=$_GET['r_hand_sensor1'];
	$r_hand_sensor2=$_GET['r_hand_sensor2'];
	$l_hand_sensor1=$_GET['l_hand_sensor1'];
	$l_hand_sensor2=$_GET['l_hand_sensor2'];
	$r_feet_sensor1=$_GET['r_feet_sensor1'];
```

```
        $r_feet_sensor2=$_GET['r_feet_sensor2'];
        $r_feet_sensor3=$_GET['r_feet_sensor3'];
        $l_feet_sensor1=$_GET['l_feet_sensor1'];
        $l_feet_sensor2=$_GET['l_feet_sensor2'];
        $l_feet_sensor3=$_GET['l_feet_sensor3'];
        $pitchBiker=$_GET['pitchBiker'];
        $rollBiker=$_GET['rollBiker'];
        $yawBiker=$_GET['yawBiker'];
        $pitchBicycle=$_GET['pitchBicycle'];
        $rollBicycle=$_GET['rollBicycle'];
        $yawBicycle=$_GET['yawBicycle'];

        $sql_insert = "insert into training (biker_bikerId,rideId,dat,latit,la,
           longit,lo,vel,ang,alt,r_hand_sensor1,r_hand_sensor2,l_hand_sensor1,l_hand_sensor2,
           r_feet_sensor1,r_feet_sensor2,r_feet_sensor3,l_feet_sensor1,l_feet_sensor2,l_feet_sensor3,
           pitchBiker,rollBiker,yawBiker,pitchBicycle,rollBicycle,yawBicycle) values ('$biker_bikerId',
           '$rideId','$created_date','$latit','$la','$longit','$lo','$vel','$ang','$alt','$r_hand_sensor1',

'$r_hand_sensor2','$l_hand_sensor1','$l_hand_sensor2','$r_feet_sensor1','$r_feet_sensor2','$r_feet_sensor3',

'$l_feet_sensor1','$l_feet_sensor2','$l_feet_sensor3','$pitchBiker','$rollBiker','$yawBiker','$pitchBicycle',
           '$rollBicycle','$yawBicycle')";
        mysql_query($sql_insert);

        if($sql_insert){
              echo "send data successful";
        }else{
              echo "send data unsuccessful";
        }

?>
```

In case of the application there are a few files that are important for the proper functioning of the application:

- connectToDB.php: Perform the connection to the database, every single php file uses this file.
- login.php: When a coach tries to access to the application, this will see if the coach is registered in the database.

- loginBiker.php: A biker tries to access to the application, this file will see if the biker is registered on the database. It's important that the biker is registered to perform a training.
- createBiker.php: Entering in the application the user will see all is bikers and has the option to register others.
- getCoachIdUsingEmail.php: At the beginning, the users enter is username (email) and with this information the file will get the CoachId.
- getCoachEmailByUsingCoachId.php: Get the email using the CoachId.
- getBikersByCoachId.php: This file will retrieve a list of the bikers that a coach is responsible.
- getTrainingArrayByBikerId.php: Will retrieve a list in a json format of all training performed for a given BikerId.
- getTrainingArrayByRiderId.php: When the coach or biker choose which training want to analyze this will get the individual training number performed.

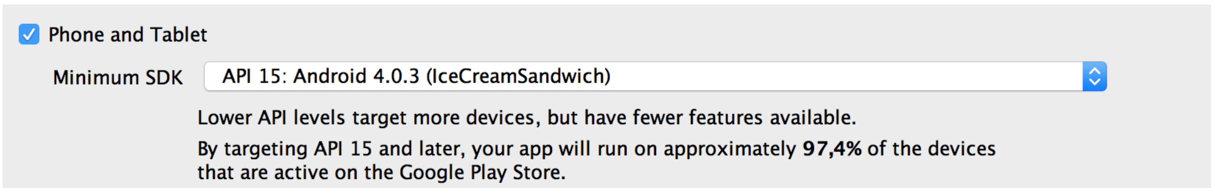
All this programming could be done in a single file, but for a good comprehension and good debugging it was decided to make it by step by step. If any problem occurs is easy to detect the fault.

.

Chapter 5 - Application

5. Mobile Application

The application is to allow very friendly access to each training session, with this application every coach and biker can access to every training session that already perform. The mobile application was developed with Android studio and can run approximately in 97,4% of the devices like is presented in fig. 5-1. The minimum operating system version targeting tablets and phones with the Android 4.0.3 (IceCreamSandwich). To execute the application, the android device necessarily requires a connection to the internet either mobile network or Wi-Fi.



✓ Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available.

By targeting API 15 and later, your app will run on approximately **97,4%** of the devices that are active on the Google Play Store.

Figure 5-1 - Configuration Android Studio

5.1.Sequence Diagram

The sequence of activities in the application have two options, depending which user is logged Coach fig. 5-2 or Biker fig. 5-3.

5.1.1. Coach Sequence Diagram

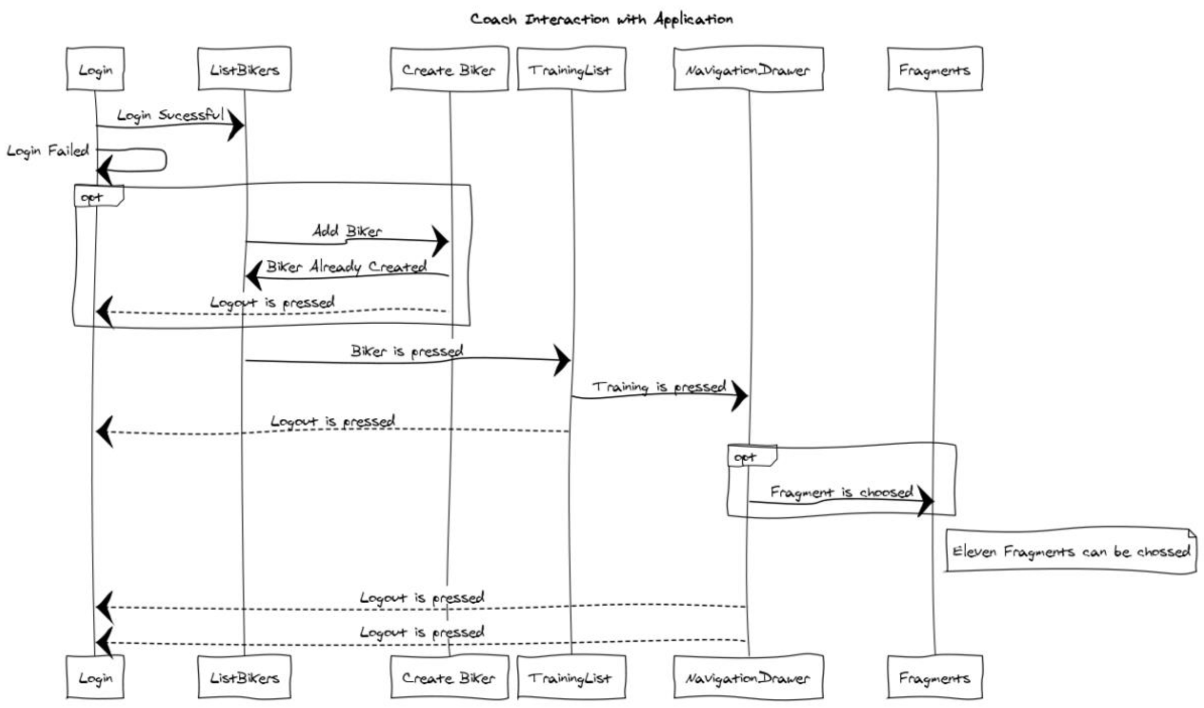


Figure 5-2 - Sequence diagram Coach interaction

5.1.2. Biker Sequence Diagram

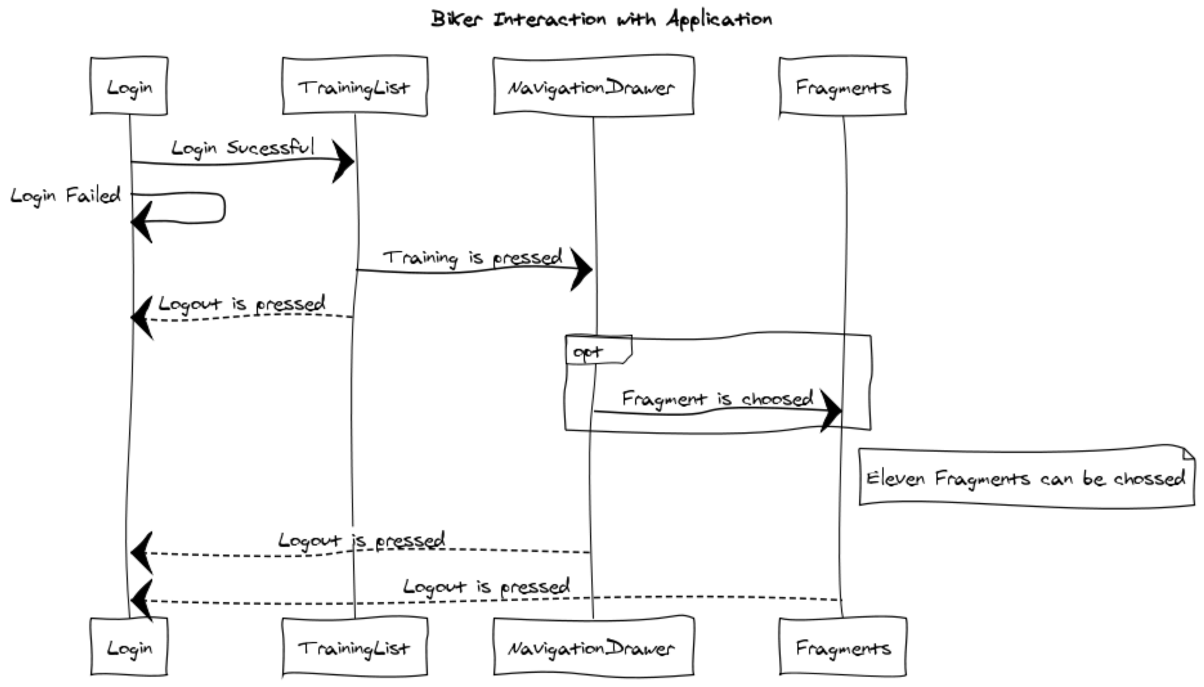


Figure 5-3 - Sequence diagram Biker interaction

Well in the diagrams is possible to see every interaction that a coach or a biker may make in the mobile application. In the login Activity fig. 5-4 the user must to choose if he had coach or biker account. By default, is selected the coach account.

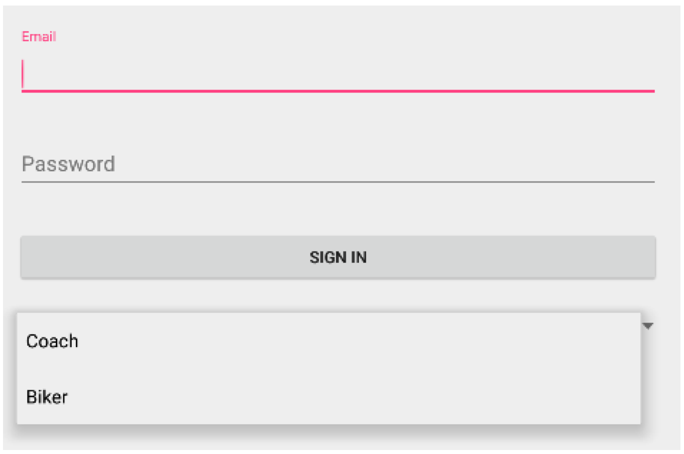


Figure 5-4 - Login view

Depending of this choice the mobile application will behave according to the diagrams presented. In the case of the fragments is where the analyze is performed, this is, the report and graphs are presented in the Fragment in total there is eleven fragments.

1. Report: present some results like the duration, maximum altitude, minimum altitude, distance, calories, maximum velocity and average speed;

```
duration = getTrainingDuration(trainingDataList);
maxAltitude = train.round(getMaxAltitude(trainingDataList),2);
minAltitude = train.round(getMinAltitude(trainingDataList),2);
maxVelocity = train.round(getMaxVelocity(trainingDataList),2);
averageSpeed = train.round(getAverageSpeed(trainingDataList),2);
distance = train.round(getDistance(trainingDataList),2);
calories = train.round(METS*getCaloriesBurned(trainingDataList)*Float.parseFloat(weight),2);

/**
 *Calculate all the time in hours that a certain biker made during is training session
 * @param trainingDataList
 * @return result (time spent in hours during the training)
 */

private float getCaloriesBurned(ArrayList<Training> trainingDataList) {

   float result=0;
   ArrayList<String> tempList = new ArrayList<>();
   for (int i = 0; i<trainingDataList.size();i++){
      tempList.add(trainingDataList.get(i).getDat().substring(0,8).trim());

   }
   Collections.sort(tempList);

   SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
```

```
    try {
        Date startTime = simpleDateFormat.parse(tempList.get(0).trim());
        Date endTime=simpleDateFormat.parse(tempList.get(tempList.size()-1).trim());


        long difference = endTime.getTime()-startTime.getTime();

        int diffSeconds = (int) difference / 1000 % 60;
        int diffMinutes =  (int) difference / (60 * 1000) % 60;
        int diffHours =  (int) difference / (60 * 60 * 1000) % 24;
        //long diffDays = difference / (24 * 60 * 60 * 1000);
        result= diffHours+diffMinutes*MINUTETOHOUR+diffSeconds*SECONDSTOHOUR;
    } catch (ParseException e) {
        e.printStackTrace();
    }



    return result;

}

/**
 *Calculate all duration of the training session
 * @param trainingDataList
 * @return value (present in HH:mm:ss)
 */
private String getTrainingDuration(ArrayList<Training> trainingDataList) {

    String value = "";
    ArrayList<String> tempList = new ArrayList<>();
    for (int i = 0; i<trainingDataList.size();i++){
        tempList.add(trainingDataList.get(i).getDat().substring(11,19).trim());

    }
    Collections.sort(tempList);

    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");


    try {
        Date startTime = simpleDateFormat.parse(tempList.get(0).trim());
        Date endTime=simpleDateFormat.parse(tempList.get(tempList.size()-1).trim());


        long difference = endTime.getTime()-startTime.getTime();

        int diffSeconds = (int) difference / 1000 % 60;
        int diffMinutes =  (int) difference / (60 * 1000) % 60;
        int diffHours =  (int) difference / (60 * 60 * 1000) % 24;
        //long diffDays = difference / (24 * 60 * 60 * 1000);
        value= diffHours+":"+diffMinutes+":"+diffSeconds;
    } catch (ParseException e) {
        e.printStackTrace();
    }
    return value;
}

/**
 * calculate the average of the velocity
 * @param trainingDataList
 * @return average velocity
 */
private float getAverageSpeed(ArrayList<Training> trainingDataList) {

    float averageValue = 0;

    for (int i=0; i< trainingDataList.size();i++){

         averageValue+=trainingDataList.get(i).getVel();
    }

    return averageValue/trainingDataList.size();

}

/**
 * get the Max velocity during all training session
 * @param trainingDataList
 * @return fastestValue
 */
private float getMaxVelocity(ArrayList<Training> trainingDataList) {

    float fastestValue = Float.MIN_VALUE;

    for (int i= 0; i<trainingDataList.size();i++){
        if ((trainingDataList.get(i).getVel())>fastestValue){
            fastestValue=(float) trainingDataList.get(i).getVel();
        }
    }


    return  fastestValue ;
}

/**
 *get the minimum altitude of the training session
 * @param trainingDataList
 * @return smallestAltitude
 */
private double getMinAltitude(ArrayList<Training> trainingDataList) {

    float smallestAltitude = Float.MAX_VALUE;

    for (int i =0; i< trainingDataList.size();i++){
        if ((trainingDataList.get(i).getAlt())< smallestAltitude){
            smallestAltitude=(float) trainingDataList.get(i).getAlt();
        }
    }


    return smallestAltitude;
}

/**
```

```
 *get the maximum altitude from the training session
 * @param trainingDataList
 * @return biggestAltitude
 */
private float getMaxAltitude(ArrayList<Training> trainingDataList) {


    float biggestAltitude = Float.MIN_VALUE;

    for (int i=0; i< trainingDataList.size();i++){
        if (( trainingDataList.get(i).getAlt())>biggestAltitude){
             biggestAltitude=(float) trainingDataList.get(i).getAlt();

        }
    }

    return biggestAltitude;
}

/**
 *Convert the latitude and longitude from DDMM to Decimal degrees
 * @param value
 * @return result
 */

private float convertDegreeMinuteMToDecimal(float value) {

    int indexOfPoint;
    indexOfPoint = new Double(value).toString().indexOf('.');

    String value1 = String.valueOf(value);

    String aux1=value1.substring(0,indexOfPoint-2);
    String aux2=value1.substring(indexOfPoint-2,value1.length());

    float result=Float.parseFloat(aux1)+(Float.parseFloat(aux2)/60);

    return result;
}

/**
 *Receive one list and calculate the total distance of the path
 * @param listOfRides
 * @return distance
 */

private float getDistance( ArrayList<Training> listOfRides){

    ArrayList<Training> tempList = new ArrayList<>();
    double distance = 0;
    for (int i =0; i<listOfRides.size();i++){


        Training training = new Training(listOfRides.get(i).getDat(),convertDegreeMinuteMToDecimal((float)
listOfRides.get(i).getLatit()),listOfRides.get(i).getLa(),convertDegreeMinuteMToDecimal((float)
listOfRides.get(i).getLongit()),listOfRides.get(i).getLo());
        tempList.add(training);

    }
    Collections.sort(tempList, new TrainingComparator());
    for (int i =0; i<tempList.size();i++){
        //  Log.d("tempListDat",""+tempList.get(i).getDat()+" "+tempList.get(i).getLatit()+"
"+tempList.get(i).getLongit());
        if (i!=0){

            float dLat= (float) Math.toRadians(tempList.get(i).getLatit()-tempList.get(i-1).getLatit());
            float dLong = (float) Math.toRadians(tempList.get(i).getLongit()-tempList.get(i-1).getLongit());

            float a = (float) (Math.sin(dLat/2)*Math.sin(dLat/2)+Math.cos(Math.toRadians(tempList.get(i-
1).getLatit()))*Math.cos(Math.toRadians(tempList.get(i).getLatit()))*
                    Math.sin(dLong/2)*Math.sin(dLong/2));
            float c = (float) (2*Math.atan2(Math.sqrt(a),Math.sqrt(1-a)));

            distance+=RADIUSEARTH*c;
        }


    }
    return (float)train.round(distance,2);
}
```

Is important to refer that for the calculation of the calories, the formula above is used:

$$Calories = METS \times Body\ Weight(Kg) \times Time(h) \tag{14}$$

The formula is used to simplify the calculation of the energy expended in the physical activity. In order to simplify the calculation, the scientists have come up with a measure called MET. One MET is the energy an average person burns at rest per kilogram (kg) of body weight per hour (h). In other words, one MET is the calories 1 kg of resting human tissue burns in 1h. It is also known as the RMR. Essentially, with the introduction of MET concept, the rate of energy we spend performing various tasks is expressed as a multiple of our RMR. For example, a physical activity that has two METS requires twice the energy we spend when we rest.

Sports physiologists have calculated the MET values for many sports and daily physical activities. So the estimated value for bicycling, BMX or mountain is 8.5 METS (Levi, 2013).

2. Force Sensor Gloves: this fragment is responsible for presenting the graphs of the four sensors in gloves athlete;
3. Force Sensor Shoes: In this fragment is possible to see the graphs that translate the force of the six mounted sensors;
4. GPS Tracking: Gives an idea of the route that the biker performed on is workout;

5. Velocity: present the velocity graph in km/h;
6. Altitude: shows an altitude graph in meters.
7. Angle: Demonstrate the angle that the GPS sensor is making with Satellite.
8. Angles Biker: In the chest of the biker the IMU calculate the pitch, roll and yaw of the biker and in the fragment is presented the graph of that values.
9. Angles Bike: Like in the Biker, the bicycle has another IMU. So to see the movements of the bicycle the graph is shown in this fragment.
10. IMU Bicycle and Biker difference: To see the difference between the biker and the bicycle movements this fragment present a graph of that.
11. Export: All the data is saved in the database, but if the user wants to export the training data here is possible, only need to give a file name and press a button.

5.2.Main Features

This application uses a two secure authentication, if any user wants to use this application he needs to be register in the database. Only registered users can access to this application, in order to protect user data. This mechanism is based in username and password to identify the user and login into the application. At the coordinator only bikers that have a card or tag registered can perform a training session.

The communication with the database in the application is made through Hypertex Transfer Protocol (HTTP) requests using the Volley library ("mcxiaoke/android-volley", 2016). Volley is an HTTP library that makes possible the networking for Android easier and faster.

For the visual presentation of the data collected from the database, is used two libraries GraphView ("Graph View", 2016) and MPAndroidChart ("PhilJay/MPAndroidChart", 2016) with these two libraries is possible to create the line graphs, bar charts and the points Charts that is possible to observe in the fragments. With these libraries is possible to display to the user's multiple series of data, scroll with a finger touch move gesture and read the values from the graphs. Allowing the user analyze every aspect of the graph created.

Chapter 6 Results and Evaluation

6. Evaluation

At the beginning in order to obtain the first results and to design the WSN there was the need to create simple circuits fig. 6-1 and fig. 6-2.

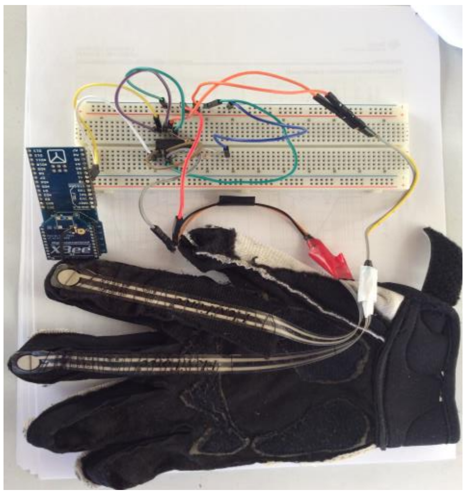


Figure 6-1 - Force sensors gloves initial assembly

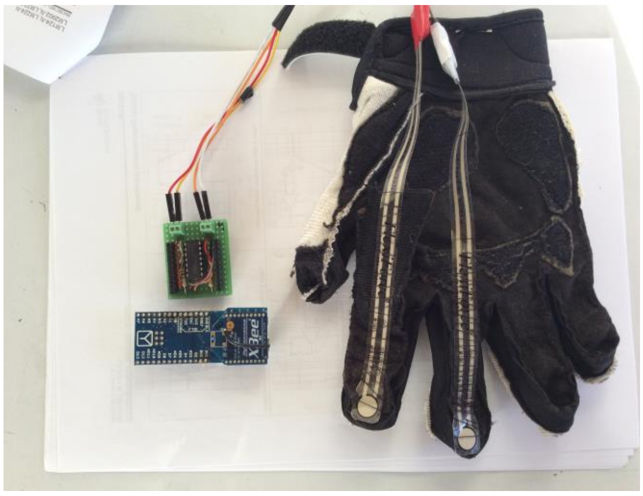


Figure 6-2 - Force sensors gloves finally assembly

At the first image the two sensors are connected to the condition circuit, in the second image the connections are the same but the condition circuit now are placed in a board. The connections are the same but is possible to see that all the cables are soldered to silicon plate. This work was made for the gloves and shoes.

In the case of the IMU is little more easy, only four connections are needed.

In the case of the coordinator fig. 6-3 the story is more complex, well the connections are made to the shields and to every shield it was necessary to test every connection and every function.

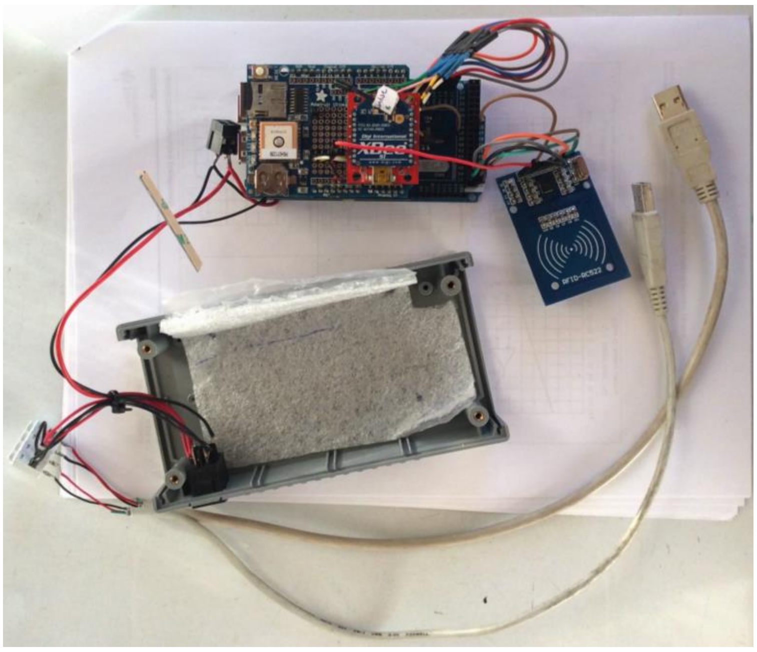


Figure 6-3 - Coordinator assembly

Every microcontroller have is own individual program, and all end-nodes communicate with coordinator, the communication is made using the XBee. So to visualize the data is very important to create the android application.

All the microcontrollers are conditioning in little bags fig. 6-4, fig. 6-5 and fig. 6-6, so they cannot interfere with the natural movement of the cycling and has comfortable to use. For the acquisition of the values the android application will not needed to be operational, the only things that is needed is the registration of the coach and biker.

In the case of fig. 6-4 and fig. 6-5 is presented the equipment of the biker with the embedded sensors Gloves and insoles.

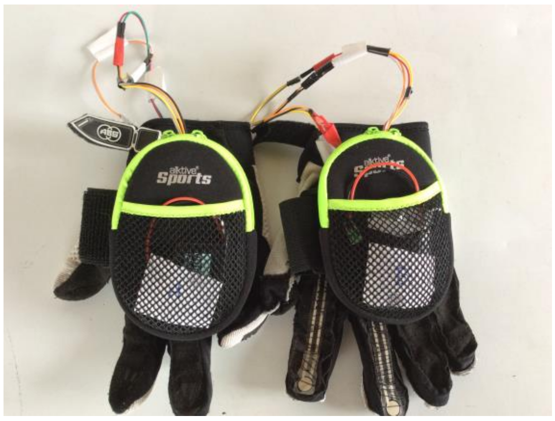


Figure 6-4 - Finnaly Assembly Gloves Sensors

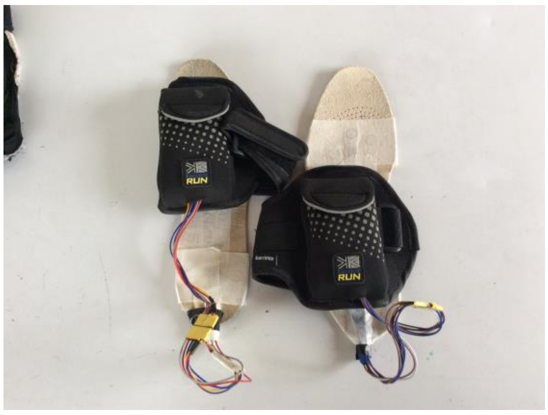


Figure 6-5 - Finally assembly shoes sensors

In the frame of the bicycle fig. 6-6 is placed the coordinator, in order to protect and to secure the shields and the microcontroller to the bicycle is used a little bag. Like we see and explain above the objective is to get all the data from the others end-nodes and to send the data to the cloud.

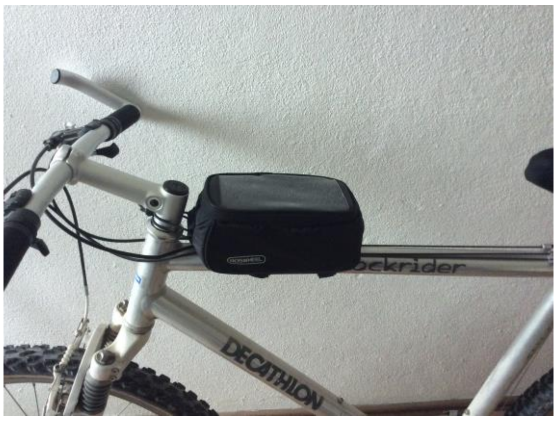


Figure 6-6 - Finally assembly coordinator

6.1.Results

Several tests were carried out using the implemented system and the data are presented in the android application. During the training session several variables are collected and are expressed by:

1. Date, Latitude, Longitude, Altimetry, Velocity, Angle – delivered by GPS Shield;
2. Motions of the Biker and Bicycle (pitch, roll and yaw) – delivered by IMU boards nodes
3. Force applied on brakes and pedals (ten sensors)- delivered by Force Sensors Nodes

During the training session is perceptible to understand which path the rider traveled, here the coach can see the latitude and longitude where the rider has, only with touch of a finger.

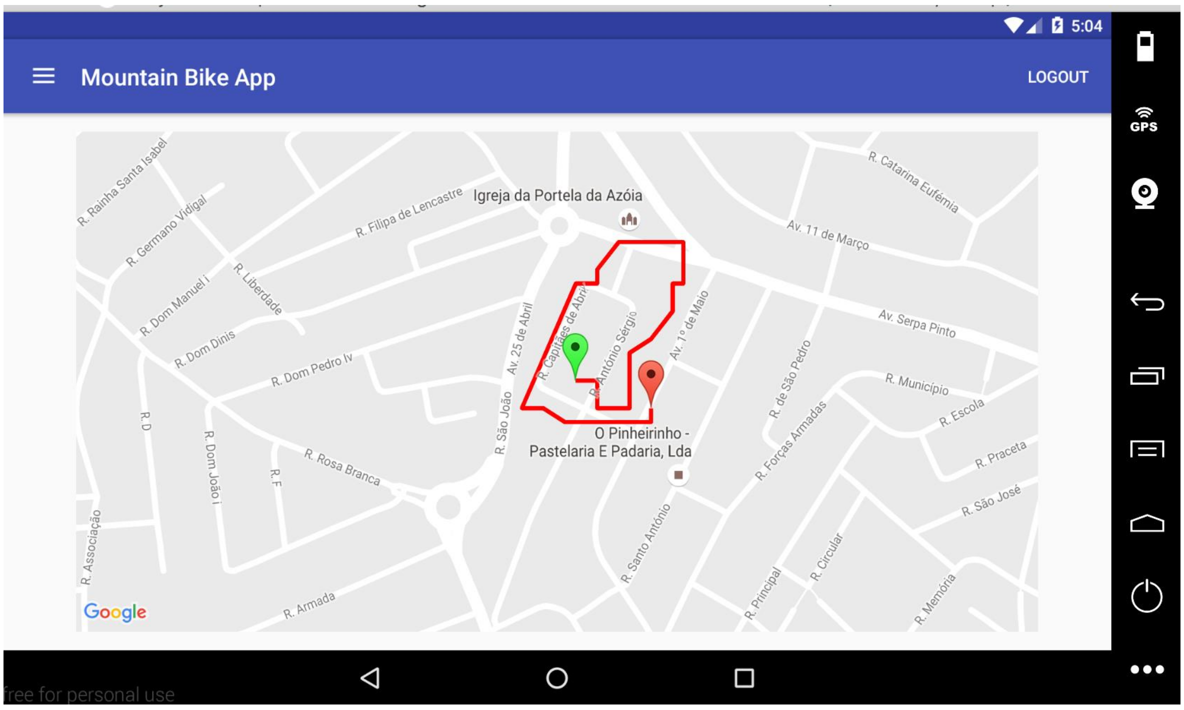

5:04
Mountain Bike App
LOGOUT
GPS
Igreja da Portela da Azóia
O Pinheirinho - Pastelaria E Padaria, Lda
Google
free for personal use

Figure 6-7 – Application GPS analyze

Here the velocity during the track is presented and analyzed, velocity vs. time.

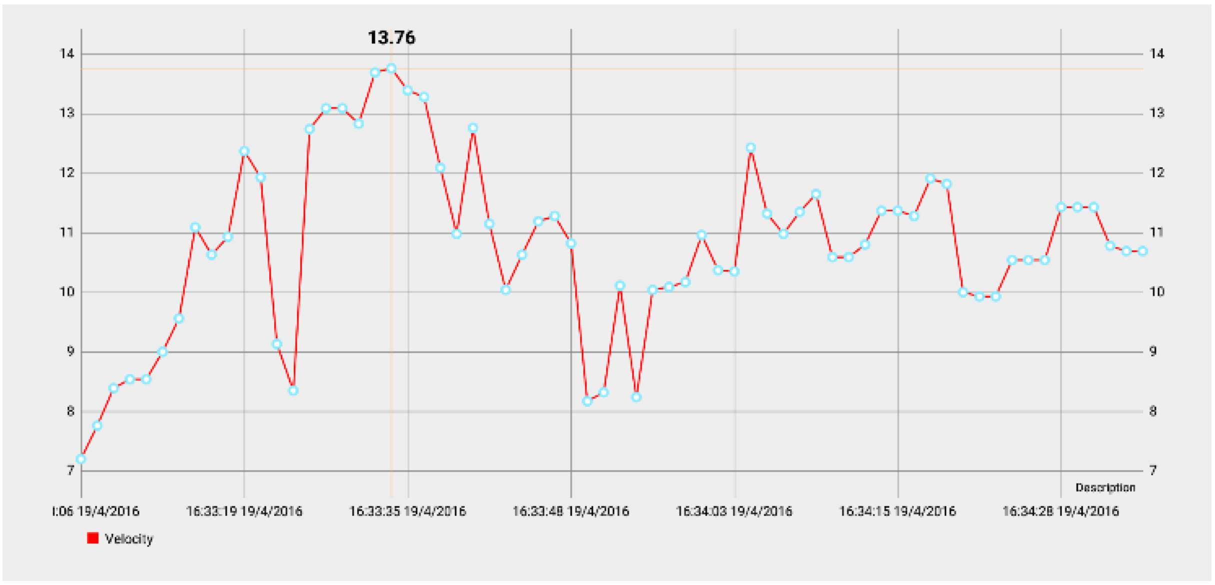


Figure 6-8 - Application velocity analyze

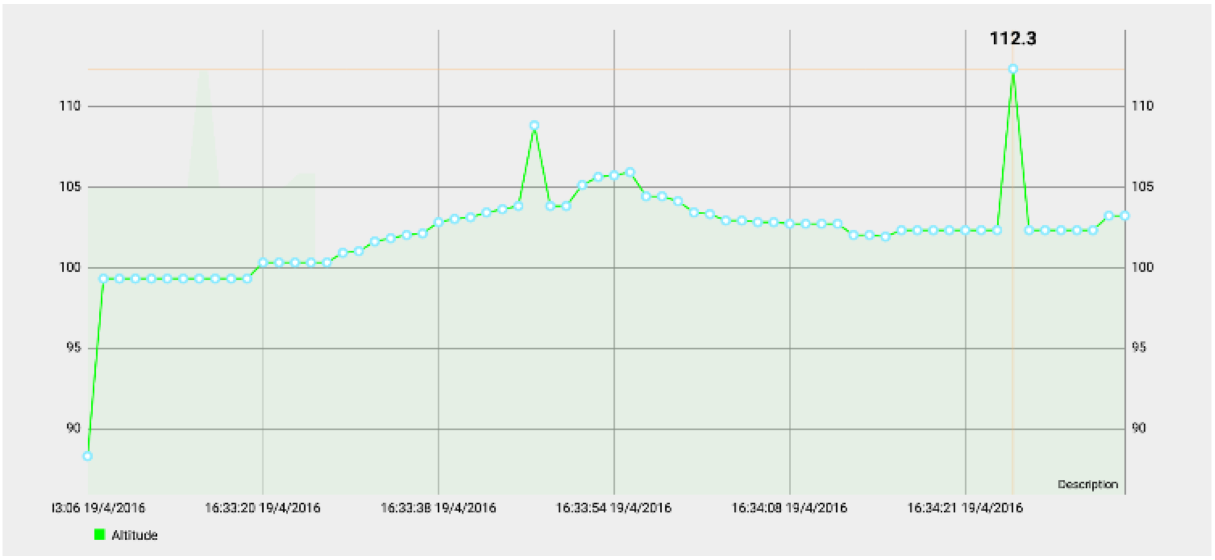


Figure 6-9 - Application altitude analyze

The elevation is a very important point to consider, because depending on the altitude will require more effort by the bikers.

In the case of the information collected from the force sensors nodes, the fig. 6-11 shows the values of force applied to the brakes. In here is represented the four sensors in the gloves. The values represented in this graph is the velocity (Km/h) vs. force applied (N).

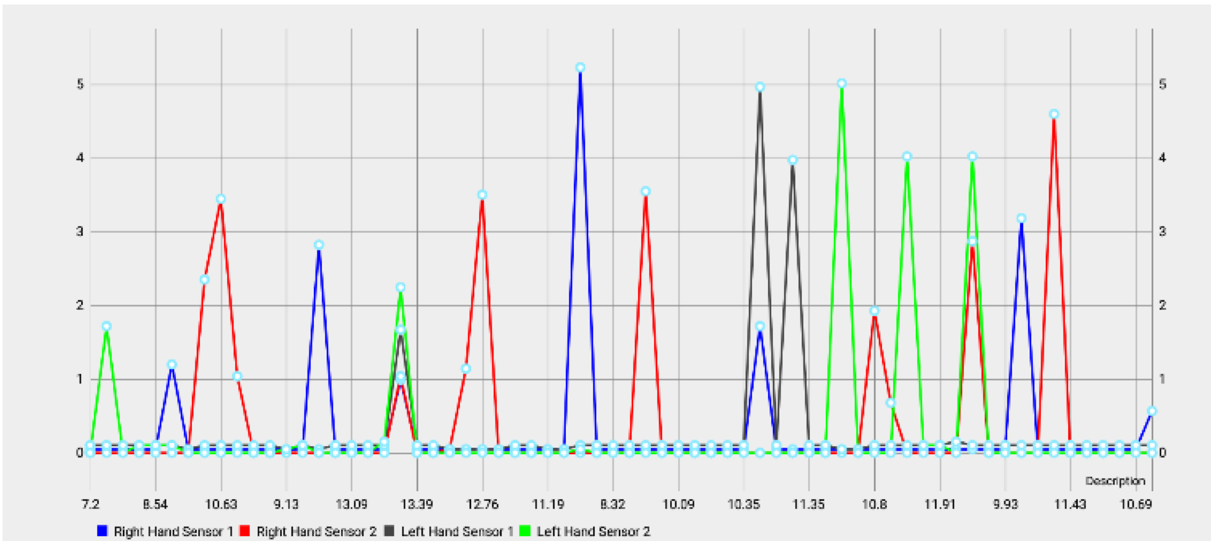


Figure 6-10 - Application analyze gloves sensors (Force)

This bar chart represents the utilization of each force sensor during the training session (e.g., 5 min). Through the fig. 6-11 is possible to see that right hand sensor 2 is the most used by the biker during the training session.

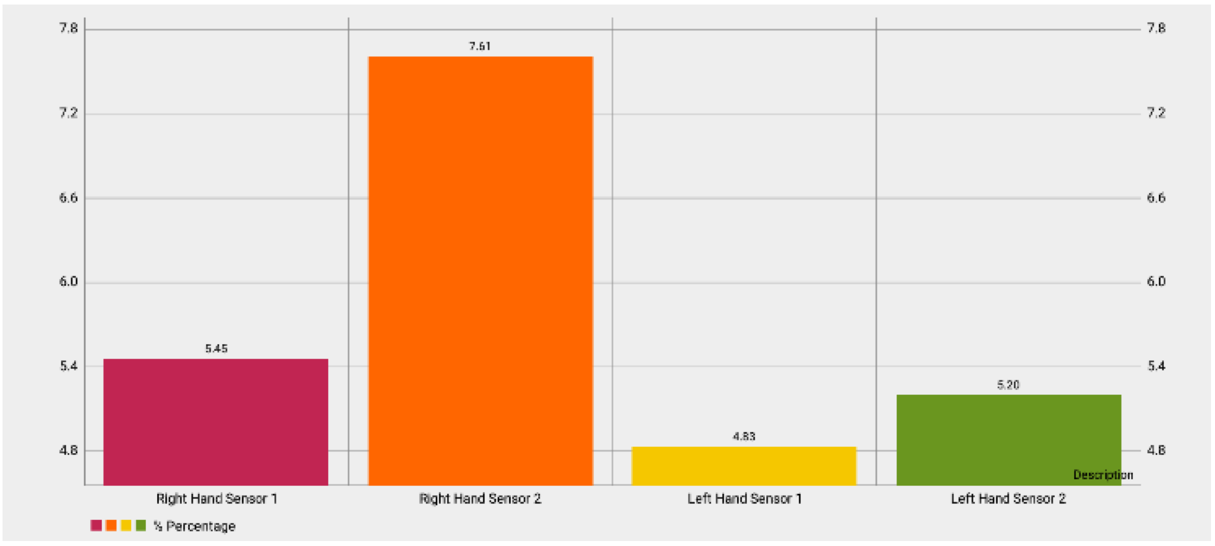


Figure 6-11 - Application analyze gloves sensors (% usage)

Like in the gloves, for the shoes we perform the same analyze, with the difference that we have six force sensors to collect information.

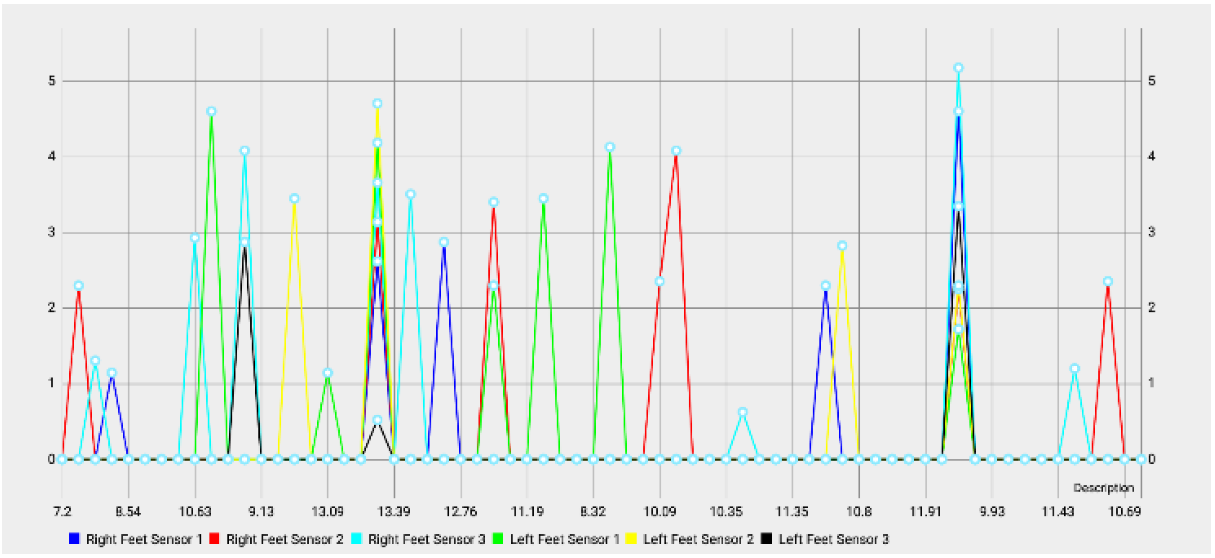


Figure 6-12 - Application analyze shoes sensors (Force)

Here is represented the percentage of the usage of the force sensors in the shoes, during the training session.

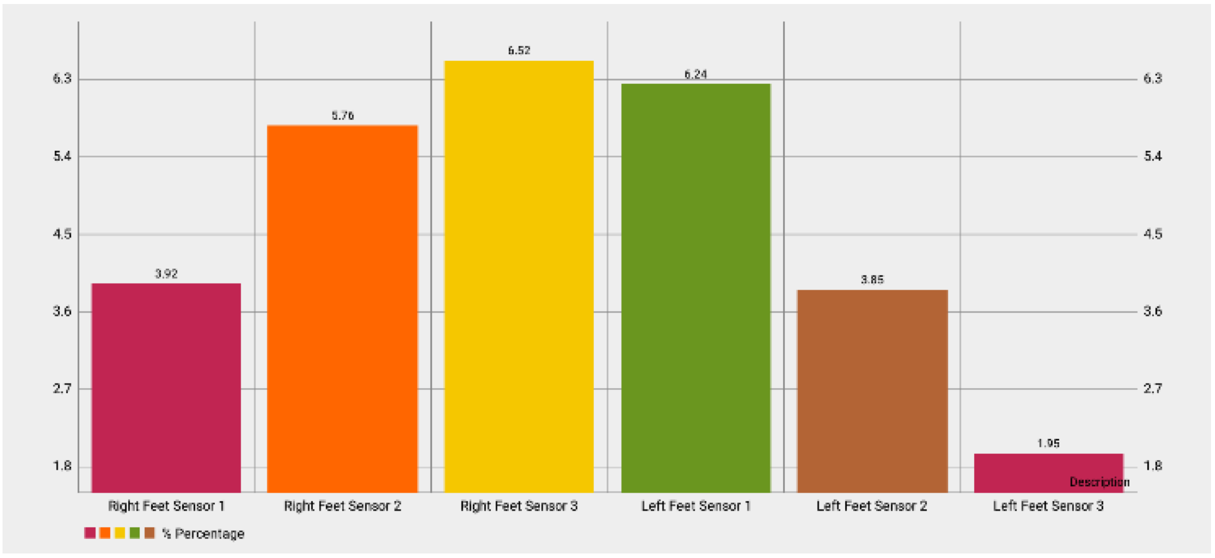


Figure 6-13 - Application analyze shoes sensors (% usage)

Chapter 7 - Conclusions

The main goal of this dissertation was to design and implement of a Wireless Sensor Network collect information during the training in the mountain bike sport modality.

Hardware and software components were designed and tested. Referring the hardware part several conditioning circuits, coordinator and end-nodes modules including sensors were developed. The WSN nodes were distributed on the smart gloves and smart shoes level, that communicate with the coordinator node through a wireless connection. In total are present six end-nodes (RFD) and a coordinator (FFD). The coordinator works as smart gateway and it performs the data acquisition (receive all information of the other end-nodes) data processing, prepares the information and send it through Wi-Fi connection to the server.

In the server “Cloud” is used a LAMP architecture that supports a database created for this demanding project. The LAMP architecture presents advantages in terms of stability and reliability of Unix operating system, fast server side scripting and high performance database interaction. The communication with the coordinator, database and application is achieved through of the usage of PHP modules. With these PHP modules it is possible to read and write in the database in a simple and effective way.

For the complementation of this work was developed an Android application that permits to a user to be registered in the database, and to access of all history and all training sessions analyzing the information collected from the sensors.

This whole system was created with the purpose of helping to improve the overall performance of bikers, analyzing their data training collected by the wireless sensor nodes. Using the mobile developed application, a biker’s performance analysis can be carried out by coach but also by the biker.

To summarize, the system’s architecture is responsible for the acquisition, processing and data display of the training data including historical data stored in the cloud. Based on the developed Android APPs the data analysis can be performed in any smartphone or tablet and helps every biker to increase performance and improve the methodology of his own training.

6.2. Contributions

This dissertation offers as original contributions regarding the design and implementation of a system, characterized by the usage of M2M, WSN and Cloud Computing technologies for athlete's performance assessment, that are mountain bike users.

Part of the work and preliminary results were included as part of an article (Appendix A) that was accepted, presented in an international conference IEEE EPE 2016, 20-22 October, Iasi Romania. The article will be published in the IEEEXplorer.

Ribeiro T., Postolache O., and Passos P., "Performance Assessment for Mountain bike based on WSN and Cloud Technologies", EPE 2016, IASI Romania 2016 International Conference and Exposition on Electrical and Power Engineering.

A User Manual (Appendix B) and a Technical Manual (Appendix C) were also made to assist in the use and understanding of this whole Project.

6.3. Future Work

For the future work, some new features and improvements can be done in order to improve the system:

- Introducing better force sensors, more flexible;
- Introducing better GPS module, to allow a better perspective of the track;
- Improve server and the database, more security, storage capacity, better processing and better connection to the Ethernet;
- Create smaller PCB circuits using lightweight materials;
- Create a central switch for all nodes to turn on/ off each node.
- In the application more methods to correlate data can be done (for example, introduce technical parameters of the bicycle, weight, tire type, gear ratio, etc,.).

References

A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. « Starlino Electronics. (2009). Starlino.com. Retrieved 9 June 2016, from http://www.starlino.com/imu_guide.html

Abvio | Cyclemeter. (2016). Abvio.com. Retrieved 9 September 2016, from https://abvio.com/cyclemeter/

Ahmed, K., & Gregory, M. (2011). Integrating wireless sensor networks with cloud computing. Proceedings - 2011 7th International Conference on Mobile Ad-Hoc and Sensor Networks, MSN 2011, 364–366. http://doi.org/10.1109/MSN.2011.86

Akyildiz, I. F., Gutierrez-Estevez, D. M., & Reyes, E. C. (2010). The evolution to 4G cellular systems: LTE-Advanced. Physical Communication, 3(4), 217–244. http://doi.org/10.1016/j.phycom.2010.08.001

Almassri, A. M., Hasan, W. Z. W., Ahmad, S. A., Ishak, A. J., Ghazali, A. M., Talib, D. N., & Wada, C. (2014). Pressure Sensor: State of the Art, Design, and Application for Robotic Hand. Journal of Sensors, 2015.

Arduino - ArduinoBoardFio. (2016). Arduino.cc. Retrieved 25 March 2016, from https://www.arduino.cc/en/Main/ArduinoBoardFio

Arduino - ArduinoBoardLeonardo. (2016). Arduino.cc. Retrieved 9 May 2016, from https://www.arduino.cc/en/Main/ArduinoBoardLeonardo

Arduino - ArduinoBoardMega. (2016). Arduino.cc. Retrieved 9 May 2016, from https://www.arduino.cc/en/Main/arduinoBoardMega

Arduino - Software. (2016). Arduino.cc. Retrieved 22 September 2016, from https://www.arduino.cc/en/Main/Software

Barreiro, J., Postolache, O., & Passos, P. (2014). WSN and M2M for Mountain Biking Performance Assessment. ISCTE-IUL.

Beck, R. F. (2009). Mountain bicycle acceleration and braking factors. Accident Reconstruction Journal, 19(4), 49–56.

Bell, C. (2013). Beginning sensor networks with Arduino and Raspberry Pi. [New York]: Apress.

Bicycle dynamics, control and handling — Sports Biomechanics Lab. (2016). Biosport.ucdavis.edu. Retrieved 15 August 2016, from http://biosport.ucdavis.edu/research-projects/bicycle#section-8

Bike Gears, Bike Gear Calculator application for iPhone and iPod Touch.. (2016). Bikegearcalculator.com. Retrieved 9 September 2016, from http://www.bikegearcalculator.com

Bike Maps, Cycling Workout, Biking Routes | MapMyRide. (2016). Mapmyride.com. Retrieved 9 September 2016, from http://www.mapmyride.com

Bikes | Trek Bikes. (2016). Trekbikes.com. Retrieved 9 April 2016, from http://www.trekbikes.com/us/en_US/bikes/c/B100

Campos, A., Fonseca, I., & Lopes, F. (2011). Projecto e Construção de um Sistema Embebido de Tempo-Real Baseado em Linux. ISEC.

Chung, W., Yu, P., & Huang, C. (2013). Cloud computing system based on wireless sensor network. *Computer Science and …*, 877–880. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6644114

cyclists, H. (2016). Heart-rate monitor training for cyclists. BikeRadar. Retrieved 13 August 2016, from http://www.bikeradar.com/gear/article/heart-rate-monitor-training-for-cyclists-28838/

Dobrnjac, M. (2016). ANNALS of Faculty Engineering Hunedoara – International Journal of Engineering THERMAL CHARACTERISTICS AND POTENTIAL FOR, 41–45.

Elahi, A. & Gschwender, A. (2010). ZigBee wireless sensor and control network. Upper Saddle River, N.J.: Prentice Hall.

Endomondo. (2016). Endomondo.com. Retrieved 9 September 2016, from https://www.endomondo.com

Ethernet Tutorial - Part I: Networking Basics | Lantronix. (2016). Lantronix.com. Retrieved 18 May 2016, from https://www.lantronix.com/resources/networking-tutorials/ethernet-tutorial-networking-basics/

Fässler, M. (2010). Force Sensing Technologies. Studies on Mechatronics , ETH, 1–49.

FlexiForce A201 Sensor. (2014). Tekscan. Retrieved 20 April 2016, from https://www.tekscan.com/products-solutions/force-sensors/a201

García-hernández, C. F., Ibargüengoytia-gonzález, P. H., García-hernández, J., & Pérez-díaz, J. a. (2007). Wireless Sensor Networks and Applications : a Survey. Journal of Computer Science, 7(3), 264–273. http://doi.org/10.1109/MC.2002.1039518

GlobalTop Technology Inc. (2011). FGPMMOPA6H GPS Standalone Module Data Sheet, (16), 1–37.

Goncalves, A., Thomaz, S., Sa, D. L., & Henrique, D. (2016). Towards a Real-Time Embedded System for Water. Sensors 2016, 1–20. http://doi.org/10.3390/s16081226

Guedes, T., Neto, H., & Véstias, M. (2007). Câmara em Rede com Tecnologia FPGA. IST.

Hazry, D., Sofian, M., & Azfar, a Z. (2009). Study of Inertial Measurement Unit Sensor. International Conference on ManMachine Systems ICoMMS, (October), 11–13.

Henzinger, T. A., & Sifakis, J. (2006). The Embedded Systems Design Challenge. In J. Misra, T. Nipkow, & E. Sekerinski (Eds.), *FM 2006: Formal Methods: 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006. Proceedings* (pp. 1–15). inbook, Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/11813040_1

Hillman, M. (n.d.). An Overview of ZigBee Networks A guide for implementers and security testers. Retrieved from https://www.mwrinfosecurity.com/assets/Whitepapers/mwri-zigbee-overview-finalv2.pdf

HISTORY | The Marin Museum of Bicycling and Mountain Bike Hall of Fame. (2016). Mmbhof.org. Retrieved 9 September 2016, from http://mmbhof.org/mtn-bike-hall-of-fame/history/

Höfer, C. N., & Karagiannis, G. (2011). Cloud computing services: Taxonomy and comparison. Journal of Internet Services and Applications, 2(2), 81–94. http://doi.org/10.1007/s13174-011-0027-x

Home - GPS Tracking Systems for Elite Sports. (2016). GPSports | Global Positioning Systems. Retrieved 11 August 2016, from http://gpsports.com

Honglong, C., Liang, X., Wei, Q., Guangmin, Y., & Weizheng, Y. (2008). An integrated MEMS gyroscope array with higher accuracy output. Sensors, 8(4), 2886–2899. http://doi.org/10.3390/s8042886

Huang, Y., Hsieh, M., & Sandnes, F. E. (2008). Wireless Sensor Networks and Applications. Communication, 2004, 199–219. http://doi.org/10.1007/978-0-387-49592-7

I2C - learn.sparkfun.com. (2016). Learn.sparkfun.com. Retrieved 15 September 2016, from https://learn.sparkfun.com/tutorials/i2c

Instrumented Bicycle — Sports Biomechanics Lab. (2016). Biosport.ucdavis.edu. Retrieved 15 August 2016, from http://biosport.ucdavis.edu/research-projects/bicycle/instrumented-bicycle

Jary,. (2014). What is the difference between 3G, 4G, GPRS, E and Wi-Fi. PC Advisor. Retrieved 11 July 2016, from http://www.pcadvisor.co.uk/feature/network-wifi/what-is-difference-between-3g-4g-gprs-e-wi-fi-3509254/

Kartsakli, E., Lalos, A., Antonopoulos, A., Tennina, S., Renzo, M., Alonso, L., & Verikoukis, C. (2014). A Survey on M2M Systems for mHealth: A Wireless Communications Perspective. Sensors, 14(10), 18009–18052. http://doi.org/10.3390/s141018009

Lee, J., Su, Y., & Shen, C. (2007). A Comparative Study of Wireless Protocols : IECON Proceedings (Industrial Electronics Conference), 46–51. http://doi.org/10.1109/IECON.2007.4460126

Levi, C. (2013). How Many Calories Do I Burn Bicycling? The Answer Is Here!. CoachLevi.com. Retrieved 12 August 2016, from http://coachlevi.com/health/calories-burned-bicycling/

Lie, D., & Sung, C. K. (2010). Synchronous brake analysis for a bicycle. Mechanism and Machine Theory. http://doi.org/10.1016/j.mechmachtheory.2009.11.006

Lima, R. R., & Torres, L. a. B. (2012). Performance Evaluation of Attitude Estimation Algorithms in the Design of an AHRS for Fixed Wing UAVs. 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium, (2), 255–260. http://doi.org/10.1109/SBR-LARS.2012.61

LM324-N | Operational Amplifiers | Amplifier | Description & parametrics. (2016). Ti.com. Retrieved 17 September 2016, from http://www.ti.com/product/LM324-N

Manual, I. U. (2016). FlexiForce® Sensors User Manual. Tekscan Inc, 1–15. Retrieved from https://www.tekscan.com/support/faqs/flexiforce-user-manual

mcxiaoke/android-volley. (2016). GitHub. Retrieved 25 June 2016, from https://github.com/mcxiaoke/android-volley

Mehndiratta, N., & Bedi, H. (2013). Design Issues for Routing Protocols in WSNs. International Journal of Application or Innovation in Engineering & Management (IJAIEM), 2(3), 312–320. Retrieved from http://www.ijaiem.org/Volume2Issue3/IJAIEM-2013-03-23-071.pdf

Miller, M. (2013). Wireless networking. Indianapolis, Ind.: Que.

Mohammadi, M. (2016). Analysis of low complexity uplink/downlink full-duplex wireless access with spatially random nodes. IET Communications, 10(14), 1777–1785. http://doi.org/10.1049/iet-com.2016.0237

Molloy, D. (2015). Exploring BeagleBone. Wiley.

Molloy, D. (2016). Exploring Raspberry Pi. Wiley.

Mozer, D. (2016). Bicycle History Timeline. Ibike.org. Retrieved 9 May 2016, from http://www.ibike.org/library/history-timeline.htm

MySQL :: MySQL Workbench Manual. (2016). Dev.mysql.com. Retrieved 16 July 2016, from http://dev.mysql.com/doc/workbench/en/

Oecd. (2012). Machine-to-Machine Communications: Connecting Billions of Devices. Cities, 192(192), 45. http://doi.org/http://dx.doi.org/10.1787/5k9gsh2gp043-en

Our Members | The ZigBee Alliance. (2016). Zigbee.org. Retrieved 25 May 2016, from http://www.zigbee.org/zigbeealliance/our-members/

Pandian, P. S., Bioengineering, D., Nagar, C. V. R., Safeer, K. P., Gupta, P., Shakunthala, D. T., … Padaki, V. C. (2008). Wireless Sensor Network for Wearable Physiological Monitoring. Journal of Networks, 3, 21–29. http://doi.org/10.4304/jnw.3.5.21-29

Pearson. (2011). Evolution of Cellular Technologies Chapter 1. Retrieved from http://cdn.ttgtmedia.com/searchTelecom/downloads/SearchTelecom_Fundamentals_of_LTE_Chapter_1.pdf

Pereira, V., & Sousa, T. (2004). Evolution of Mobile Communications: from 1G to 4G. 2nd International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks, (July).

Performance of Wireless Networks: WiFi - High Performance Browser Networking (O'Reilly). (2016). High Performance Browser Networking. Retrieved 24 August 2016, from https://hpbn.co/wifi/

Performance, H. (n.d.). MEMS Inertial Measurement Units for Complex Motion Capture and Processing.

PhilJay/MPAndroidChart. (2016). GitHub. Retrieved 25 June 2016, from https://github.com/PhilJay/MPAndroidChart

Pololu - MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carrier). (2016). Pololu.com. Retrieved 9 May 2016, from https://www.pololu.com/product/2468

Prayudi, I., & Kim, D. (2012). Design and implementation of IMU-based human arm motion capture system. 2012 IEEE International Conference on Mechatronics and Automation, ICMA 2012, 670–675. http://doi.org/10.1109/ICMA.2012.6283221

Radio Frequency Identification Tags - Semi Passive (Battery Assisted Backscatter) - Available Technologies - PNNL. (2016). Availabletechnologies.pnnl.gov. Retrieved 26 August 2016, from http://availabletechnologies.pnnl.gov/technology.asp?id=90

Ran, C., & Cheng, X. (2016). A Direct and Non-Singular UKF Approach Using Euler Angle Kinematics for Integrated Navigation Systems. Sensors, 16(9), 1415. http://doi.org/10.3390/s16091415

Sanches, R. (2015). Conheça os tipos e modelos de Bicicleta. Núcleo Bike. Retrieved 10 June 2016, from http://www.nucleobike.com.br/dicas/conheca-os-tipos-e-modelos-de-bicicleta/

Santos, R. & Perestrelo, L. (2015). BeagleBone For Dummies. Wiley.

Semiconductors, N. X. P. (2016). MFRC522 Standard performance MIFARE and NTAG frontend, (April).

Serial Communication - learn.sparkfun.com. (2016). Learn.sparkfun.com. Retrieved 17 September 2016, from https://learn.sparkfun.com/tutorials/serial-communication

STEYN, W. J. vdM., VAN NIEKERK, H., & JACOBS, W. M. G. (2014). CLASSIFICATION OF MOUNTAIN BIKE TRAILS USING VEHICLE-PAVEMENT INTERACTION PRINCIPLES. South African Journal for Research in Sport, Physical Education & Recreation (SAJR SPER), 36(1), 211–227. Retrieved from http://ezproxy.leedsbeckett.ac.uk/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=s3h&AN=95529862&site=eds-live&scope=site

Steyn, W. J., & Warnich, J. (2014). Comparison of Tyre Rolling Resistance for Different Mountain Bike Tyre Diameters and Surface Conditions, 36(2), 179–193.

Stočes, M., Vaněk, J., Masner, J., & Pavlík, J. (2016). Agris on-line Papers in Economics and Informatics Internet of Things (IoT) in Agriculture -Selected Aspects. *AGRIS on-Line Papers in Economics and Informatics*, *1*(1), 83–88. http://doi.org/10.7160/aol.2016.080108

Strava | Run and Cycling Tracking on the Social Network for Athletes. (2016). Strava.com. Retrieved 9 September 2016, from https://www.strava.com

Sundström, D., Bäckström, M., Carlsson, P., & Tinnsten, M. (2015). Optimal distribution of power output and braking for corners in road cycling, (2), 1–2.

Tavares, J., Velez, F. J., & Ferro, J. M. (2008). Application of Wireless Sensor Networks to Automobiles. Measurement Science Review, 8(3), 65–70. http://doi.org/10.2478/v10048-008-0017-8

Understanding Bike Frame Materials - REI Expert Advice. (2016). Rei.com. Retrieved 15 June 2016, from https://www.rei.com/learn/expert-advice/bike-frame-materials.html

Woodman, O. J. (2007). An Introduction to Inertial Navigation. University of Cambridge. http://doi.org/10.1017/S0373463300036341

Wynn, N. & Elton-Walters, J. (2016). The best cycling apps for iPhone and Android (video) - Cycling Weekly. Cycling Weekly. Retrieved 31 July 2016, from http://www.cyclingweekly.co.uk/news/product-news/best-cycling-apps-143222

XBee ZigBee Addressing. (2016). kb. Retrieved 18 June 2016, from http://knowledge.digi.com/articles/Knowledge_Base_Article/XBee-ZigBee-Addressing/?l=en_US&fs=Search&pn=1

XCTU - Next Gen Configuration Platform for XBee/RF Solutions - Digi International. (2016). Digi.com. Retrieved 18 July 2016, from http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

Yun Shield. (2016). Dragino.com. Retrieved 22 July 2016, from http://www.dragino.com/products/yunshield/item/86-yun-shield.html

Yuriyama, M., & Kushida, T. (2010). Sensor-cloud infrastructure physical sensor management with virtualized sensors on cloud computing. *Proceedings - 13th International Conference on Network-Based Information Systems, NBiS 2010*, 1–8. http://doi.org/10.1109/NBiS.2010.32

Zeng, D., Guo, S., & Cheng, Z. (2011). The Web of Things: A Survey (Invited Paper). Journal of Communications, 6(6), 424–438. http://doi.org/10.4304/jcm.6.6.424-438

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7–18. http://doi.org/10.1007/s13174-010-0007-6

Zhao Lin, Xia, L., Liu, F., & Cheng, Y. (2007). Application of UKF for MEMS IMUs and Fluxgate Sensors Based Attitude and Heading Reference System of Carriers, 0, 2278–2283.

ZigBee. (2016). Gta.ufrj.br. Retrieved 15 June 2016, from http://www.gta.ufrj.br/grad/10_1/zigbee/padrao.html

Appendix A

Article

This article has been accepted and presented in EPE 2016 9th edition October 20-22, 2016 IASI, Romania 2016 International Conference and Exposition on Electrical and Power Engineering.

EPE 2016
9th
editon
October 20-22, 2016
IASI, ROMANIA
2016 INTERNATIONAL CONFERENCE AND EXPOSITION ON ELECTRICAL AND POWER ENGINEERING
FACULTY OF ELECTRICAL ENGINEERING
IEEE
Romania Section
SETIS
IAȘI
1910

Organized by the Faculty of Electrical Engineering in Iasi and SETIS Association, the EPE Conference is confirmed as an important event in the community of Electrical Engineering. The conference started in 1999 and is being held on alternate years, with the intent of attracting a wide national an international audience from both the academic and industrial communities.

The conference is technically co-sponsored by IEEE Romanian Section and is included in the IEEE meetings database with record number #38374.

Performance Assessment for Mountain bike based on WSN and Cloud Technologies

Tiago Ribeiro
Telecommunications Institute
ISCTE-IUL
Lisboa, Portugal
Tiago_Ribeiro@iscte.pt

Octavian Postolache
Telecommunications Institute
ISCTE-IUL
Lisboa, Portugal
Octavian.adrian.postolache@iscte.pt

Pedro Passos
CIPER
FMH
Universidade de Lisboa, Portugal
ppassos@fmh.ulisboa.pt

Abstract—**The mountain bike is one of the most used equipment's in outdoor sports activities. The present work presents a distributed sensing system for cycling assessment to increase performance. Thus a wireless sensor network attached to the sport equipment provides to the athlete and the coach with performance values during practice. The sensors placed in biker equipment's behave as nodes of a WSN. This is possible with the developing of IoT-based systems in sports, the tracking and monitoring of athletes in his activities has an important role on his formation as bikers and helps to increase performance, through the analyze of each session. The implemented system performs acquisition, processing and transmission, of data using a ZigBee wireless networks that provide also machine-to-machine communication and data storage in a server located in the cloud. As in many cycling applications use the phone as a module to get the values, this work will be a little different making use of phone/tablet to consult information. The information stored on the cloud server will be accessed through a mobile application that analyses and correlates all metrics calculated using the training data obtained during practice. Additional information regarding the health status may be also considered. Therefore, the system permits that athletes perform an unlimited number of trainings, these can be accessed at any time through the mobile application. Making possible to save a history of the evolution of each athlete the system permits to perform appropriate comparisons between different training sessions and different athletes performances.**

Keywords— machine-to-machine; bicycle; cloud; wireless sensor network

I. Introduction

The history of bicycle remote us to the year of 1418 where the engineer Giovanni Fontana built a four wheeled "bike" with rope connected by gears. Only 400 years later, in response to the starvation and the slaughtering of horses Baron Von Drais built the first two wheeled bike that has a cord connected to the back wheel. These velocipedes were made entirely of wood and were propelled by pushing off the feet [1]. The years passed and the innovations were many, as the pedals, brakes, suspension, and the lighter and comfortable materials that we see today in our bikes. Nowadays bicycles can be classified in four types: urban bike, BMX, road bikes and mountain bikes [2].

For the conception of this work will be used a mountain bike, this bicycles are typically used in single tracks where the terrain can be unpaved. On these tracks commonly we could encounter rocks, loose gravel, roots and steep grades inclines and declines. Under these conditions the biker will need to get a functional interaction with his "machine", in order to achieve a balance between speed and safety. The mountain bike is produced to handle with this type of tracks, normally the material that is widely used is aluminum (offers a lighter, stiffer and efficient ride) [3]. To control the bicycle along the track the biker needs to use the mechanisms with his hands (e.g., brakes) and feet (i.e., pedals), and the positioning of his body is also fundamental. A functional position will permit the biker make rapid changes of direction and get the desirable speed. The biker position could be essential to deal with some external factors such as the slope of the track and its obstacles. It is the interactive behavior of the bicycle and biker which wanted to acquire, dynamic and kinematic data through a Wireless Sensor Network was designed and implemented. These WSN permits to get values to some of the following variables: i) braking intensity and frequency (both hands); ii) pedal strength (both feet); iii) position of the bicycle in the three plans of motion (x, y and z); iv) position of the biker in the three plans of motion. With this set of variables, it was possible to calculate some important aspects to describe the bicycle ~ biker interaction.

For the athletes the training is the most important way to achieve a better performance whether professional or recreational level. In this era of the Internet of Things (IoT), this term IoT refers to the connection of physical objects to the Internet and that can communicate through wireless or wired connections and interact with each other and collaborate with other things or objects, to create new services in order to achieve common objectives [4]. Where vehicles, buildings, health systems [5]-[6] and a lot of another's things are characterized by sensors, software and network connectivity that enable the data collection and data exchange remotely across existing network. Creating a large pool of resources inter-liked through a dynamic network of networks. In 2020 it is estimated that will be around 50 billion devices connected to the internet. One sustainable IoT platform relies on three pillars (renewable energy, connectivity, and collaboration) [7]. This system will allow the users to obtain data from his/her own training session. The data capture with this sensors network provides the users with information which can be used to adapt the training methods and improve the performance. The connectivity and collaboration are a fundamentals pillars in this work. Because an autonomous exchange of information through the WSN will permit feed the database and by consequence a coach and a biker will collaborate in the analyze of a determined training session.

II. Related Work

Cycling it's a complex physical activity that involves a set of movements. The bicycle with a human comprises a human-vehicle with a dynamic behavior. In contrast with an automobile or aircraft, the pilot of a bicycle comprises 80%~90% of the overall system mass and thus the motion of the driver rider is not negligible. These movements can be analyzed using different techniques, such as instrumented bicycle or motion capture techniques [8]. These two techniques highlight the complexity of the study. The first one it's related with the use of sensors on the bike or in the biker's equipment like pressure sensors and GPS. The second technique could be related with the movement of the biker related with the bicycle, that is, analyze the influence of the position of the upper body and the legs while the bicycle is on movement. These two techniques allows the creation of models that describe the movement, the points of pressure [8] and speed that a biker could be doing while training on certain track. These data could be used to provide a track-specific performance report.

Machine-to-Machine (M2M) communication [9] is an emerging technology that envisions the interconnection of machines without the need of human interventions. The main concept lies in seamlessly connecting an autonomous and self-organizing network of M2M-capable devices to a remote client, through wired or wireless network. The M2M is a combination of various heterogeneous electronic, communication, and software technologies. An intelligent software application is usually employed at the remote client to process the collect data and provide the end user with a set of smart services and a practical interface. The above challenges stress the imperative need for standardization of M2M communications [10]. The ability to connect new devices to the network lead to the evolution of the Internet of Things (IoT) [11]. In this scenario, Cloud Computing [12] can be seen as a scalable infrastructure [13] that supports computing power, storage and software services.

With the advances in wireless sensor networks (WSN) the use of these networks for collect and interpret data in real-time was facilitated. In the past they were typically used wired sensor networks but have always been expensive, due to installation and maintenance costs. But the large amount of research projects in this area allows for the existence of tiny hardware (and more accurate) devices with reduced cost/size. One of the areas on the rise is the monitoring in automobiles [14] and the physiological monitoring [15]. These advances include the development of communication standards such as IEEE 802.15.4/ ZigBee. This sensor network support small power consumption and node expansion compared to other networks standards for WSN [16].

Joining M2M communication, WSN and Cloud Computing. This architecture allows to stored and processed sensor data in a more accessible form, available timely, and cost-effective. This concept can be called sensor-cloud, it allows easier integration with new mobile devices like tablets or smartphones through customized mobile applications that collect data from cloud and process all sensor data.

The growth of information technology and the easy access to it, currently thousands of application are released to the market per day, some of these apps are related with the use of bicycles and cycling, but most only collect information via mobile phone modules. The difference through this work is to use WSN that can in real time detect all interactions that a biker has with his bike and through M2M communication send this data to the cloud, to be analyzed and processed in the mobile application.

III. Overview

The architecture of the system has three blocks *Fig. 1*. At first block is represented the bicycle and the biker, so it's easy to assume that wireless sensor network was placed, on the

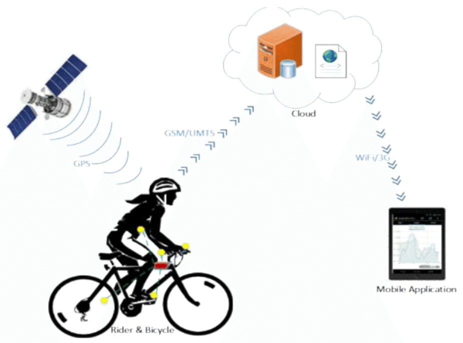


Figure 1 - System Architecture

sports equipment (i.e., gloves, shoes, chest trap and bicycle frame). Each of the end nodes has a microcontroller *Fig. 2* that will read the behavior and the interaction of the biker with his bicycle. These nodes will make the acquisition and processing of primary data and then send it to coordinator, that stores the information on a local database and send it to the cloud. The second block is the server (cloud), which receives the information and makes it available to be accessed later. The third block, represents a mobile application that accesses the information in the cloud, interprets and correlates all biker information and allows data visualization with a friendly graphical user interface.

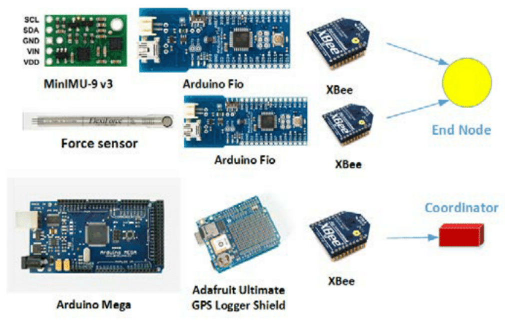


Figure 1 – WSN End Node and Coordinator components

IV. Hardware

The hardware involved in the WSN is responsible for the acquisition, processing and sending of data to the server. So, in the WSN are the force sensors, inertial measurement boards and microcontrollers. The microcontrollers interpret and read the information from sensors and inertial measurement boards.

A. *Sensors and conditioning circuits*

Force sensors from FlexiForce [17], act as a force sensing resistor in an electric circuit, when no pressure is applied to the sensor the resistance is very high, otherwise the resistance decrease. In the project these sensors are in the biker shoes and gloves, these ten sensors acquire the interaction described. The following conditioning circuit is used:

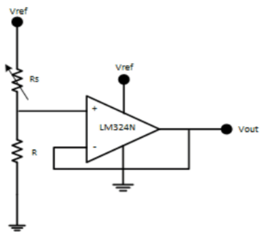


Figure 3 - Conditioning circuit for force sensors

The equations of the output voltage are represented as follows:

$$Vout = \frac{R}{Rs+R} \times Vref \qquad (1)$$

where, the parameters in equation (1) are defined as follows:

Vout: is the output voltage [V];

Rs: is the variable resistance, force sensor [Ω]

R: is the reference resistor [Ω];

Vref: is the reference voltage [V];

B. *Inertial measurement unit*

IMU, is an inertial measurement unit that packs an L3GD20H 3-axis gyro and an LSM303D 3-axis accelerometer and 3-axis magnetometer onto a tiny 0.8x0.5 board. This board allows to calculate some angles between the bicycle and the rider as well as the direction of the movement.

The nine independent rotation, acceleration and magnetic readings (known as 9 Degrees of Freedom) provide all the data needed to make an attitude and heading reference system (AHRS). These sensors, combined with a built-in processor, create an inertial sensor system fully cable of measuring the attitude of objects in 3D space. The accelerometers measure proper acceleration – the rate at which the velocity of an object is changing. They measure the static (gravity) or dynamic (motion vibration) acceleration forces of a given object. The ideal accelerometer in AHRS provides a long term stability, low vibration error and reliability. Magnetometers are used in AHRS to measure the direction of the magnetic field at a point in space. In case of gyroscope the AHRS demand very precise sensors; the gyros are used as the primary source of orientation information. The quality of these devices has big impacts in overall performance of the inertial sensor system [18]. Though this technology provides good accuracy and reliability, it is not conducive to a MEMS-based AHRS due to its larger size and greater power requirements.

This sensor is used in the chest of the rider to measure the upper body motion and in the bicycle frame to record the oscillations on the three plans of motion [19]. Each of the three sensors acts as a slave device on the same I^{2}C bus [20].

The orientation of the bicycle/rider is often described by three consecutive rotations, whose order is important. The angular rotations are called the Euler angles. The orientation of the body frame with respect to the fixed earth frame was determined in the following manner.

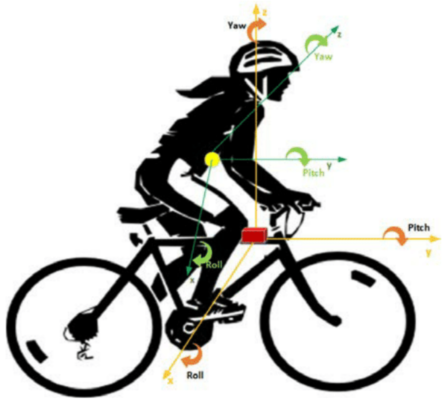


Figure 4 - Representation IMU axis

C. *Embedded Systems*

The Arduino Fio is a microcontroller board based on the ATmega328P runs at 3.3V and 8MHz. It has connections for a Lithium Polymer battery. An Xbee socket is available on the bottom of the board. The Arduino is intended for wireless applications [21]. On this project exists five of these boards, they work as end nodes and are responsible for the acquisition of sensor data to the network. Another microcontroller is an Arduino Mega it as based on the ATmega1280. It contains everything to support the microcontroller. The Mega is compatible with most shields in the market [22]. He works as a coordinator and could find it on the frame of the bicycle. All the communication between the WSN is done through a ZigBee network 802.15.4 [23] with modules coupled to each microcontroller.

As can be seen in the *Fig. 5*, the network has a star topology, in which the Arduino Mega *Fig. 2* (center node) as the coordinator is placed in bicycle frame.

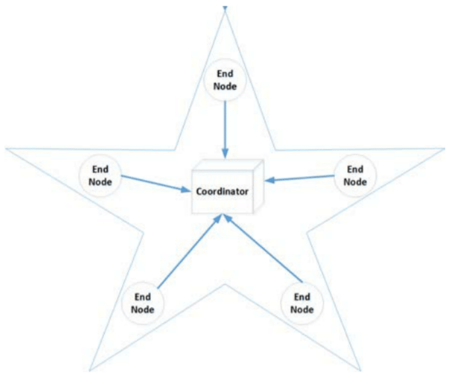


Figure 5 - WSN Topology

The coordinator node it also has a GPS shield that registers all movements and has the capability to locate our exact position within a few meters, as well as other variables like speed, altitude and angles.

V. System Software

It was embedded software in the microcontrollers, which is responsible for the interpretation of the data that the sensors capture. At this moment the system is in an early stage, only the local database is function properly.

All the microcontrollers are programmed using C language and also some Arduino libraries to deal with the shields and the sensor boards.

In our AHRS system is used the Direction Cosine Matrix Algorithm (DCM), he calculates the orientation of a rigid body, in respect to the rotation of the earth by using rotation matrices. Rotation matrices have the advantage of being a natural fit to control and navigation. A rotation matrix describes the orientation of one coordinate system with respect to another.

$$R = \begin{array}{c|ccc|c} & xb & yb & zb & \\ & & & & xe \\ & & & & ye \\ & & & & ze \end{array} \qquad R^{-1} = \begin{array}{c|ccc|c} & xe & ye & ze & \\ & & & & xb \\ & & & & yb \\ & & & & zb \end{array}$$

The columns of the matrix are the unit vectors in one system as seen in another system. A vector in one system can be transformed into the other system by multiplying it by the rotation matrix. The transformation in the reverse direction is accomplished with the inverse of the rotation matrix, which turns out to be equal to its transpose. Certain types of vectors (directions, velocity accelerations and translations) can be transformed between rotated reference frames with 3x3 matrix [24].

A_P = a vector A measured in the frame of reference of the plane;

A_G = a vector A measured in the frame of reference of the ground;

$$R = \begin{vmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{vmatrix} = rotation\ matrix$$

$$A_G = RA_P \tag{2}$$

The rotation matrices are related to the Euler angles, which describe the three consecutive rotations needed to describe the orientation. So the relation between the direction cosine matrix and Euler angles is:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} cos\ \phi & sin\ \phi & 0 \\ sin\ \phi & cos\ \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X'' \\ Y'' \\ Z'' \end{bmatrix} \tag{5}$$

The coordination system shown in *Fig. 4* is typically used in flight simulators with the origin located at the aircraft centroid, with y-axis pointing forward, the x-axis off the right and the z-axis pointing up. Note that orders matters, (3)->(4)->(5). Combine all three transformations (roll, pitch and yaw).

$$R = \begin{bmatrix} cos\theta cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ cos\theta sin\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & cos\psi sin\theta sin\phi - sin\psi cos\phi \\ -sin\theta & sin\psi cos\theta & cos\psi cos\theta \end{bmatrix} \tag{6}$$

Equation (2) and (6) express how to rotate a vector measured in the frame of reference of the plane to the frame of reference of the ground. Equation (2) is expressed in terms of direction cosines. Equation (6) is expressed in terms of Euler angles [25]. The following equations permit to calculate the Pitch, Roll and Yaw:

$$\begin{gathered} Pitch \rightarrow \psi = -\operatorname{asin}(r_{zx}) \\ Roll \rightarrow \theta = atan2(r_{zy}, r_{zz}) \\ Yaw \rightarrow \phi = atan2(r_{yx}, r_{xx}) \end{gathered} \tag{7}$$

For a visual representation of DCM algorithm *Fig. 6*.

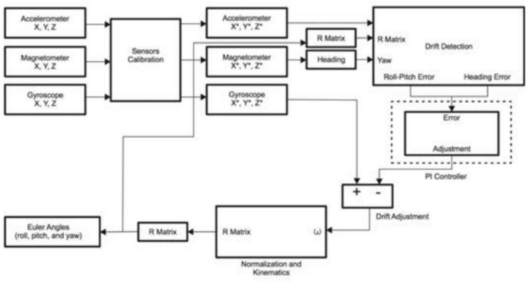


Figure 6 - Direct Cosine Matrix Algorithm Overview

Gyroscopes readings have different offsets depending on which direction the gyroscope is facing; when these readings are integrated over time it causes the integral result to drift. Before apply equation (7), it is essential to correct this problem (drift), and the process of enforcing the orthogonally conditions "renormalization" is used. First compute the dot product of the X and Y rows of the matrix, which is supposed to be zero, so the result is a measure of how much the X and Y rows are rotating toward each other:

$$X = \begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} \; Y = \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix}$$

$$error = X.Y = X^T Y = \begin{bmatrix} r_{xx} r_{xy} r_{xz} \end{bmatrix} \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} \qquad (8)$$

We apportion half of the error each to the X and Y rows, and approximately rotate the X and Y rows in the opposite direction by cross coupling:

$$X = \begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} = X_{orthogonal} = X - \frac{error}{2} Y$$

$$Y = \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} = Y_{orthogonal} = Y - \frac{error}{2} X \qquad (9)$$

Orthogonal error is greatly reduced by substituting equation (9) into (8), keeping in mind that the magnitude of each row and column of the R matrix is approximately equal to one. The next step is to adjust the Z row of the matrix to be orthogonal to X and Y row. The way to do that is to simply set the Z row to be the cross product of the X and Y rows:

$$\begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} = Z_{orthogonal} = X_{orthogonal} \times Y_{orthogonal} \qquad (10)$$

The last step in the renormalization process is to scale the rows of the R matrix to assure that each has a magnitude equal to one. The resulting magnitude adjustment equations for the row vectors are:

$$X_{normalized} = \frac{1}{2}(3 - X_{orthogonal} \cdot X_{orthogonal}) X_{orthogonal})$$

$$Y_{normalized} = \frac{1}{2}(3 - Y_{orthogonal} \cdot Y_{orthogonal}) Y_{orthogonal})$$

$$Z_{normalized} = \frac{1}{2}(3 - Z_{orthogonal} \cdot Z_{orthogonal}) Z_{orthogonal})$$

(11)

These equations state that to adjust the magnitude of each row vector to one, is necessary to subtract the dot product of the vector with itself (the square of the magnitude), subtract from three, multiply by ½, and multiply each element of the vector by the result.

The accelerometer is not affected by drift; therefore, it can be used as an orientation reference in the X and Y axis of the rigid body to compensate the roll-pitch error (gyro's offset error). The magnetometer's readings are used to calculate the heading of the rigid body; this one helps to compensate yaw error. The heading of the system used as the reference vector in the Z axis (yaw error), in addition to roll-pitch error calculated by the accelerometer, it allows the system to calculate the rotations correction matrix.

VI. Preliminary Results And Discussion

The tests are made step by step, the serial of Arduino it's an important thing to analyze and helpful tool. At this early stage all the information goes to the local database through the serial of Arduino. The tests that were made provide data that was graphically analyzed. In the local database are stored the following variables: time, date, longitude, latitude, velocity, angle, and altitude; plus ten values of the force sensors, pitchReceived, rollReceived, yawReceived, pitch, roll, and yaw. It's important to tell that the variables (pitchReceived, rollReceived, yawReceived) correspond to the measurement of movements of the biker, and the variables (pitch, roll, and yaw) correspond to the movements of the bicycle.

Analyzing these six variables from the movements that the biker produces together with his bicycle can be detected and related (e.g., in a curve compare both rolls and see how can the movement of biker could influence its trajectory).

The magnitude and the frequency of brake on each hand, the path that runs through the GPS and the measure of pitch, roll and yaw angles, are examples of some tests already done. In the next graphics are demonstrate some of the possible tests that can be performed in WSN.

Some considerations are necessary to understand first, the range of the Euler angles are presented in the table 1.

Angles	*Min. value*	*Máx. value*
Roll	-180º	180º
Pitch	-90º	90º
Yaw	-180	180

Table 1 - Range of Euler angles

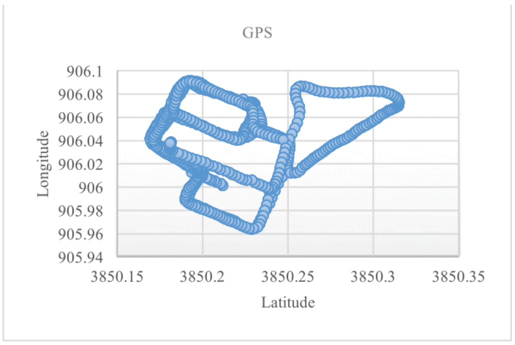


Figure 7 - GPS Tracking, Latitude and Longitude

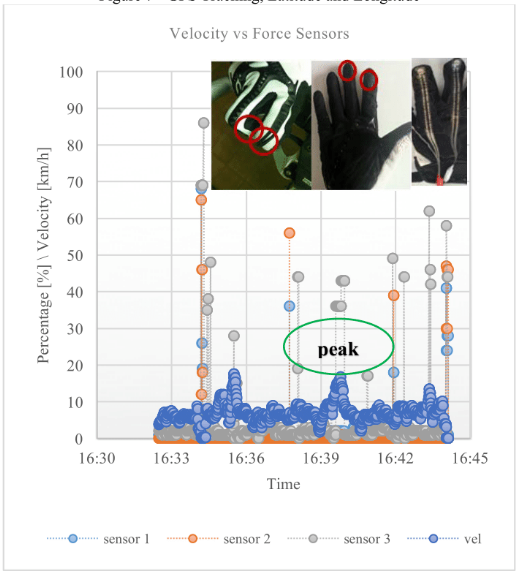


Figure 8 - Velocity vs. Force Applied

In this figure can be analyzed the relation between the velocity and the magnitude of the pressures on the brakes. When exist a velocity peak and the biker press the brake the velocity reduces. The sensors are inside of gloves on *Fig. 8* we can measure the pressure that the biker do to apply the force on brake.

Lat. (start)	***Long. (start)***	***Lat. (end)***	***Long. (end)***	***Duration [s]***	***Brake [%]***
3850.2256	906.073	3850.2244	906.0719	00:00:06	72.19%
3850.2539	906.0508	3850.2273	905.9646	00:00:20	38.98%

TABLE 2 - POSITION(START/END), DURATION AND BRAKE INTENSITY

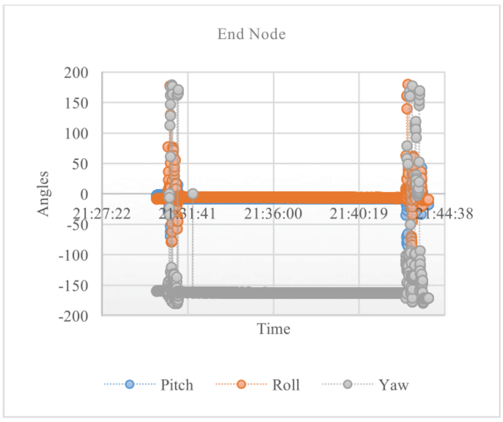


Figure 9 - Angles IMU End Node

Considering the *Fig.10, Table 1* and *Fig.4* it's possible to observe that numbers 1 and 2 are representing positive movements of bicycle, so pitch is point upwards and the roll is presenting an inclination to the right side, relative to the axis of the earth. In case of numbers 3 and 4 is the opposite (pointing down and inclination to the left). Looking to number 5 is representing the yaw values, so values very close to zero means that the bicycle is pointing to North.

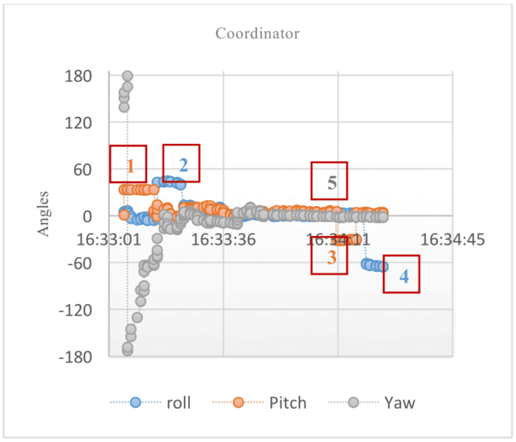


Figure 10 – Angles IMU Coordinator

VII. CONCLUSIONS AND FUTURE WORK

This paper presents an exploratory approach of this sensors network. Internet of Things applications are pushing the development of new platforms that support high level of connectivity between smart devices and also between people and smart devices, and provide new services for the peoples according with their goals. The results at an early stage they are quite encouraging. As described above only local database is running at the moment, so now the connection to the cloud is being developed. For this connection we are using the Yun

Shield [26] this shield is one of the most powerful shields for Arduino boards and he's designed to solve the internet connectivity issue [27]. After this problem is solved, it's time to construct the database in the cloud and create the application to analyze all information stored. Considering that the Android operating system allows data storage, connectivity, web browser, media support, development environment, we propose to develop an application that will come in support to any biker who is trying to achieve a better performance. With the help of such application the bikers and coaches can monitor measured values of force exerted on (brakes and pedals), tracking, velocity, angles between bicycle and biker, etc. The application offers the possibility of each training session to be accessed over time, creating a historical of all already training sessions. All this principles of Android operating system combined with IoT, such as data storage, connectivity, media support will be possible to create a sport application. The innovation on this work is presented on the fact that information is collect through a wireless sensor network. Making use of the phone/tablet just to observe the data collected from each training session.

REFERENCES

[1] D. Mozer. "Chronology of the Growth of Bicycling and the Development of Bicycle Technology". [Online]. Available: http://www.ibike.org/library/history-timeline.htm. [Accessed: 9-May-2016].

[2] R. Snches. "Conheça os tipos e modelos de Bicicleta". [Online]. Available: http://www.nucleobike.com.br/dicas/conheca-os-tipos-e-modelos-de-bicicleta/. [Accessed: 8-May-2016].

[3] B. Puliti. "Types of Mountain Bike Frame Materials". [Online]. Available: http://mountainbike.about.com/od/technologyinnovations/a/Types-Of-Mountain-Bike-Frame-Materials.htm. [Accessed: 9-May-2016].

[4] A. Gabbai. (2015, Jan.). "Kevin Ashton describes 'the Internet of Things': The innovator weighs in on what human life will be like a century from now. *Smithsonian Magazine*, USA. [Online]. Available: http://www.smithsonianmag.com/innovation/kevin-ashton-describesthe-internet-of-things-180953749/?no-istD

[5] A. Antonovici, I. Chiuchisan, O. Geman, and A. Tomegea, "Acquisition and management of biomedical data using Internet of Things concepts," *Fundam. Electr. Eng. (ISFEE), 2014 Int. Symp.*, pp. 1–4, 2014.I. Chiuchisan, H. Costin, and O. Geman, "Adopting the Internet of Things Technologies in Health Care Systems," *Int. Conf. Expo. Electr. Power Eng.*, no. Epe 2014, pp. 532–535, 2014.

[6] [1] C. F. Pasluosta, H. Gassner, J. Winkler, J. Klucken, and B. M. Eskofier, "An emerging era in the management of Parkinson's disease: Wearable technologies and the internet of things," *IEEE J. Biomed. Heal. Informatics*, vol. 19, no. 6, pp. 1873–1881, 2015.

[7] S.B.L. in U. of California "Instrumented Bicycle – Sports Biomechanics Lab." [Online]. Available: http://biosport.ucdavis.edu/research-projects/bicycle/instrumented-bicycle. [Accessed: 5-Jan-2016].

[8] D. Lie, C.-K. Sung. "Synchronous brake analysis for a bicycle," Mech. Mach. Theory, vol. 45, no. 4, pp. 543-554, Apr. 2010.

[9] E. Kartsakli, A. Lalos, A. Antonopoulos, S. Tennina, M. Renzo, L. Alonso, and C. Verikoukis, "A Survey on M2M Systems for mHealth: A Wireless Communications Perspective," *Sensors*, vol. 14, no. 10, pp. 18009–18052, 2014.

[10] S. K. Tan, M. Sooriyabandara, and Z. Fan, "M2M Communications in the Smart Grid: Applications, Standards, Enabling Technologies, and Research Challenges," *Int. J. Digit. Multimed. Broadcast.*, vol. 2011, pp. 1–8, 2011.

[11] D. Zeng, S. Guo, and Z. Cheng, "The Web of Things: A Survey (Invited Paper)," *J. Commun.*, vol. 6, no. 6, pp. 424–438, 2011.

[12] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010.

[13] C. N. Höfer and G. Karagiannis, "Cloud computing services: Taxonomy and comparison," *J. Internet Serv. Appl.*, vol. 2, no. 2, pp. 81–94, 2011.

[14] J. Tavares, F. J. Velez, and J. M. Ferro, "Application of Wireless Sensor Networks to Automobiles," *Meas. Sci. Rev.*, vol. 8, no. 3, pp. 65–70, 2008.

[15] P. S. Pandian, D. Bioengineering, C. V. R. Nagar, K. P. Safeer, P. Gupta, D. T. Shakunthala, B. S. Sundersheshu, and V. C. Padaki, "Wireless Sensor Network for Wearable Physiological Monitoring," *J. Networks*, vol. 3, pp. 21–29, 2008.

[16] C. F. García-hernández, P. H. Ibargüengoytia-gonzález, J. García-hernández, and J. a Pérez-díaz, "Wireless Sensor Networks and Applications : a Survey," *J. Comput. Sci.*, vol. 7, no. 3, pp. 264–273, 2007.

[17] Tekscan, "FlexiForce® Force Sensors | Single Button Force Sensing Resistor | Tekscan." [Online]. Available: http://www.tekscan.com/flexible-force-sensors#third. [Accessed: 20-Apr-2016]

[18] C. Honglong, X. Liang, Q. Wei, Y. Guangmin, and Y. Weizheng, "An integrated MEMS gyroscope array with higher accuracy output," *Sensors*, vol. 8, no. 4, pp. 2886–2899, 2008.

[19] I. Prayudi and D. Kim, "Design and implementation of IMU-based human arm motion capture system," 2012 IEEE Int. Conf. Mechatronics Autom. ICMA 2012, pp. 670–675, 2012.

[20] Pololu. "MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carrier)," [Online]. Available: https://www.pololu.com/product/2468. [Accessed: 13-Feb-2016].

[21] Arduino, "Arduino Fio," 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardFio. [Accessed: 4-Jan-2016].

[22] Arduino, "Arduino Mega," 2016. [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardMega. [Accessed: 3-Jan-2016].

[23] SparkFun Electronics, "Xbee Buying Guide", 2015. [Online]. Available: https://www.sparkfun.com/pages/xbee_guide. [Accessed: 20-Dec-2015].

[24] R. R. Lima and L. a. B. Torres, "Performance Evaluation of Attitude Estimation Algorithms in the Design of an AHRS for Fixed Wing UAVs," *2012 Brazilian Robot. Symp. Lat. Am. Robot. Symp.*, no. 2, pp. 255–260, 2012.

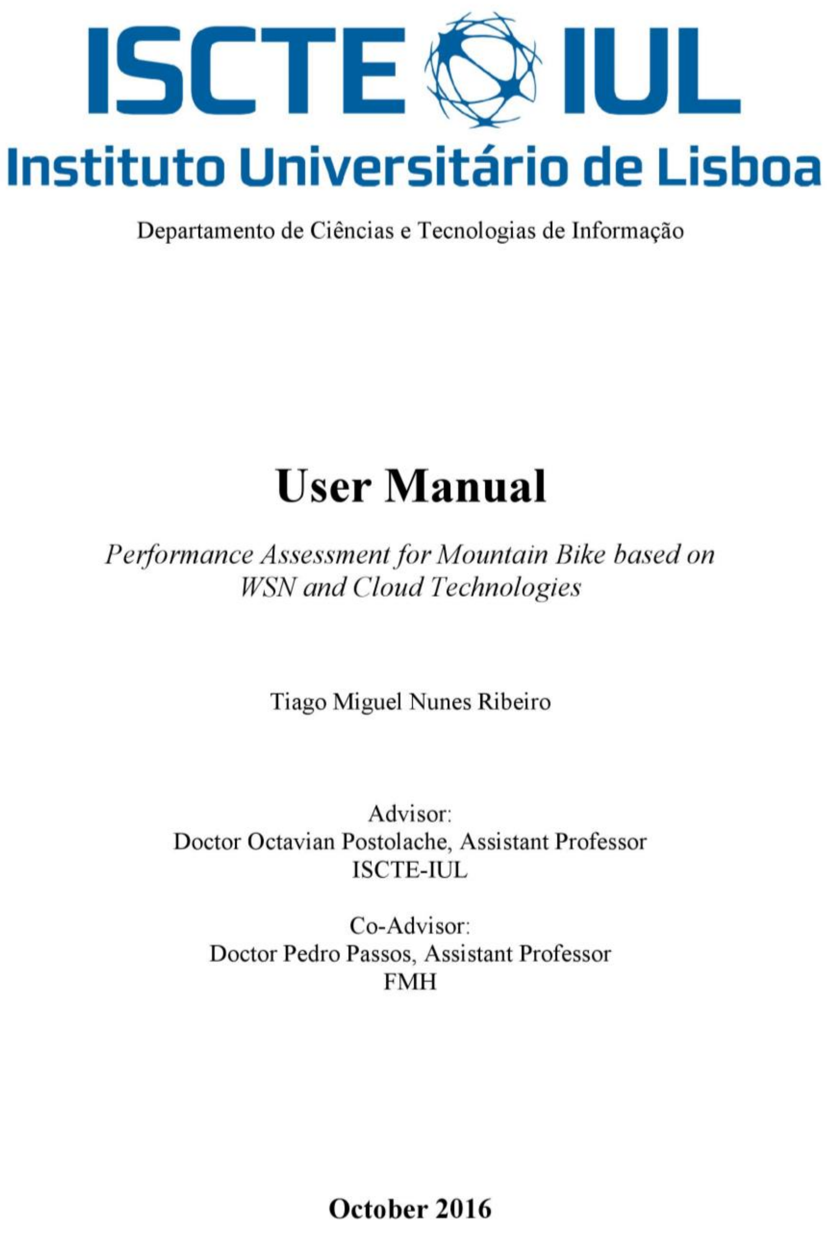
[25] Zhao Lin, L. Xia, F. Liu, and Y. Cheng, "Application of UKF for MEMS IMUs and Fluxgate Sensors Based Attitude and Heading Reference System of Carriers," vol. 0, pp. 2278–2283, 2007.

[26] Dragino, "Yun Shield," 2016. [Online]. Available: http://www.dragino.com/products/yundhield/item/86-yun-shield.html. [Accessed:25-Apr-2016].

[27] M. Schwartz, *Internet of Things with the Arduino Yún*. PACKT, 2014.

Appendix B

User Manual



ISCTE IUL
Instituto Universitário de Lisboa

Departamento de Ciências e Tecnologias de Informação

User Manual

Performance Assessment for Mountain Bike based on WSN and Cloud Technologies

Tiago Miguel Nunes Ribeiro

Advisor:
Doctor Octavian Postolache, Assistant Professor
ISCTE-IUL

Co-Advisor:
Doctor Pedro Passos, Assistant Professor
FMH

October 2016

Figures

Contents

User Manual

There are two kinds of users in this application bikers and coaches. The application has the possibility to handle with both with minor differences.

1. Coach Instructions

Looking to the user "coach", firstly every coach need to ask for registration to have access to the application. At the first time in the application the coach will encounter the follow mockups.

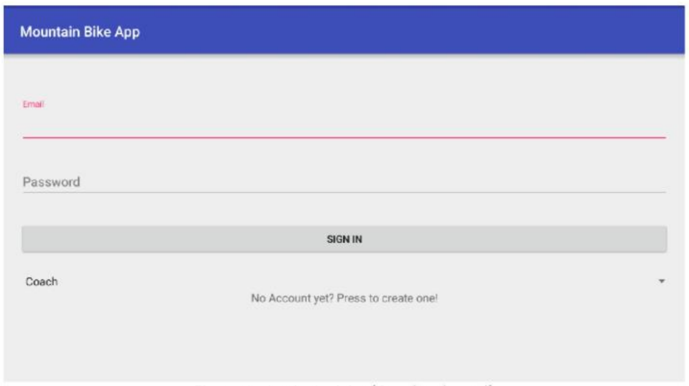


Figure 1 - Login Activity (Coach selected)

Here the coach need to insert the credentials which was provided by the admin of the database. After that the application will verify if the user is registered in case of success the coach will login in the application fig.2. In the case of failure will present a message and the window will not change.

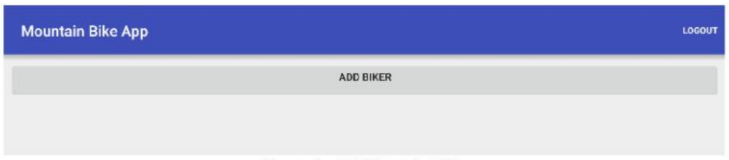


Figure 2 - ListBikers Activity

At first interaction with the application, the coach doesn't have any biker register so he can't see any workout performed, so the first thing to do is register one biker. To do that the coach must press the button "ADD BIKER" fig.3.

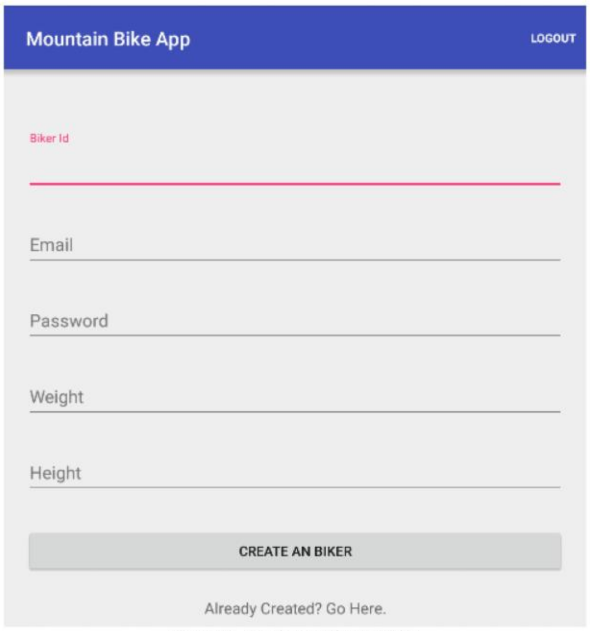


Figure 3 - Create an Biker Activity

Only coaches can register bikers, one biker can't register in the database by himself. This is the required fields to register one biker. In the field Biker Id the coach must insert a ID register in the SD card on the coordinator, at the moment the coordinator support twelve bikers.

Let's see a practical example, one coach wants to register one biker, first he need to know the tag Id like we can see in table 1. So the coach has the tag Id, he goes to the SD Card and write the tag Id on the text file "authids.txt". The coach can find the SD card in the coordinator like is presented in fig.4. In the file the coach only need to write the tag Id number but must remember that the Id is very important, because is the Id that he will use to register one biker.

Id	Tag Id
0	27654
1	1821
2	5467
3	28698
4	17665
5	07833
6	21347
7	8765
8	0765
9	16332
10	7643
11	09432

Table 1 - Ids and Tags Id

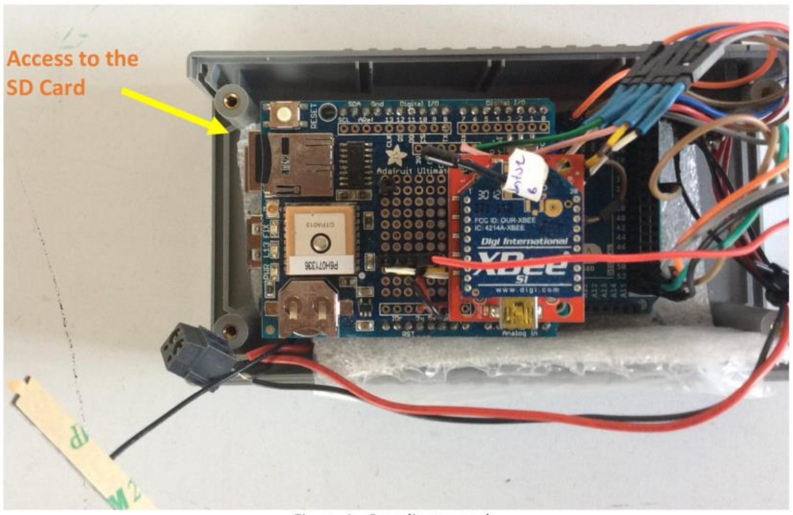


Figure 4 - Coordinator node

So the coach goes to SD Card and insert in file "authids.txt" the tag from the card he wants to register. The tag number is "16332", after is insertion he can see that the number is in the position nine so the number to insert in the application is the number nine.

After all data is inserted the coach press the button "CREATE AN BIKER", and biker will be created, then he need to press the link "Already Created? Go Here".

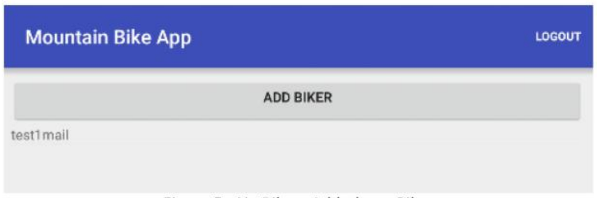
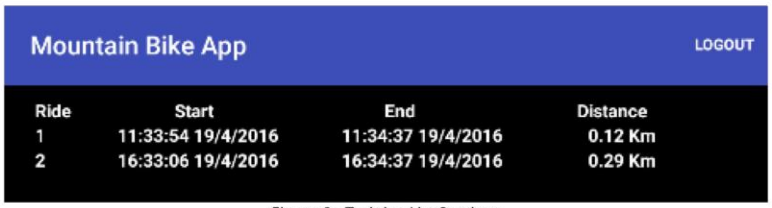


Figure 5 - ListBikers Added one Biker

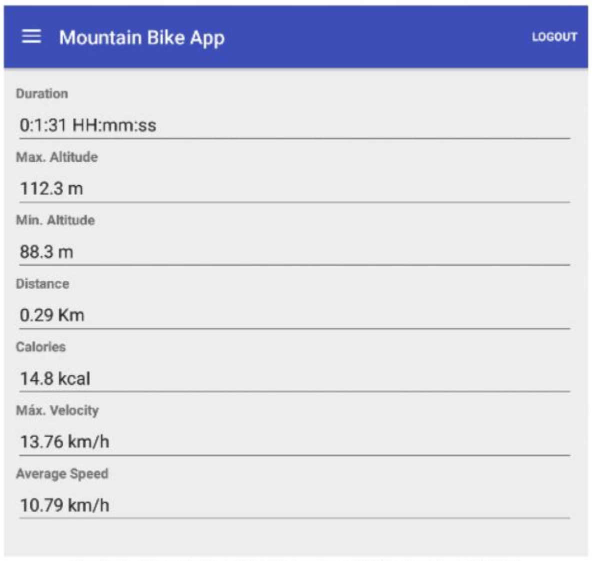
Pressing the link, sends the coach to this mockup, here is presented the list of the bikers already registered. Selecting the biker in this case "test1mail" will send the coach to another activity. Well we will assume that the biker register by the coach, already have access to the mountain bike and perform his trainings.



Ride	Start	End	Distance
1	11:33:54 19/4/2016	11:34:37 19/4/2016	0.12 Km
2	16:33:06 19/4/2016	16:34:37 19/4/2016	0.29 Km

Figure 6 - Training List Sessions

In this activity is shown another list, but now is presented the number of trainings already made for the registered biker. Choosing one training will send the coach to another mockup where is presented a simple report of the training aspects.



Duration	0:1:31 HH:mm:ss
Max. Altitude	112.3 m
Min. Altitude	88.3 m
Distance	0.29 Km
Calories	14.8 kcal
Máx. Velocity	13.76 km/h
Average Speed	10.79 km/h

Figure 7 - Report from Training pressed (Coach point of View)

To consult another aspects of the training session the coach need to press the icon in the upper left corner. Pressing the icon will present a Navigation Drawer with all fragments possible to analyze all aspects of training. In the graphs is possible to get every point value, the coach only need to press with is finger.

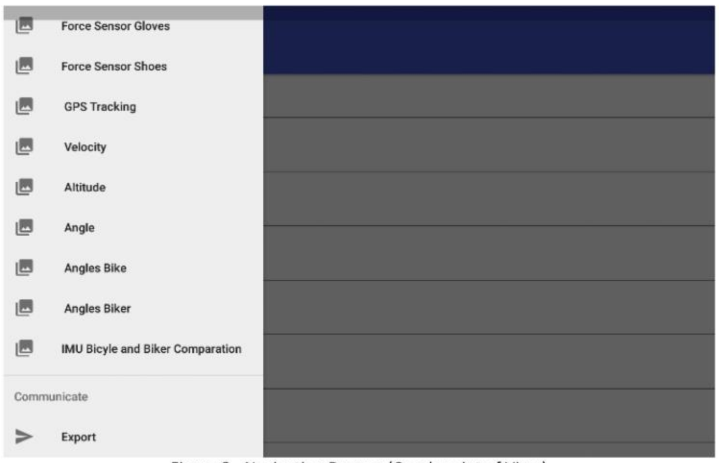
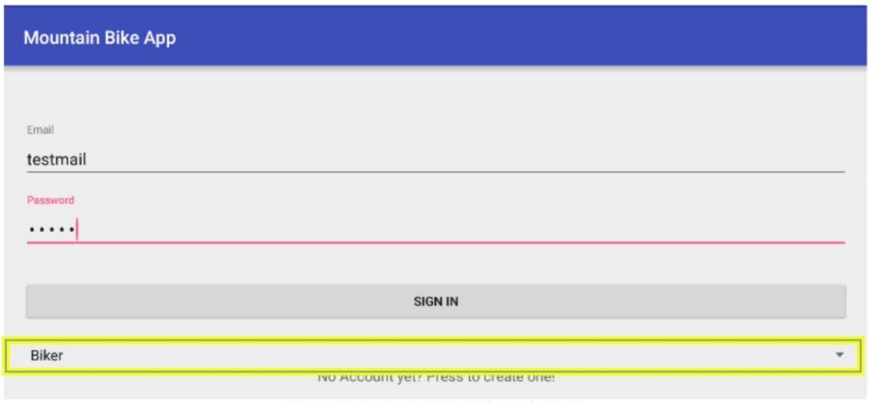


Figure 8 - Navigation Drawer (Coach point of View)

All this features are the same if you enter as coach or a biker, the main difference in a coach and a biker is that a coach can register another bikers and can see all training sessions of all different biker's register associated to him and the biker only can see is own trainings.

2. Biker Instructions

In the case of the biker the things are a little simpler, this is, the biker only has access is own trainings. The biker need to ask the credentials to the coach. The credentials are the same that the coach define when register the biker.


Mountain Bike App

Email
testmail

Password
•••••

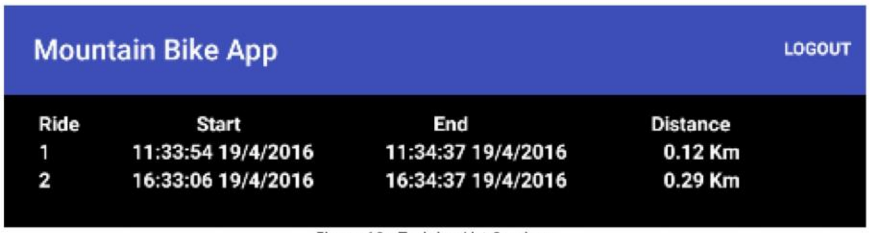
SIGN IN

Biker

No Account yet? Press to create one!

Figure 9 - Login Activity (biker selected)

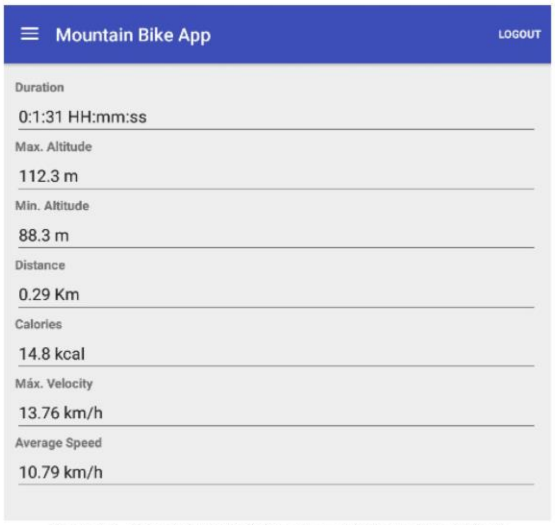
To enter as a biker, the user need to choose the option biker. Like is shown in fig.9. After that the user enter is credentials and if is register in the database the next mockup will be the training list.


Mountain Bike App LOGOUT

Ride	Start	End	Distance
1	11:33:54 19/4/2016	11:34:37 19/4/2016	0.12 Km
2	16:33:06 19/4/2016	16:34:37 19/4/2016	0.29 Km

Figure 10 - Training List Sessions

This is the activity shown to the biker the training list sessions that the biker already performed. Choosing one training will send the biker to another mockup where is presented a simple report of the training aspects. If a user enters as a biker, he only sees is own training sessions.



Mountain Bike App LOGOUT

Duration	0:1:31 HH:mm:ss
Max. Altitude	112.3 m
Min. Altitude	88.3 m
Distance	0.29 Km
Calories	14.8 kcal
Máx. Velocity	13.76 km/h
Average Speed	10.79 km/h

Figure 11 - Report from Training pressed (Biker point of View)

To consult another aspects of the training session the biker need to press the icon in the upper left corner. Pressing the icon will present a Navigation Drawer with all fragments possible to analyze all aspects of training. In the graphs is possible to get every point value, the biker only need to press with is finger.

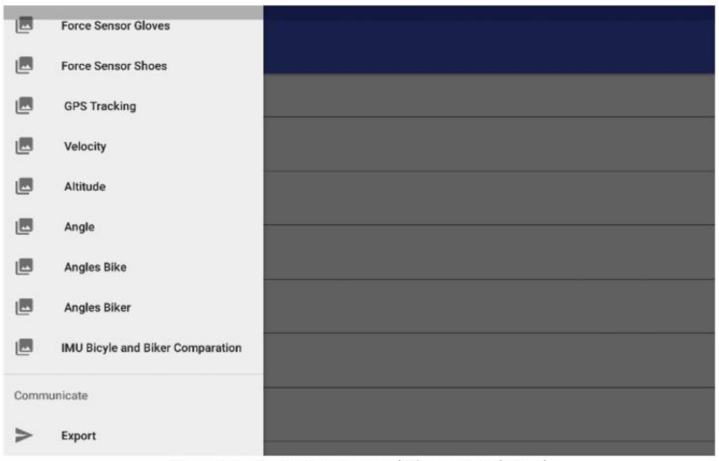


Figure 12 - Navigation Drawer (Biker point of View)

All this features that we see in the Navigation Drawer are the same if the user is a biker or a coach, can analyze the workout the same way, and export the training session to SD card.

2.1. Perform a training Session

To perform a training session, the only things that a biker need to do is turning ON the microcontrollers and put the gloves and insoles in the shoes. There are six microcontrollers so the biker need to connect all six microcontrollers.

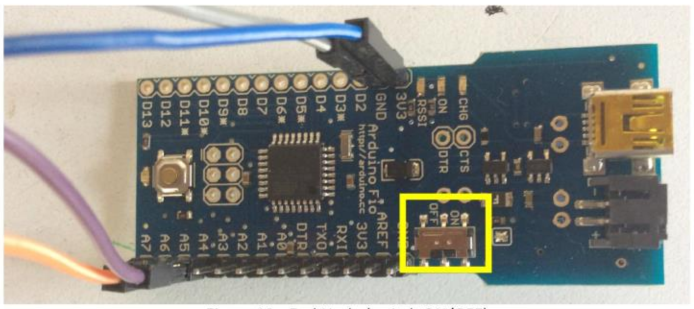


Figure 13 - End Node (switch ON/OFF)

It is advisable to connect the coordinator for last. When all equipment is connected the user pass the tag or card in the coordinator and wait to led gets ON, after that the biker can perform is training.

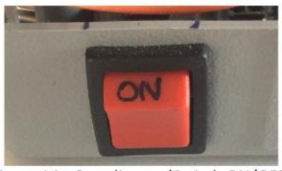
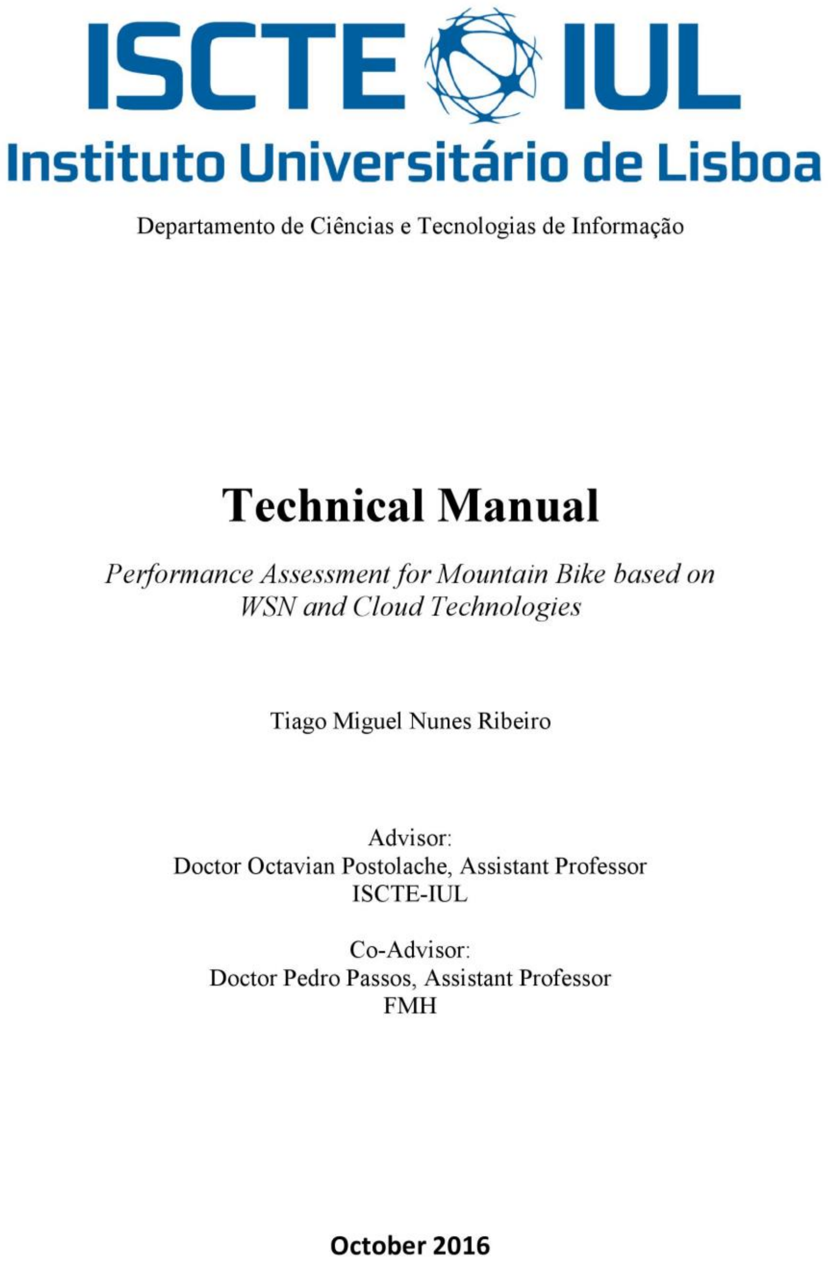


Figure 14 - Coordinator (Switch ON/OFF)

It's very important that when the biker end is training session make the same thing, this is the biker the to turn off all equipment. This is all the necessary procedures to a system works well.

Appendix C

Technical Manual



ISCTE IUL
Instituto Universitário de Lisboa

Departamento de Ciências e Tecnologias de Informação

Technical Manual

Performance Assessment for Mountain Bike based on WSN and Cloud Technologies

Tiago Miguel Nunes Ribeiro

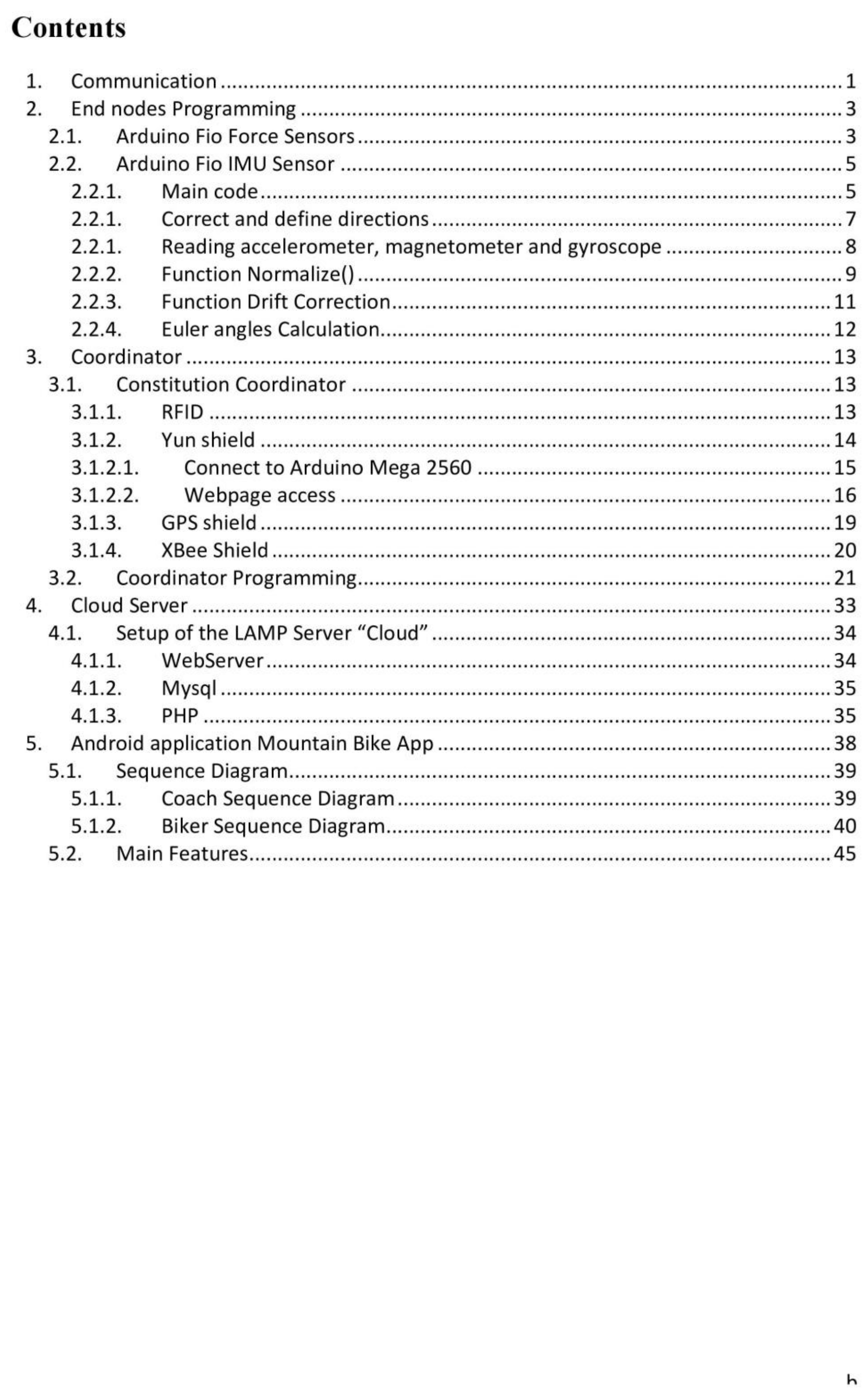
Advisor:
Doctor Octavian Postolache, Assistant Professor
ISCTE-IUL

Co-Advisor:
Doctor Pedro Passos, Assistant Professor
FMH

October 2016

Figures

Tables



Contents

h

1. Communication

All the communication between the End nodes and the coordinator is made by XBee using the protocol ZigBee based on IEEE 802.15.4.

The XBee modules can be configured in two ways: Transparent Mode (AT) and API Mode (Application Programming Interface). The mode AT are limited to point-to-point communication between two XBees. In API mode, we can trivially send and receive from both the coordinator and many XBees out in the WSN. Additionally, API mode will expose a variety of additional information encoded in each packet. In this work the communication is point to point but we use the mode API, this mode is a frame-based method for sending and receiving data to and from a radio's serial UART. This mode composes a packet and permit that the information doesn't suffer interference.

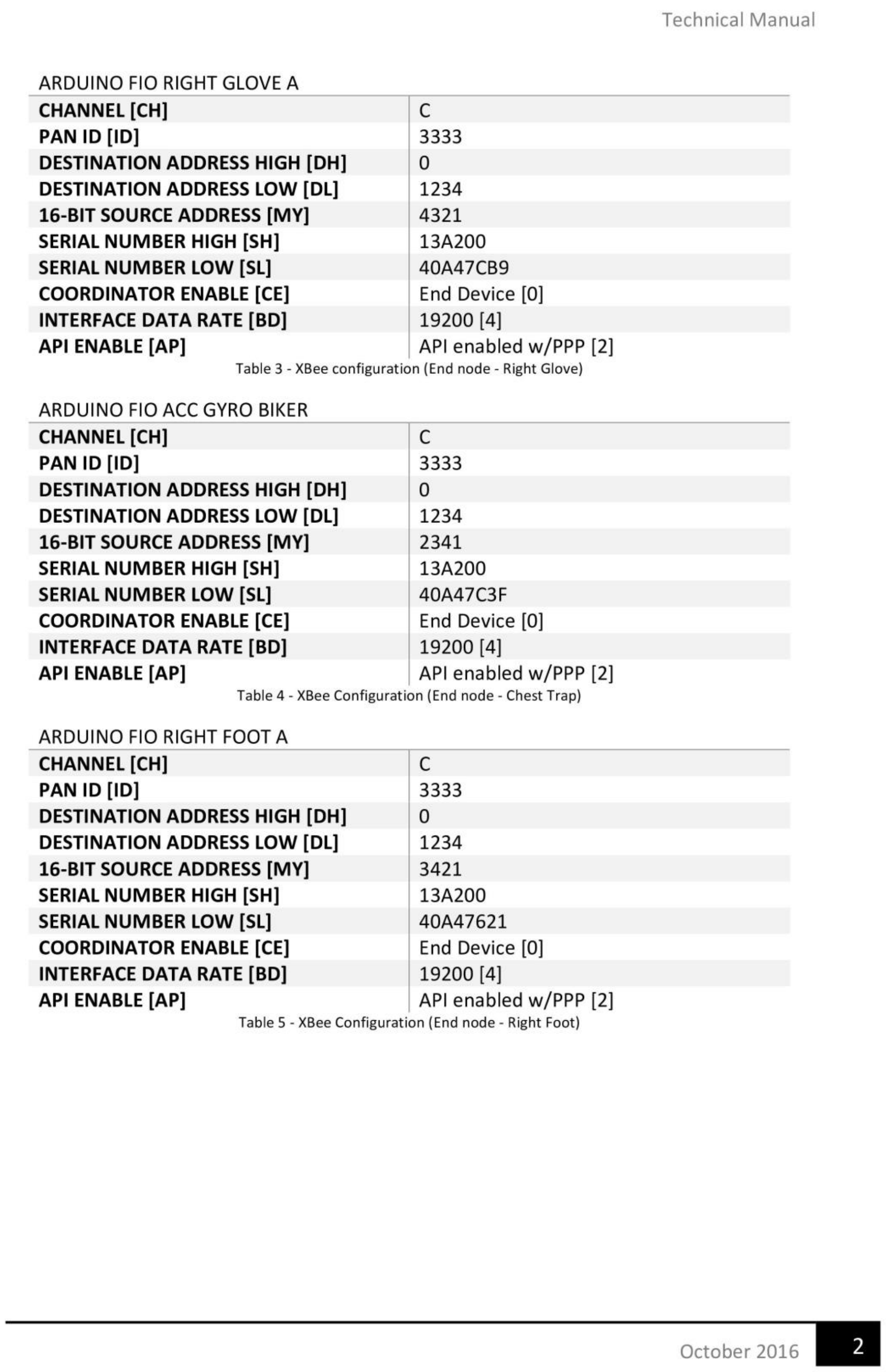
Below there are the settings made in the all XBee nodes through the XCTU software:

COORDINATOR	
CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	4321
16-BIT SOURCE ADDRESS [MY]	1234
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40A48802
COORDINATOR ENABLE [CE]	Coordinator [1]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled [1]

Table 1 - XBee configuration (Coordinator Bicycle Frame)

ARDUINO FIO LEFT GLOVE B	
CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	1234
16-BIT SOURCE ADDRESS [MY]	3214
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40A487A2
COORDINATOR ENABLE [CE]	End Device [0]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled w/PPP [2]

Table 2 - XBee configuration (End node - Left Glove)



ARDUINO FIO RIGHT GLOVE A

CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	1234
16-BIT SOURCE ADDRESS [MY]	4321
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40A47CB9
COORDINATOR ENABLE [CE]	End Device [0]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled w/PPP [2]

Table 3 - XBee configuration (End node - Right Glove)

ARDUINO FIO ACC GYRO BIKER

CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	1234
16-BIT SOURCE ADDRESS [MY]	2341
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40A47C3F
COORDINATOR ENABLE [CE]	End Device [0]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled w/PPP [2]

Table 4 - XBee Configuration (End node - Chest Trap)

ARDUINO FIO RIGHT FOOT A

CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	1234
16-BIT SOURCE ADDRESS [MY]	3421
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40A47621
COORDINATOR ENABLE [CE]	End Device [0]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled w/PPP [2]

Table 5 - XBee Configuration (End node - Right Foot)

ARDUINO FIO LEFT FOOT B	
CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	1234
16-BIT SOURCE ADDRESS [MY]	2314
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40A47C28
COORDINATOR ENABLE [CE]	End Device [0]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled w/PPP [2]

Table 6 - XBee configuration (End node - Left Foot)

ARDUINO FIO ACC GYRO BICYCLE	
CHANNEL [CH]	C
PAN ID [ID]	3333
DESTINATION ADDRESS HIGH [DH]	0
DESTINATION ADDRESS LOW [DL]	1234
16-BIT SOURCE ADDRESS [MY]	2431
SERIAL NUMBER HIGH [SH]	13A200
SERIAL NUMBER LOW [SL]	40F9DE39
COORDINATOR ENABLE [CE]	End Device [0]
INTERFACE DATA RATE [BD]	19200 [4]
API ENABLE [AP]	API enabled w/PPP [2]

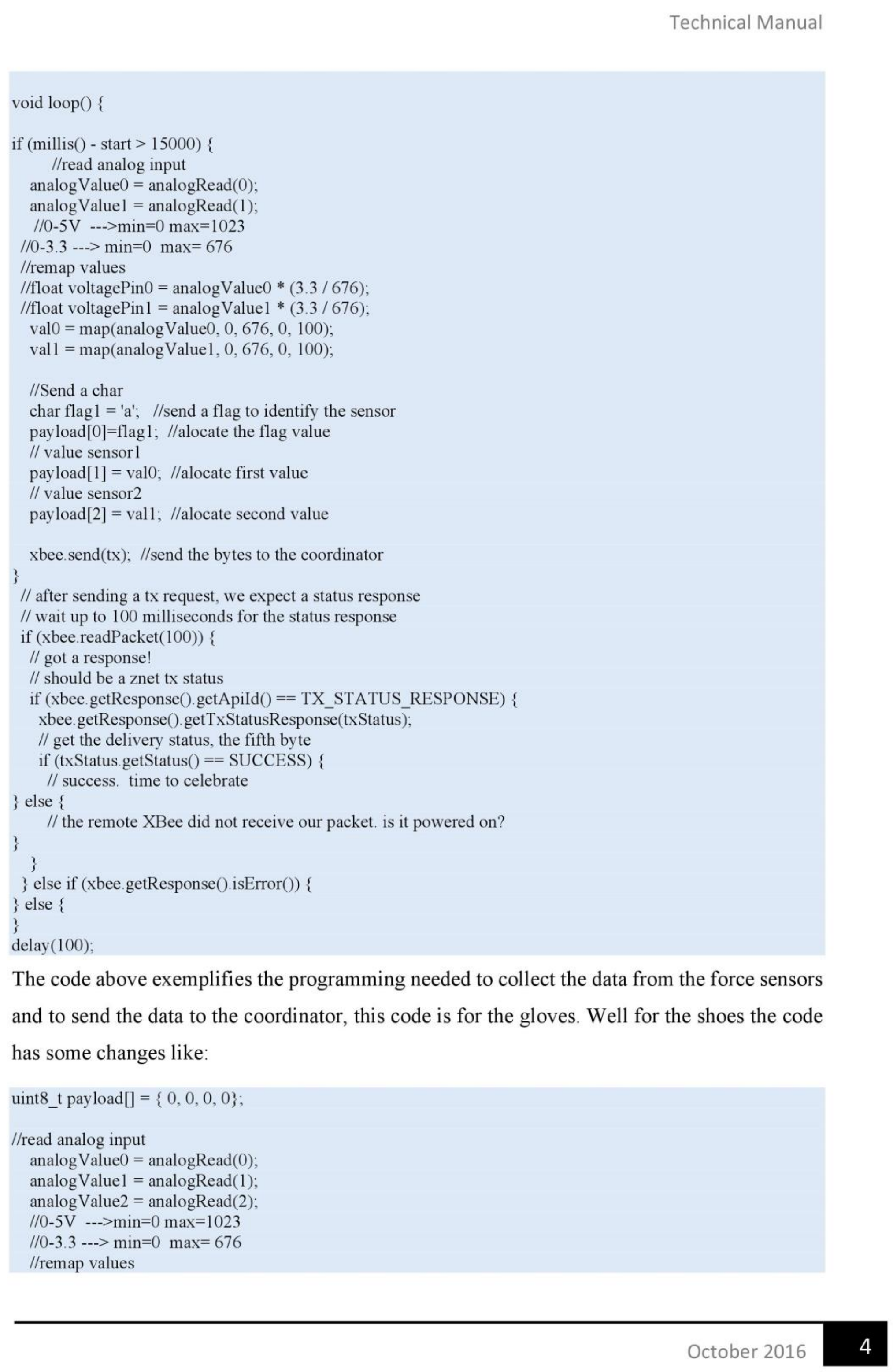
Table 7 - XBee configuration (End node - Bicycle Frame)

2. End nodes Programming

2.1. Arduino Fio Force Sensors

In this work four of the microcontrollers are using the force sensors A201 from the FlexiForce. The values retrieved from the sensors are converted ADC values to percentage. So the XBee have to send percentage values for the coordinator.

```
#include <XBee.h> //use the library XBee.h

XBee xbee = XBee();
unsigned long start = millis();

uint8_t payload[] = { 0, 0, 0};

// 64-bit addressing: This is the SH + SL address of remote XBee
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x40a48802);
Tx64Request tx = Tx64Request(addr64, payload, sizeof(payload));
TxStatusResponse txStatus = TxStatusResponse();

void setup() {
Serial.begin(19200);
  xbee.setSerial(Serial);
}
```



```
void loop() {

if (millis() - start > 15000) {
        //read analog input
    analogValue0 = analogRead(0);
    analogValue1 = analogRead(1);
     //0-5V  --->min=0 max=1023
  //0-3.3 ---> min=0  max= 676
  //remap values
  //float voltagePin0 = analogValue0 * (3.3 / 676);
  //float voltagePin1 = analogValue1 * (3.3 / 676);
    val0 = map(analogValue0, 0, 676, 0, 100);
    val1 = map(analogValue1, 0, 676, 0, 100);

    //Send a char
    char flag1 = 'a';   //send a flag to identify the sensor
    payload[0]=flag1;  //alocate the flag value
    // value sensor1
    payload[1] = val0;  //alocate first value
    // value sensor2
    payload[2] = val1;  //alocate second value

    xbee.send(tx);  //send the bytes to the coordinator
}
  // after sending a tx request, we expect a status response
  // wait up to 100 milliseconds for the status response
  if (xbee.readPacket(100)) {
    // got a response!
    // should be a znet tx status
    if (xbee.getResponse().getApiId() == TX_STATUS_RESPONSE) {
      xbee.getResponse().getTxStatusResponse(txStatus);
      // get the delivery status, the fifth byte
      if (txStatus.getStatus() == SUCCESS) {
        // success.  time to celebrate
} else {
        // the remote XBee did not receive our packet. is it powered on?
}
    }
  } else if (xbee.getResponse().isError()) {
} else {
}
delay(100);
```

The code above exemplifies the programming needed to collect the data from the force sensors and to send the data to the coordinator, this code is for the gloves. Well for the shoes the code has some changes like:

```
uint8_t payload[] = { 0, 0, 0, 0};

//read analog input
    analogValue0 = analogRead(0);
    analogValue1 = analogRead(1);
    analogValue2 = analogRead(2);
    //0-5V  --->min=0 max=1023
    //0-3.3 ---> min=0  max= 676
    //remap values
```

```
  //float voltagePin0 = analogValue0 * (3.3 / 676);
  //float voltagePin1 = analogValue1 * (3.3 / 676);
 //float voltagePin2 = analogValue2 * (3.3 / 676);
  val0 = map(analogValue0, 0, 676, 0, 100);//map the value to transform in percentage
  val1 = map(analogValue1, 0, 676, 0, 100);
  val2 = map(analogValue2, 0, 676, 0, 100);
  //Send a char
  char flag1 = 'd';
  payload[0] = flag1;
  //send value sensor1
  payload[1] = val0;  //alocate first value
  //send value sensor2
  payload[2] = val1;  //alocate second value
  //send value sensor3
  payload[3] = val2;  //alocate third value
```

These are the main changes to use the three force sensors in the shoes, but programming the XBee does not change. The baud rate used in this work is 19200. The baud rate specifies how fast data is sent over a serial line. It's usually expressed in units of bits-per-second (bps). If we invert the baud rate, we can find out how just long it takes to transmit a single bit. This value determines how long the transmitter holds a serial line high/low or at what period the receiving device sample its line. Baud rates can be just about any value, the only requirement is that both devices operate at the same rate. The more common baud rates are 9600, 19200,38400, 57600 and 115200.

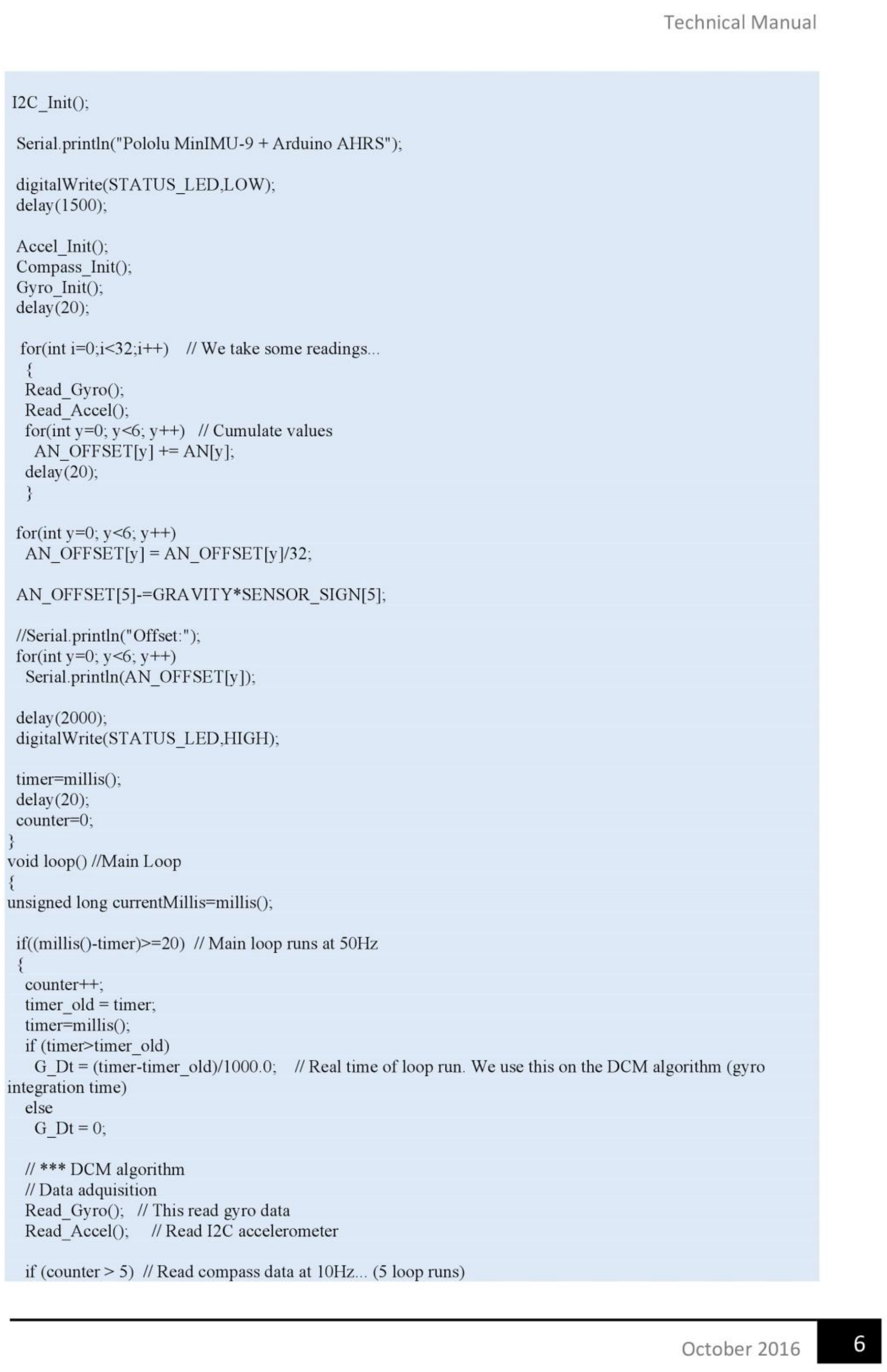
2.2. Arduino Fio IMU Sensor

In the case of the IMU sensor the history is little more difficult to explain and understand. This is a Direction Cosine Matrix (DCM) based Attitude Heading Reference System (AHRS) with gyro drift correction based on accelerometer (gravity) vector and magnetometer (compass) vector.

2.2.1. Main code

The main code is presented above, where is possible to see the initiation of the I^2C, then the gyro and accelerometer are reading, some values are accumulated to create a reference. After that the values are taken and some calculations are made. The main loop runs at 50Hz and the compass data are read at 10Hz. At the main code some important functions are called as will be mentioned below:

```
void setup()
{
 Serial.begin(19200);
 pinMode (STATUS_LED,OUTPUT);  // Status LED
```



Technical Manual

```
I2C_Init();

 Serial.println("Pololu MinIMU-9 + Arduino AHRS");

 digitalWrite(STATUS_LED,LOW);
 delay(1500);

 Accel_Init();
 Compass_Init();
 Gyro_Init();
 delay(20);

  for(int i=0;i<32;i++)    // We take some readings...
   {
   Read_Gyro();
   Read_Accel();
   for(int y=0; y<6; y++)   // Cumulate values
     AN_OFFSET[y] += AN[y];
   delay(20);
   }

 for(int y=0; y<6; y++)
   AN_OFFSET[y] = AN_OFFSET[y]/32;

 AN_OFFSET[5]-=GRAVITY*SENSOR_SIGN[5];

 //Serial.println("Offset:");
 for(int y=0; y<6; y++)
   Serial.println(AN_OFFSET[y]);

 delay(2000);
 digitalWrite(STATUS_LED,HIGH);

 timer=millis();
 delay(20);
 counter=0;
}
void loop() //Main Loop
{
unsigned long currentMillis=millis();

 if((millis()-timer)>=20)  // Main loop runs at 50Hz
  {
   counter++;
   timer_old = timer;
   timer=millis();
   if (timer>timer_old)
     G_Dt = (timer-timer_old)/1000.0;    // Real time of loop run. We use this on the DCM algorithm (gyro
integration time)
   else
     G_Dt = 0;

   // *** DCM algorithm
   // Data adquisition
   Read_Gyro();   // This read gyro data
   Read_Accel();     // Read I2C accelerometer

   if (counter > 5)  // Read compass data at 10Hz... (5 loop runs)
```

October 2016 6

```
    {
    counter=0;
    Read_Compass();    // Read I2C magnetometer
    Compass_Heading(); // Calculate magnetic heading
    }

  // Calculations...
  Matrix_update();
  Normalize();
  Drift_correction();
  Euler_angles();
  // ***

  if((currentMillis-previousMillis)>=interval){
    previousMillis=currentMillis;
    printdata();
  }
 }
}
```

For a visual and a better comprehension is shown the visual representation of the DCM Algorithm:

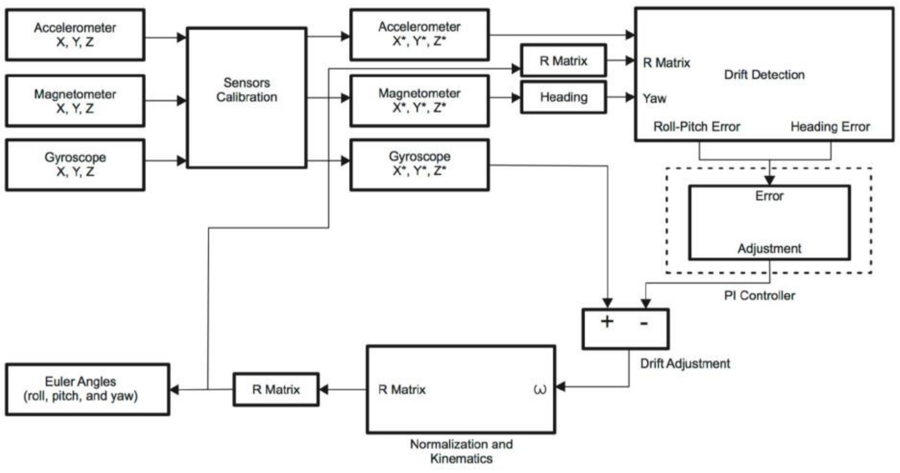


Figure 1 - Direct Cosine Matrix Algorithm Overview

2.2.1. Correct and define directions

```
int SENSOR_SIGN[9] = {1,1,-1,-1,-1,-1,1,1,-1}; //Correct directions x,y,z - gyro, accelerometer, magnetometer
```

This little peace of code, defines/correct the positon of the axis. Looking to the IMU board the axis will be defined as follows:

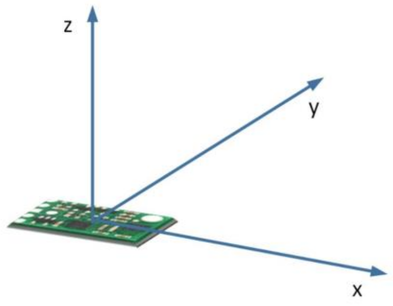


Figure 2 - Definition IMU axis

The board must be mounted in this way to respect the orientation of the axis.

2.2.1. Reading accelerometer, magnetometer and gyroscope

The L3GD20H's gyro and the LSM303D's accelerometer and magnetometer libraries can be queried and configured through the I^{2}C bus. Using the library LSM303D was performed a calibration using the sketch "calibration.ino", this calibration means applying the correct gain multipliers to the magnetometer signals before applying the update algorithm. Each of the three sensors acts as a slave device on the same I^{2}C bus, their clock and data lines are tied together to ease communication with microcontrollers operating at the same voltage as VIN. The L3GD20H and LSM303D each have separate slave addresses on the I^{2}C bus. The following code is used to read values from gyro, accelerometer and magnetometer and the configurable options to select the sensitivities for the gyro, accelerometer and magnetometer.

```
#include <L3G.h>
#include <LSM303.h>

L3G gyro;
LSM303 compass;

void I2C_Init(){
  Wire.begin();
}

void Gyro_Init(){
  gyro.init();
  gyro.enableDefault();
  gyro.writeReg(L3G::CTRL_REG4, 0x20); // 2000 dps full scale
  gyro.writeReg(L3G::CTRL_REG1, 0x0F); // normal power mode, all axes enabled, 100 Hz
}

void Read_Gyro(){
  gyro.read();
  AN[0] = gyro.g.y; //roll
  AN[1] = gyro.g.x;  //pitch
  AN[2] = gyro.g.z;  //yaw
  gyro_x = SENSOR_SIGN[0] * (AN[0] - AN_OFFSET[0]);
  gyro_y = SENSOR_SIGN[1] * (AN[1] - AN_OFFSET[1]);
  gyro_z = SENSOR_SIGN[2] * (AN[2] - AN_OFFSET[2]);
}

void Accel_Init(){
  compass.init();
  compass.enableDefault();
  switch (compass.getDeviceType())
  {
    case LSM303::device_D:
      compass.writeReg(LSM303::CTRL2, 0x18); // 8 g full scale: AFS = 011
      break;
    case LSM303::device_DLHC:
      compass.writeReg(LSM303::CTRL_REG4_A, 0x28); // 8 g full scale: FS = 10; high resolution output mode
      break;
    default: // DLM, DLH
      compass.writeReg(LSM303::CTRL_REG4_A, 0x30); // 8 g full scale: FS = 11
  }
}
// Reads x,y and z accelerometer registers
void Read_Accel(){
  compass.readAcc();

  AN[3] = compass.a.y >> 4; // shift left 4 bits to use 12-bit representation (1 g = 256)  //roll
  AN[4] = compass.a.x >> 4;  //pitch
  AN[5] = compass.a.z >> 4;  //yaw
  accel_x = SENSOR_SIGN[3] * (AN[3] - AN_OFFSET[3]);
  accel_y = SENSOR_SIGN[4] * (AN[4] - AN_OFFSET[4]);
  accel_z = SENSOR_SIGN[5] * (AN[5] - AN_OFFSET[5]);
}

void Compass_Init(){
  // doesn't need to do anything because Accel_Init() should have already called compass.enableDefault()
}

void Read_Compass(){
  compass.readMag();

  magnetom_x = SENSOR_SIGN[6] * compass.m.y;  //roll
  magnetom_y = SENSOR_SIGN[7] * compass.m.x;  //pitch
  magnetom_z = SENSOR_SIGN[8] * compass.m.z;  //yaw
}
```

2.2.2. Function Normalize()

Then the matrix is updated, this is the matrix is populated at each iteration. The next process at each iteration is the implementation of the DCM algorithm through the function Normalize(), in this function the vector dot and cross products are used.

```
void Normalize(void)
{
  float error=0;
float temporary[3][3];
float renorm=0;

error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5; //eq. 2

Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq. 2
Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq. 2

Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//eq. 2
Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//eq. 2

Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x b //eq. 3

renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //eq. 4
Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //eq. 4
Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //eq. 4
Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}
```

First compute the dot product of X and Y rows of the matrix, which is supposed to be zero, so the result is a measure of how much the X and Y rows are rotating toward each other.

$$X = \begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} \; Y = \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix}$$

$$error = X.Y = X^T Y = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \end{bmatrix} \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} \tag{1}$$

We apportion half of the error each to the X and Y rows, and approximately rotate the X and Y rows in the opposite direction by cross coupling.

$$X = \begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} = X_{orthogonal} = X - \frac{error}{2} Y$$

$$Y = \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} = Y_{orthogonal} = Y - \frac{error}{2} X \tag{2}$$

Orthogonal error is greatly reduced by substituting equation (2) into (1), keeping in mind that the magnitude of each row and column of the R matrix is approximately equal to one. The next step is to adjust the Z row of the matrix to be orthogonal to X and Y row. The way to do that is to simply set the Z row to be the cross product of the X and Y rows.

$$\begin{bmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{bmatrix} = Z_{orthogonal} = X_{orthogonal} \times Y_{orthogonal} \tag{3}$$

The last step in the renormalization process is to scale the rows of the R matrix to assure that each has a magnitude equal to one. The resulting magnitude adjustment equations for the row vectors are:

$$X_{normalized} = \frac{1}{2}(3 - X_{orthogonal}.X_{orthogonal})X_{orthogonal}$$

$$Y_{normalized} = \frac{1}{2}(3 - Y_{orthogonal}.Y_{orthogonal})Y_{orthogonal}$$

$$Z_{normalized} = \frac{1}{2}(3 - Z_{orthogonal}.Z_{orthogonal})Z_{orthogonal}$$

(4)

These equations state that to adjust the magnitude of each row vector to one, is necessary to subtract the dot product of the vector with itself (the square of the magnitude), subtract from three, multiply by ½ , and multiply each element of the vector by the result.

2.2.3. Function Drift Correction

Although the gyros perform rather well, with an uncorrected offset on the order of a few degrees per second, eventually we have to do something about their drift. What is done is to use other orientation references in this case is used a correction based on compass magnetic heading. Then the drift correction is applied:

```
void Drift_correction(void)
{
  float mag_heading_x;
  float mag_heading_y;
  float errorCourse;
  //Compensation the Roll, Pitch and Yaw drift.
  static float Scaled_Omega_P[3];
  static float Scaled_Omega_I[3];
  float Accel_magnitude;
  float Accel_weight;


  //*****Roll and Pitch***************

  // Calculate the magnitude of the accelerometer vector
  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] + Accel_Vector[1]*Accel_Vector[1] +
Accel_Vector[2]*Accel_Vector[2]);
  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.
  // Dynamic weighting of accelerometer info (reliability filter)
```

```
 // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)
 Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);  //

 Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0]); //adjust the ground of
reference
 Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);

 Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_weight);
 Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);

 //*****YAW***************
 // We make the gyro YAW drift correction based on compass magnetic heading

 mag_heading_x = cos(MAG_Heading);
 mag_heading_y = sin(MAG_Heading);
 errorCourse=(DCM_Matrix[0][0]*mag_heading_y) - (DCM_Matrix[1][0]*mag_heading_x);  //Calculating
YAW error
 Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw correction to the XYZ rotation of
the bicycle, depeding the position.

 Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01proportional of YAW.
 Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding  Proportional.

 Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001Integrator
 Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the Omega_I
}
```

2.2.4. Euler angles Calculation

The next process is the calculation of the pitch roll and yaw through the following equations:

```
void Euler_angles(void){
 pitch = -asin(DCM_Matrix[2][0]);
 roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
 yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}
```

$$R = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} = rotation\ matrix \tag{5}$$

$$R = \begin{bmatrix} cos\theta cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ cos\theta sin\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & cos\psi sin\theta sin\phi - sin\psi cos\phi \\ -sin\theta & sin\psi cos\theta & cos\psi cos\theta \end{bmatrix} \tag{6}$$

$$\begin{gathered} Pitch \rightarrow \psi = -\operatorname{asin}(r_{zx}) \\ Roll \rightarrow \theta = atan2(r_{zy}, r_{zz}) \\ Yaw \rightarrow \phi = atan2(r_{yx}, r_{xx}) \end{gathered} \tag{7}$$

The last but not least is the sending to the coordinator of the angles through XBee, this delivery is performed at an interval of 1000 milliseconds.

3. Coordinator

In the coordinator there are an immense of process's that can be explained. On the coordinator is received all data form end nodes, the information is processed and send through a Wi-Fi connection to the database. The connection to the database and the data sent is through a php module. In the coordinator is located the local database for security.

3.1. Constitution Coordinator

To get an idea of the constitution of the coordinator is presented a schematic circuit:

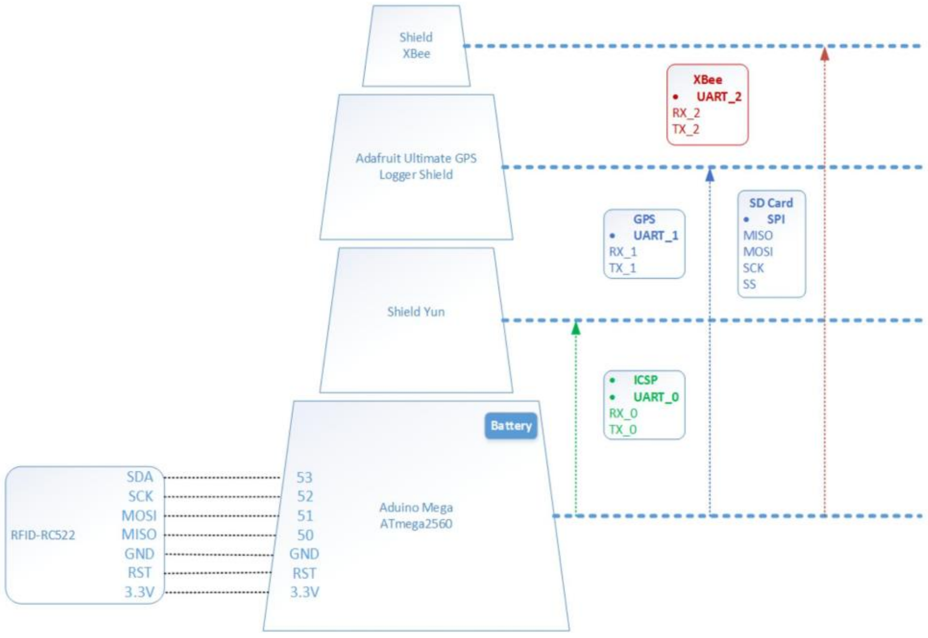


Figure 3 - Coordinator Overview

3.1.1. RFID

Radio frequency identification uses a passive device (called an RFID tag) to communicate data using radio frequency through electromagnetic induction. In the project it is used the sensor MFRC522, is a highly integrated reader/writer IC for contactless communication at 13.56MHz. The MFRC522 reader supports ISO/IEC 14443 A/MIFAR and NTAG. This sensor uses the MFRC522 library, and like in the scheme is shown this sensor is connected to the Arduino Mega through the SPI protocol.

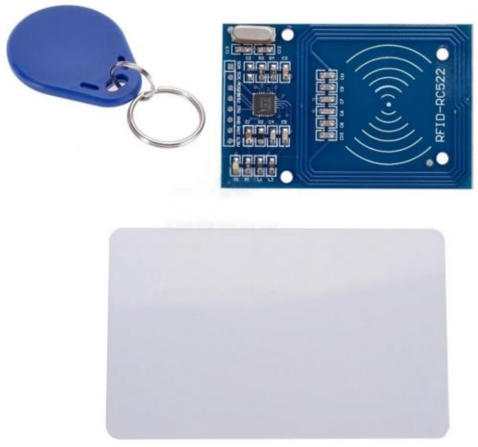


Figure 4 - Sensor RFID-RC522 & Tag and Card

3.1.2. Yun shield

One of the the problems of every project in this area was the sending of the collected data form the sensors to the cloud, using this shield the communication it was possible. This shield uses two protocols of communication with the microcontroller ICSP and UART.

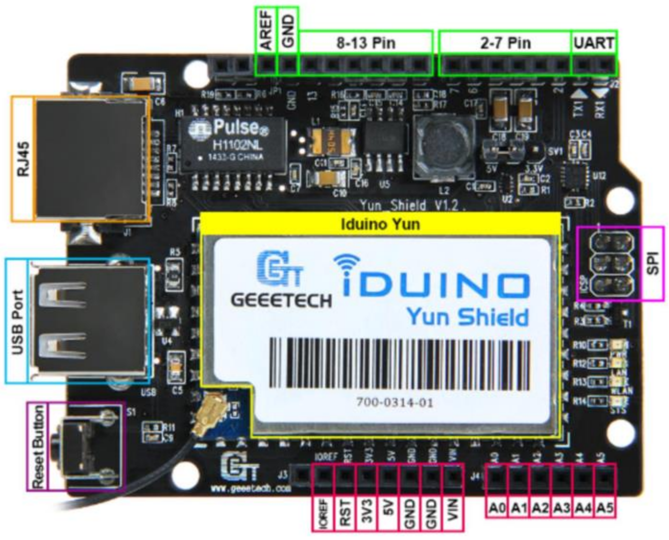


Figure 5 - Yun Shield

In the Mega2560, the UART between mega2560 and mega16u2 will influence the Bridge feature with the Iduino Yun Shield. So we have to disconnect it by setting mega16u2 into reset mode fig. 6. Made this little intervention on Arduino mega, the utilization of protocol ICSP is fundamental because it will possible to develop the applications through a Wi-Fi connection. Eliminating the need of the USB cable connected to the microcontroller, for the insertion of new sketch's.

A universal asynchronous receiver/transmitter (UART) is a block of circuit responsible for implementing serial communication. Essentially, the UART acts as an intermediary between parallel and serial interfaces. In our case the Arduino Mega – built on an ATmega2560 – has a whopping four UARTs

3.1.2.1. Connect to Arduino Mega 2560

The shield Yun and Arduino Mega (together) is similar to the Leonardo in that the ATmega16u2 has built-in USB communication, eliminating the need for a secondary processor.

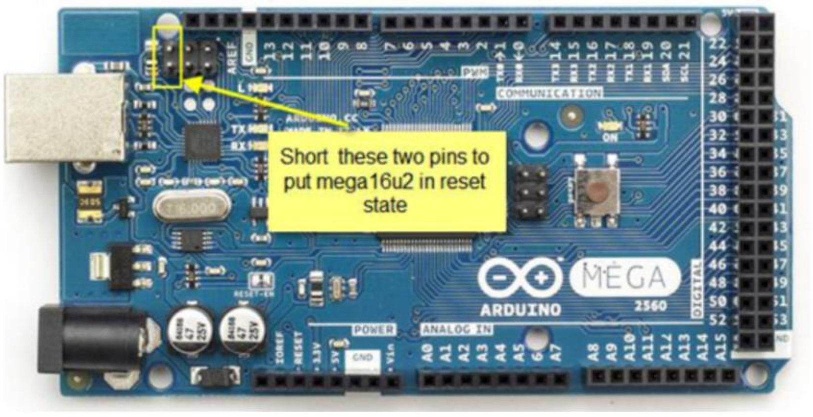


Figure 6 - Arduino Mega (configuration to support Yun Shield)

To programming the microcontroller with this shield connected is needed to add a "Mega2560 Yun" board type in the file: Arduino\hardware\arduino\avr\board.txt, as below, and reopen the Arduino IDE:

```
mega2560Yun.name=Arduino Mega 2560 -- Iduino Yún
mega2560Yun.upload.via_ssh=true
mega2560Yun.vid.0=0x2341
mega2560Yun.pid.0=0x0044
mega2560Yun.vid.1=0x2341
```

```
mega2560Yun.pid.1=0x003f
mega2560Yun.upload.tool=avrdude
mega2560Yun.upload.protocol=arduino
mega2560Yun.upload.maximum_size=258048
mega2560Yun.upload.maximum_data_size=8192
mega2560Yun.upload.speed=57600
mega2560Yun.upload.disable_flushing=true
mega2560Yun.upload.use_1200bps_touch=true
mega2560Yun.upload.wait_for_upload_port=true
mega2560Yun.bootloader.tool=avrdude
mega2560Yun.bootloader.low_fuses=0xff
mega2560Yun.bootloader.high_fuses=0xd8
mega2560Yun.bootloader.extended_fuses=0xfd
mega2560Yun.bootloader.file=stk500v2/stk500boot_v2_mega2560.hex
mega2560Yun.bootloader.unlock_bits=0x3F
mega2560Yun.bootloader.lock_bits=0x0F
mega2560Yun.build.mcu=atmega2560
mega2560Yun.build.f_cpu=16000000L
mega2560Yun.build.board=AVR_MEGA2560
mega2560Yun.build.core=arduino
mega2560Yun.build.variant=mega
```

This allows the shield Yun to appear to connected to a computer as mouse and keyboard, in addition to a virtual (CDC) serial / COM port.

3.1.2.2. Webpage access

To access to the Webpage use a browser to set Iduino_Yun_Shield, and you will see the login after entering http://192.168.240.1, if it was the first interaction with the board. Otherwise the IP can be seen on the IDE of the Arduino or in other mechanisms like an application used "Fing".

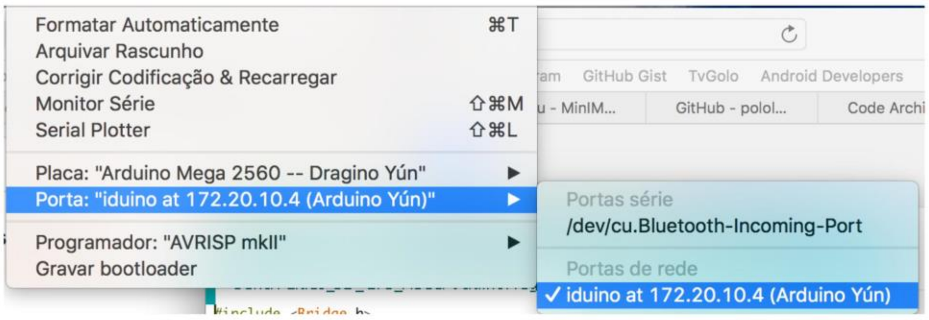
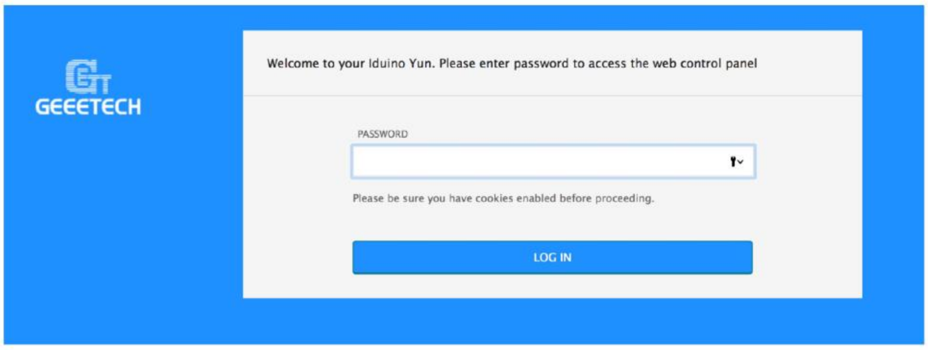

Formatar Automaticamente ⌘T
Arquivar Rascunho
Corrigir Codificação & Recarregar
Monitor Série ⇧⌘M
Serial Plotter ⇧⌘L
Placa: "Arduino Mega 2560 -- Dragino Yún"
Porta: "iduino at 172.20.10.4 (Arduino Yún)"
Programador: "AVRISP mkII"
Gravar bootloader
Portas série
/dev/cu.Bluetooth-Incoming-Port
Portas de rede
✓ iduino at 172.20.10.4 (Arduino Yún)

Figure 7 - Arduino IDE (discovering IP yun shield)

After putting the IP of the Wi-Fi shield on the web browser the login page will appear.



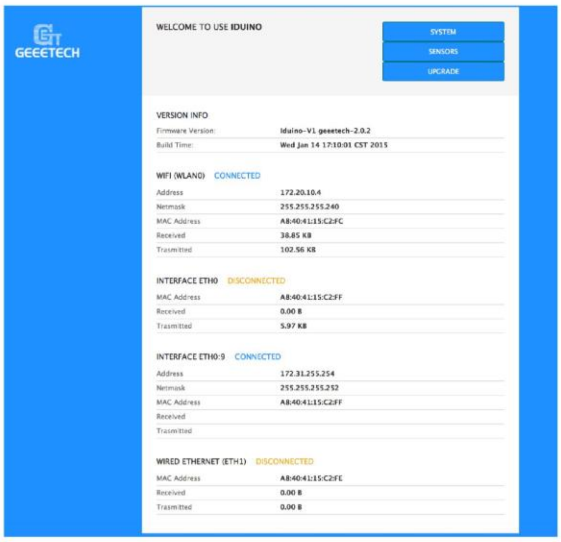
Welcome to your Iduino Yun. Please enter password to access the web control panel

PASSWORD

Please be sure you have cookies enabled before proceeding.

Figure 8 - Web Interface Yun shield (Login Area)

Default password for Iduino_Yun_Shield is ‘iduino’. After login, the gui will show the WiFi/ETH status.



WELCOME TO USE **IDUINO**

VERSION INFO

Firmware Version:	Iduino-V1 geeetech-2.0.2
Build Time:	Wed Jan 14 17:10:01 CST 2015

WIFI (WLAN0) CONNECTED

Address	172.20.10.4
Netmask	255.255.255.240
MAC Address	A8:40:41:15:C2:FC
Received	38.85 KB
Trasmitted	102.56 KB

INTERFACE ETH0 DISCONNECTED

MAC Address	A8:40:41:15:C2:FF
Received	0.00 B
Trasmitted	5.97 KB

INTERFACE ETH0:9 CONNECTED

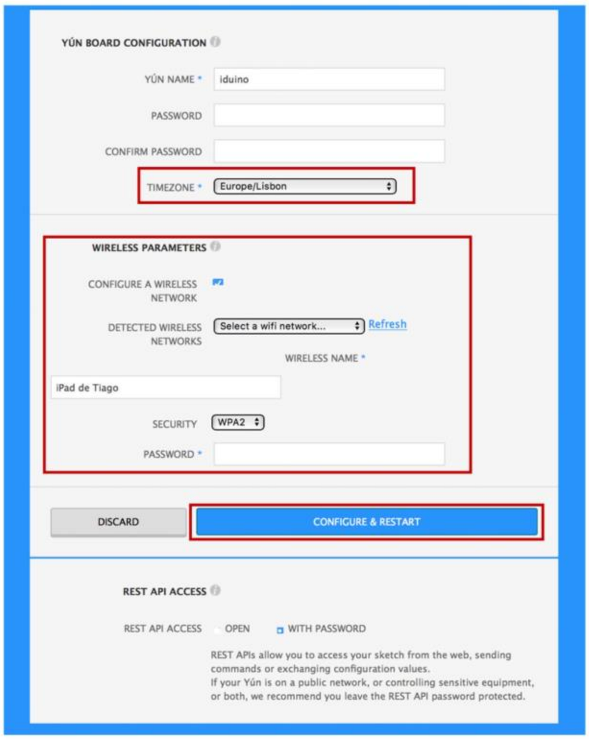
Address	172.31.255.254
Netmask	255.255.255.252
MAC Address	A8:40:41:15:C2:FF
Received	
Trasmitted	

WIRED ETHERNET (ETH1) DISCONNECTED

MAC Address	A8:40:41:15:C2:FE
Received	0.00 B
Trasmitted	0.00 B

Figure 9 - Web Interface Yun shield (After Login with success)

Clicking the SYSTEM button, you can set the device password. This page is important to define the Wireless parameters, here you can define where the yun shield is will connecting.



YÚN BOARD CONFIGURATION

YÚN NAME *	iduino
PASSWORD	
CONFIRM PASSWORD	
TIMEZONE *	Europe/Lisbon

WIRELESS PARAMETERS

CONFIGURE A WIRELESS NETWORK	✓
DETECTED WIRELESS NETWORKS	Select a wifi network... Refresh
WIRELESS NAME *	iPad de Tiago
SECURITY	WPA2
PASSWORD *	

REST API ACCESS

REST API ACCESS: OPEN [✓ WITH PASSWORD]

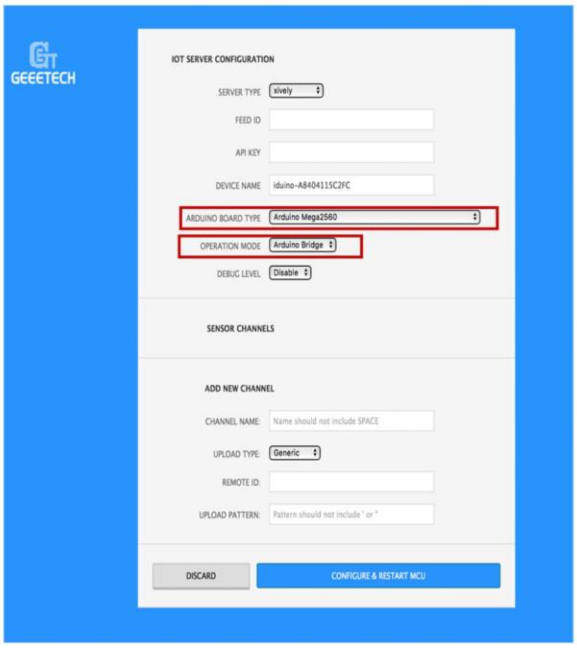
REST APIs allow you to access your sketch from the web, sending commands or exchanging configuration values.
If your Yún is on a public network, or controlling sensitive equipment, or both, we recommend you leave the REST API password protected.

Figure 10 - Web Interface Yun Shield (System)

Clicking on SESNSORS button, is important to define the settings related to the yun shield and the Arduino Mega.

Arduino board type: Define the setting of bootloader/mcu type/fuse during Sketch uploading.

Operation Mode: Make sure the mode is Arduino Bridge mode so that the Bridge class can work.



IOT SERVER CONFIGURATION

SERVER TYPE	xively
FEED ID	
API KEY	
DEVICE NAME	iduino-A8404115C2FC
ARDUINO BOARD TYPE	Arduino Mega2560
OPERATION MODE	Arduino Bridge
DEBUG LEVEL	Disable

SENSOR CHANNELS

ADD NEW CHANNEL

CHANNEL NAME:	Name should not include SPACE
UPLOAD TYPE:	Generic
REMOTE ID:	
UPLOAD PATTERN:	Pattern should not include ' or "

Figure 11 - Web Interface Yun Shield (Sensors)

3.1.3. GPS shield

In coordinator node is used a GPS receiver. The GPS shield collect the altimetry and speed of the athlete and have the ability to have an SD card, this one is useful to log the information (local data base). The GPS system operates independently of any telephonic or internet reception.

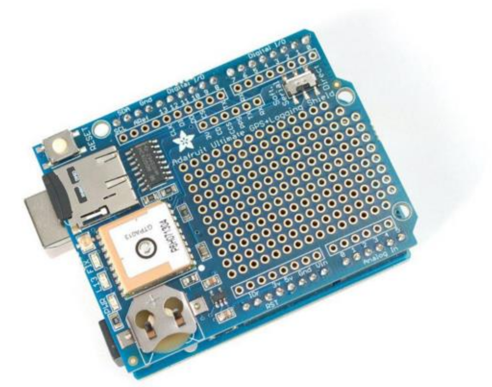


Figure 12 - Adafruit Ultimate GPS Logger Shield

The shield uses a embedded GPS module is a GlobalTop PA6H with Mediatek MT3339 chipset that achieves the industry highest level of sensitivity (-165dBm) and instant Time-to-First Fix (TTFF) with lowest power consumption for precise GPS signal processing to give the ultra-precise positioning under low receptive, high velocity conditions. FGPMMOPA6H is excellent low power consumption characteristics (acquisition 82mW, tracking 66mW), power sensitive devices, especially portable applications.

Figure 13 - GPS Module

For the GPS is used UART protocol and to connect the SD card is used SPI protocol.

3.1.4. XBee Shield

The coordinator and every end node have a simple XBee radio, this piece of electronic is based on the IEEE 802.15.4 standard designed for point-to-point and star communications. ZigBee over 802.15.4, defines specifications for low-rate WPAN for supporting simple devices that consume minimal power and typically operate in the personal space of 10m. ZigBee provides a self-organized, multi-hop, and reliable mesh networking with long battery lifetime. The shield for the XBee use UART protocol, and is responsible for the reception of all data from the WSN.

Figure 14 - Sparkfun XBee Explorer USB

3.2. Coordinator Programming

The coordinator has the particularity to use files in the SD card to allow some interactions. It is need to insert the IP of the cloud server in the file "server.txt" and register the tag Id of the bikers in the file "authids.txt" this files are present in the SD card. In the coordinator like in the end nodes it's needed use some libraries fig.15 to ensure the proper functioning of the shields and sensors connected to the coordinator (Arduino Mega 2560). On the setup is very important to ensure that the following code is in the first lines, because through this code will create the bridge between the Arduino Mega and the Yun shield.

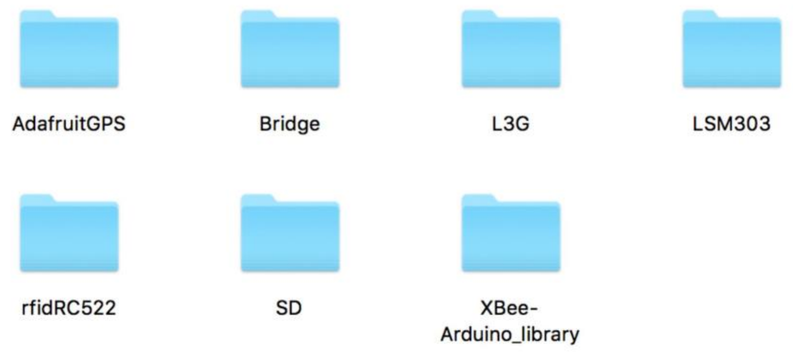
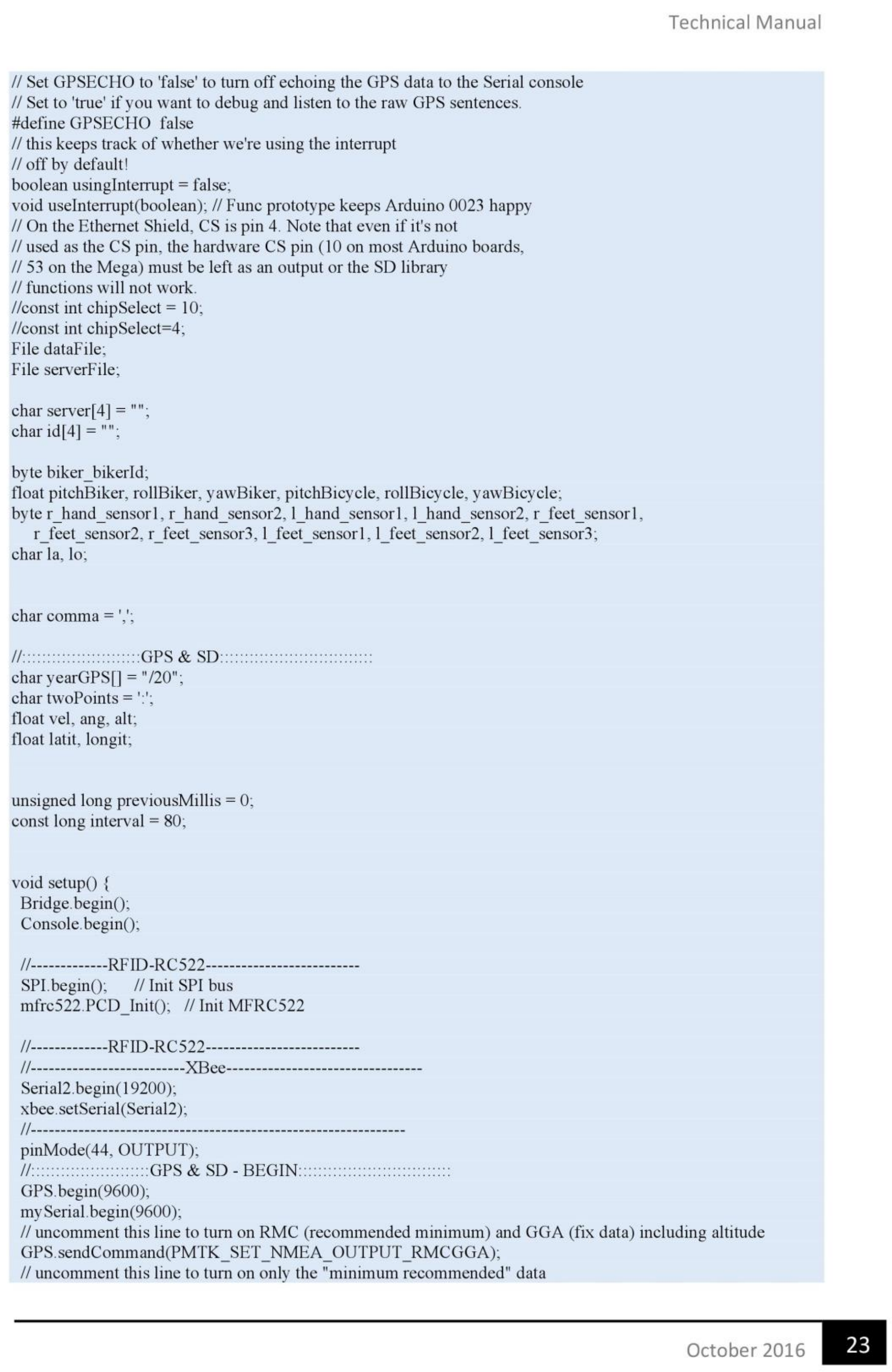


Figure 15 - Libraries used in the Arduino programming

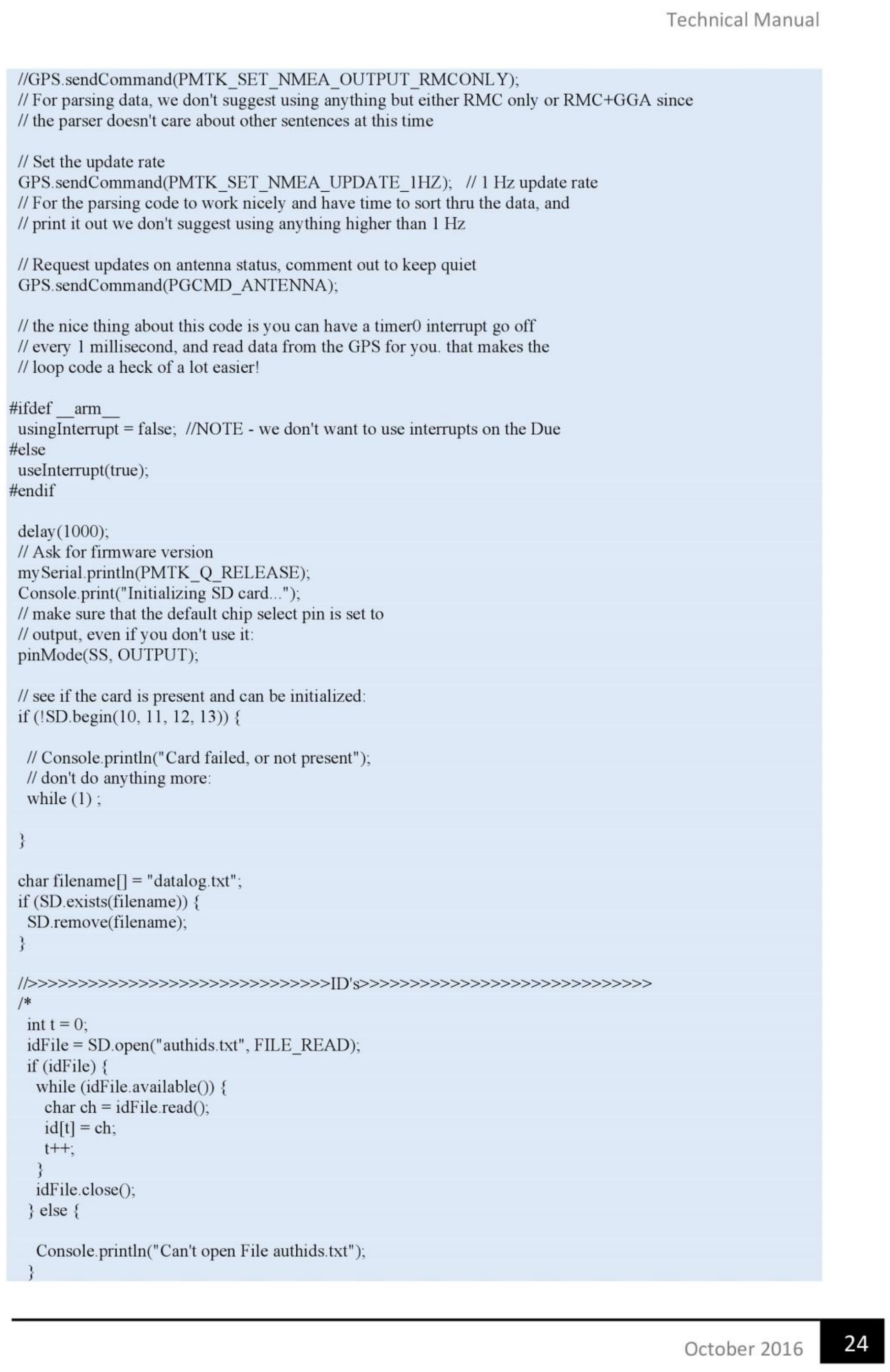
```
https://github.com/adafruit/Adafruit_GPS //Library to use the GPS shield
https://www.arduino.cc/en/Reference/YunBridgeLibrary  //library to use the Yun Shield
https://github.com/pololu/l3g-arduino  //library to get the values from the gyroscope (IMU)
https://github.com/pololu/lsm303-arduino  //library to get the values from accelerometer and magnometer (IMU)
```

```
https://github.com/miguelbalboa/rfid  //library to use the RFID Sensor
https://github.com/adafruit/SD  //library to access the SD card in the shield
https://github.com/andrewrapp/xbee-arduino  //library to receive and send data through XBee's
```

```
#include <Bridge.h>
#include <Console.h>
#include <YunClient.h>
#include <Wire.h>
#include <XBee.h>
#include <Adafruit_GPS.h>
#include <SPI.h>
#include <SD.h>
#include <MFRC522.h>

//int SENSOR_SIGN[9] = {1, 1, -1, -1, -1, -1, 1, 1, -1}; //Correct directions x,y,z - gyro, accelerometer,
magnetometer

//-----------------RFID-RC522--------------------
#define RST_PIN        35         // Configurable, see typical pin layout above
#define SS_PIN         53         // Configurable, see typical pin layout above

MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance

const int arraySize = 11;
File idFile;
int ids[arraySize];
boolean autenthicate = false;
int rideId;

//-----------------RFID-RC522--------------------

//-------------------------XBee--------------------------------
XBee xbee = XBee();
XBeeResponse response = XBeeResponse();
// create reusable response objects for responses we expect to handle
Rx16Response rx16 = Rx16Response();
Rx64Response rx64 = Rx64Response();

uint8_t option = 0;
uint8_t data = 0;

union {
  float f;
  byte b[4];

} stuff;

//--------------------------------------------------------------

//::::::::::::::::::::::::GPS & SD::::::::::::::::::::::::::::::
// GPS power pin to Arduino Due 3.3V output.
// GPS ground pin to Arduino Due ground.
// For hardware serial 1 (recommended):
//   GPS TX to Arduino Due Serial1 RX pin 19
//   GPS RX to Arduino Due Serial1 TX pin 18
#define mySerial Serial1
Adafruit_GPS GPS(&mySerial);
```



```
// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences.
#define GPSECHO  false
// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy
// On the Ethernet Shield, CS is pin 4. Note that even if it's not
// used as the CS pin, the hardware CS pin (10 on most Arduino boards,
// 53 on the Mega) must be left as an output or the SD library
// functions will not work.
//const int chipSelect = 10;
//const int chipSelect=4;
File dataFile;
File serverFile;

char server[4] = "";
char id[4] = "";

byte biker_bikerId;
float pitchBiker, rollBiker, yawBiker, pitchBicycle, rollBicycle, yawBicycle;
byte r_hand_sensor1, r_hand_sensor2, l_hand_sensor1, l_hand_sensor2, r_feet_sensor1,
    r_feet_sensor2, r_feet_sensor3, l_feet_sensor1, l_feet_sensor2, l_feet_sensor3;
char la, lo;


char comma = ',';

//::::::::::::::::::::::::GPS & SD:::::::::::::::::::::::::::::::
char yearGPS[] = "/20";
char twoPoints = ':';
float vel, ang, alt;
float latit, longit;


unsigned long previousMillis = 0;
const long interval = 80;


void setup() {
  Bridge.begin();
  Console.begin();

  //-------------RFID-RC522--------------------------
  SPI.begin();      // Init SPI bus
  mfrc522.PCD_Init();   // Init MFRC522

  //-------------RFID-RC522-------------------------
  //-------------------------XBee--------------------------------
  Serial2.begin(19200);
  xbee.setSerial(Serial2);
  //------------------------------------------------------------
  pinMode(44, OUTPUT);
  //::::::::::::::::::::::::GPS & SD - BEGIN:::::::::::::::::::::::::::::::
  GPS.begin(9600);
  mySerial.begin(9600);
  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
```


Technical Manual

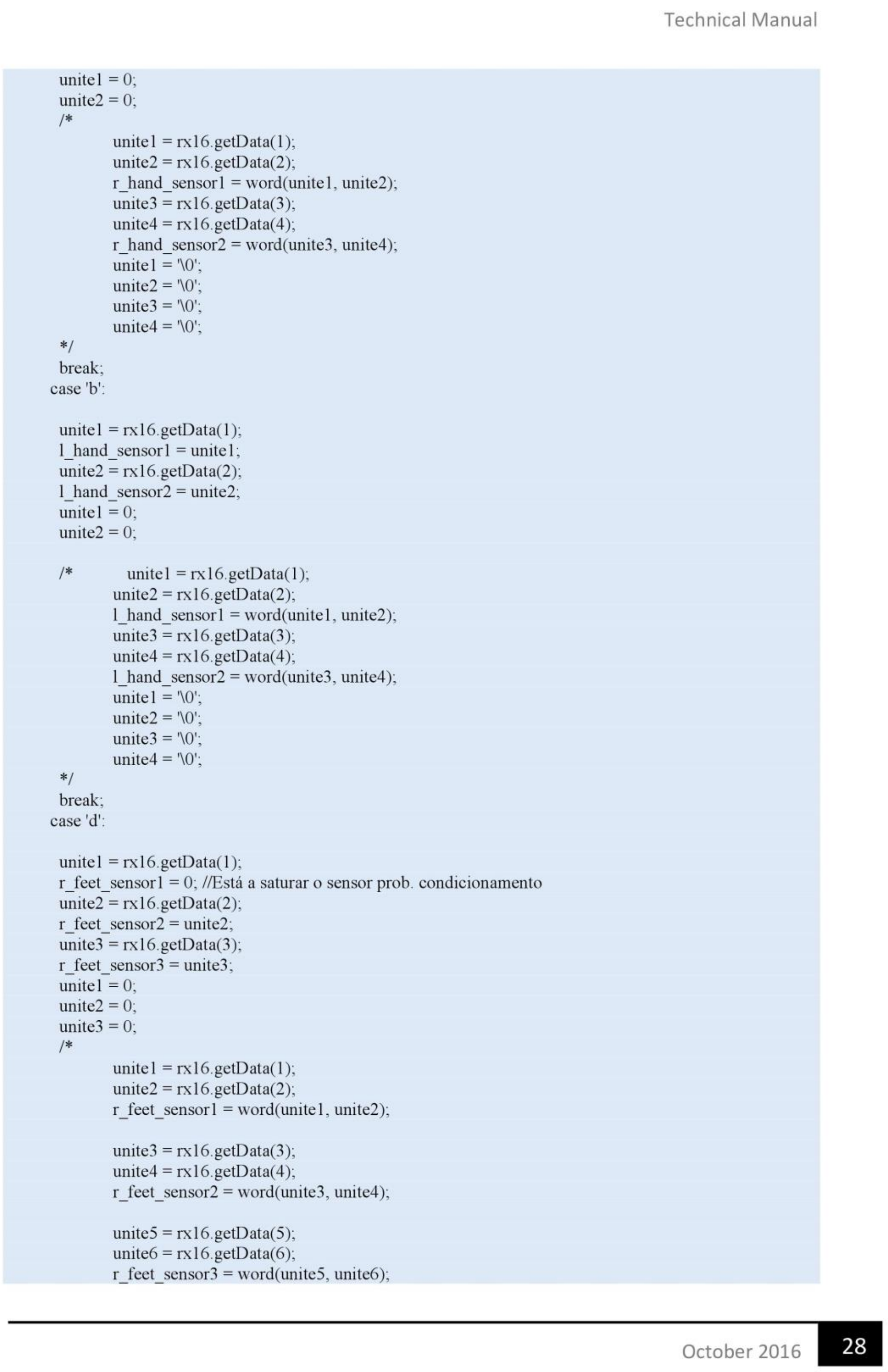
```
 //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
 // For parsing data, we don't suggest using anything but either RMC only or RMC+GGA since
 // the parser doesn't care about other sentences at this time

 // Set the update rate
 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
 // For the parsing code to work nicely and have time to sort thru the data, and
 // print it out we don't suggest using anything higher than 1 Hz

 // Request updates on antenna status, comment out to keep quiet
 GPS.sendCommand(PGCMD_ANTENNA);

 // the nice thing about this code is you can have a timer0 interrupt go off
 // every 1 millisecond, and read data from the GPS for you. that makes the
 // loop code a heck of a lot easier!

#ifdef __arm__
 usingInterrupt = false;  //NOTE - we don't want to use interrupts on the Due
#else
 useInterrupt(true);
#endif

 delay(1000);
 // Ask for firmware version
 mySerial.println(PMTK_Q_RELEASE);
 Console.print("Initializing SD card...");
 // make sure that the default chip select pin is set to
 // output, even if you don't use it:
 pinMode(SS, OUTPUT);

 // see if the card is present and can be initialized:
 if (!SD.begin(10, 11, 12, 13)) {

   // Console.println("Card failed, or not present");
   // don't do anything more:
   while (1) ;

 }

 char filename[] = "datalog.txt";
 if (SD.exists(filename)) {
   SD.remove(filename);
 }

 //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>ID's>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 /*
   int t = 0;
   idFile = SD.open("authids.txt", FILE_READ);
   if (idFile) {
     while (idFile.available()) {
       char ch = idFile.read();
       id[t] = ch;
       t++;
     }
     idFile.close();
   } else {

     Console.println("Can't open File authids.txt");
   }
```

October 2016 24

```
 */

 //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>ID's>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

 int index = 0;
 serverFile = SD.open("server.txt", FILE_READ);
 if (serverFile) {
   while (serverFile.available()) {
     char ch = serverFile.read();
     server[index] = ch;
     index++;
   }
   serverFile.close();
 } else {

   Console.println("Can't open File server.txt");
 }



 // Open up the file we're going to log to!
 dataFile = SD.open("datalog.txt", FILE_WRITE);
 if (! dataFile) {
   Console.println("error opening datalog.txt");
   // Wait forever since we cant write data
   while (1) ;
 }

 //::::::::::::::::::::::::::GPS & SD -END::::::::::::::::::::::::::::::::

 //-------------RFID-RC522--------------------------
 int t = 0;
 idFile = SD.open("authids.txt", FILE_READ);
 if (idFile) {
   while (idFile.available()) {

     ids[t] = idFile.parseInt();

     t++;

   }
   idFile.close();
 } else {

   Console.println("Can't open File authids.txt");
 }

 //-------------RFID-RC522--------------------------
}

#ifdef __AVR__
// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
 char c = GPS.read();
 // if you want to debug, this is a good time to do it!
#ifdef UDR0
 if (GPSECHO)
   if (c) UDR0 = c;
 // writing direct to UDR0 is much much faster than Serial.print
```

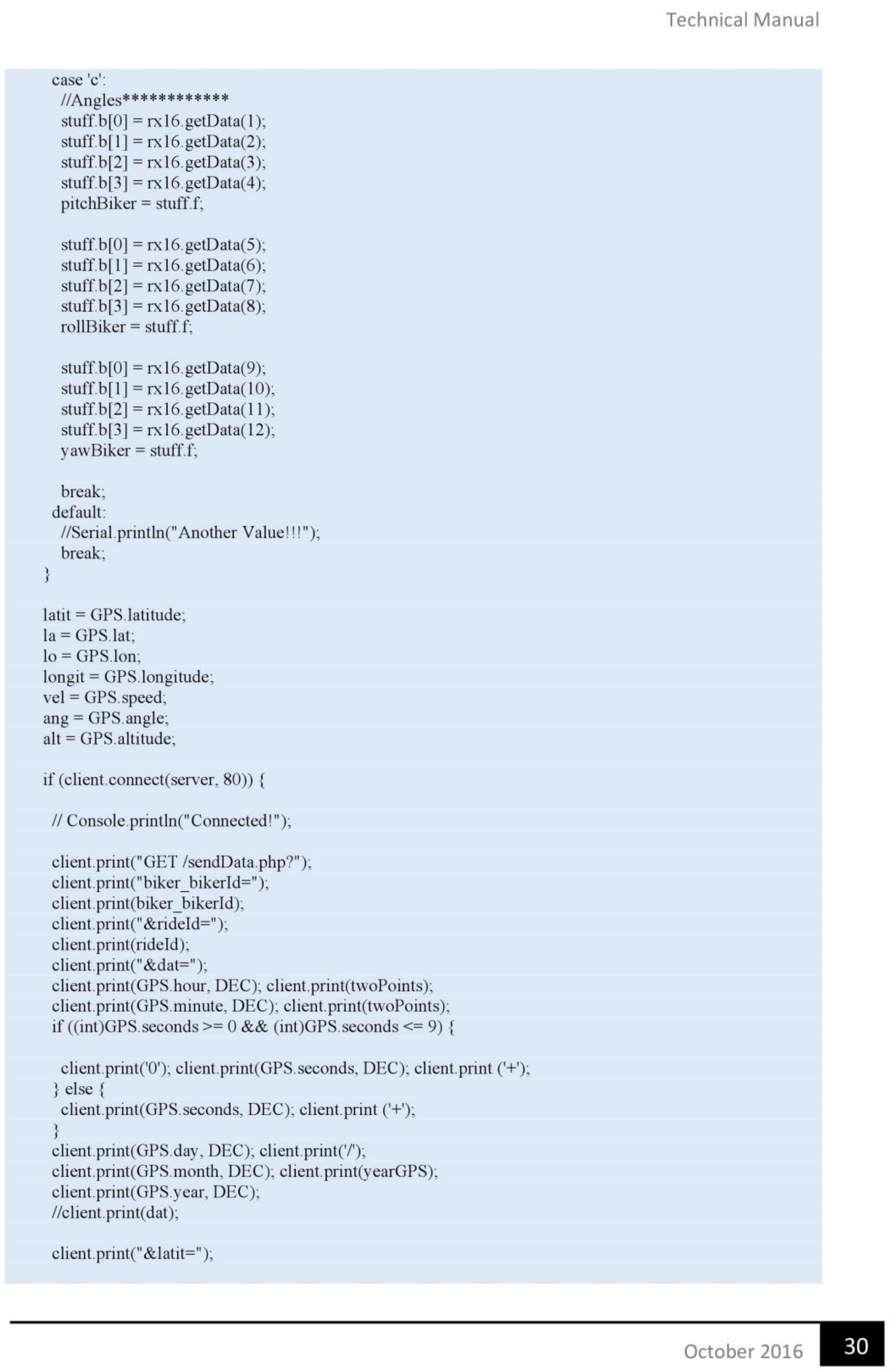
```
  // but only one character can be written at a time.
#endif
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}
#endif //#ifdef__AVR__

void loop() {
  // unsigned long currentMillis = millis();

  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag to false

    if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag to false
      return;  // we can fail to parse a sentence in which case we should just wait for another
  }
  // unsigned long currentMillis = millis();
  YunClient client;
  char incomingByte;
  byte unite1, unite2, unite3, unite4, unite5, unite6;

  //-------------RFID-RC522-------------------------
  if (!autenthicate) {
    if ( ! mfrc522.PICC_IsNewCardPresent()) {
      return;
    }
    // Select one of the cards
    if ( ! mfrc522.PICC_ReadCardSerial()) {
      return;
    }
    /*
      // Dump debug info about the card; PICC_HaltA() is automatically called
      mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
    */
    unsigned int hex_num;
    hex_num =  mfrc522.uid.uidByte[0] << 24;
    hex_num += mfrc522.uid.uidByte[1] << 16;
    hex_num += mfrc522.uid.uidByte[2] <<  8;
    hex_num += mfrc522.uid.uidByte[3];

    int  NFC_id =  (int)hex_num;

    for (int i = 0; i <= arraySize; i++ ) {
      if (ids[i] == NFC_id) {
```

```
      biker_bikerId = i;
      Console.print("found id on array:");
      Console.println(ids[i]);
      Console.print("Bike id:");
      Console.println(biker_bikerId);
    }
  }
  delay(500);

  if (client.connect(server, 80)) {
    client.print("GET /getRiderId.php?biker_bikerId=");
    client.println(biker_bikerId);
    delay(2500);
    while (client.available() >0) {
      rideId = client.parseInt();
    }
    client.stop();
  } else {
    Console.println("Connection Failed!");
  }

  rideId = rideId+1;
  autenthicate = true;

 } else {
  //-------------RFID-RC522-------------------------

  // ***
 if (GPS.fix ) {
  xbee.readPacket();

  if (xbee.getResponse().isAvailable()) {
    // got something

    if (xbee.getResponse().getApiId() == RX_16_RESPONSE || xbee.getResponse().getApiId() ==
RX_64_RESPONSE) {
      // got a rx packet

      if (xbee.getResponse().getApiId() == RX_16_RESPONSE) {
        xbee.getResponse().getRx16Response(rx16);
        option = rx16.getOption();
        data = rx16.getData(0);
      } else {
        xbee.getResponse().getRx64Response(rx64);
        option = rx64.getOption();
        data = rx64.getData(0);

      }

      //Led Sincronização
      digitalWrite(44, HIGH);

      switch (incomingByte = char(rx16.getData(0)) ) {
        case 'a':

          unite1 = rx16.getData(1);
          r_hand_sensor1 = unite1;
          unite2 = rx16.getData(2);
          r_hand_sensor2 = unite2;
```

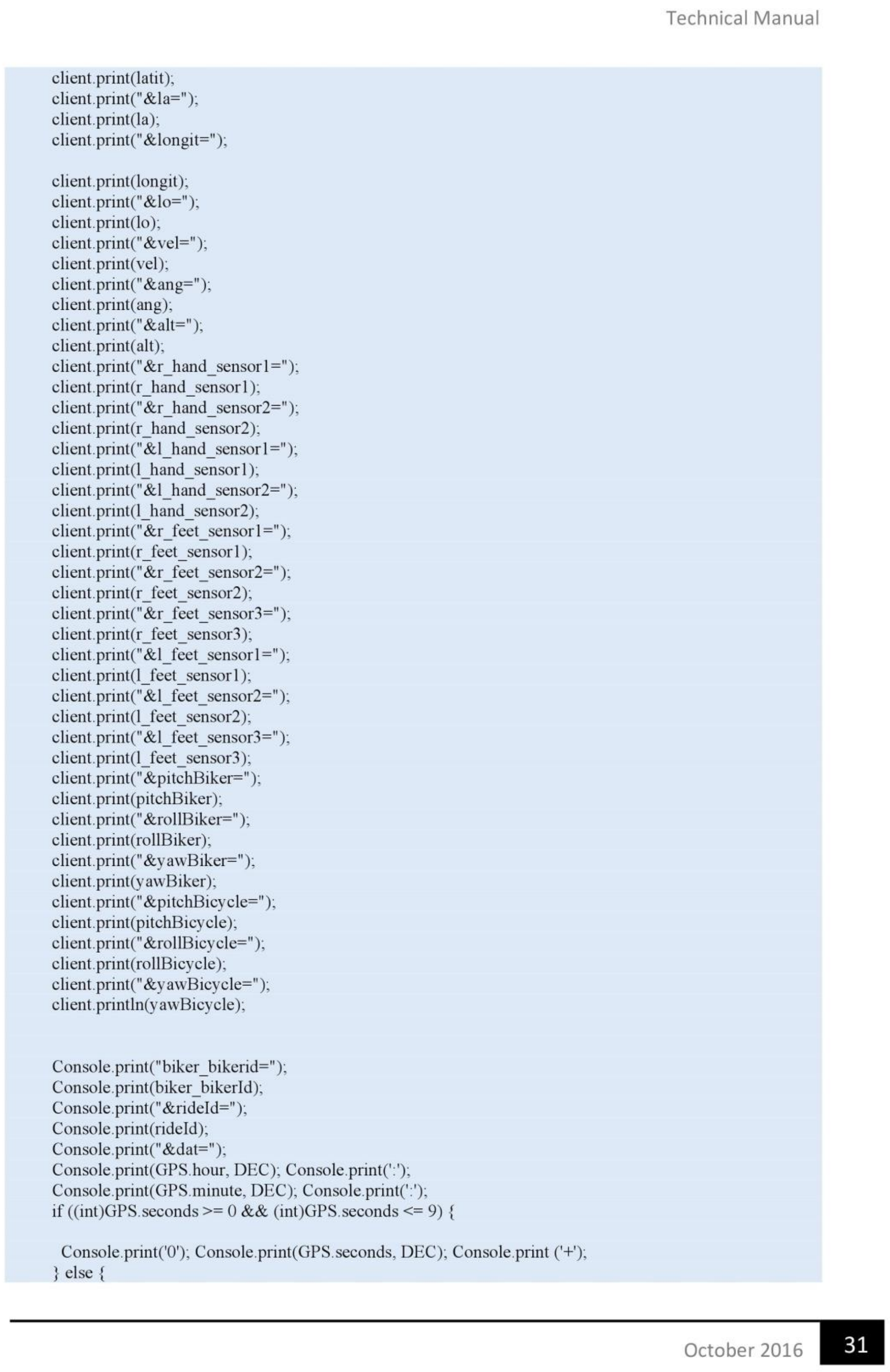


```
  unite1 = 0;
  unite2 = 0;
  /*
          unite1 = rx16.getData(1);
          unite2 = rx16.getData(2);
          r_hand_sensor1 = word(unite1, unite2);
          unite3 = rx16.getData(3);
          unite4 = rx16.getData(4);
          r_hand_sensor2 = word(unite3, unite4);
          unite1 = '\0';
          unite2 = '\0';
          unite3 = '\0';
          unite4 = '\0';
  */
  break;
case 'b':

  unite1 = rx16.getData(1);
  l_hand_sensor1 = unite1;
  unite2 = rx16.getData(2);
  l_hand_sensor2 = unite2;
  unite1 = 0;
  unite2 = 0;

  /*        unite1 = rx16.getData(1);
          unite2 = rx16.getData(2);
          l_hand_sensor1 = word(unite1, unite2);
          unite3 = rx16.getData(3);
          unite4 = rx16.getData(4);
          l_hand_sensor2 = word(unite3, unite4);
          unite1 = '\0';
          unite2 = '\0';
          unite3 = '\0';
          unite4 = '\0';
  */
  break;
case 'd':

  unite1 = rx16.getData(1);
  r_feet_sensor1 = 0; //Está a saturar o sensor prob. condicionamento
  unite2 = rx16.getData(2);
  r_feet_sensor2 = unite2;
  unite3 = rx16.getData(3);
  r_feet_sensor3 = unite3;
  unite1 = 0;
  unite2 = 0;
  unite3 = 0;
  /*
          unite1 = rx16.getData(1);
          unite2 = rx16.getData(2);
          r_feet_sensor1 = word(unite1, unite2);

          unite3 = rx16.getData(3);
          unite4 = rx16.getData(4);
          r_feet_sensor2 = word(unite3, unite4);

          unite5 = rx16.getData(5);
          unite6 = rx16.getData(6);
          r_feet_sensor3 = word(unite5, unite6);
```

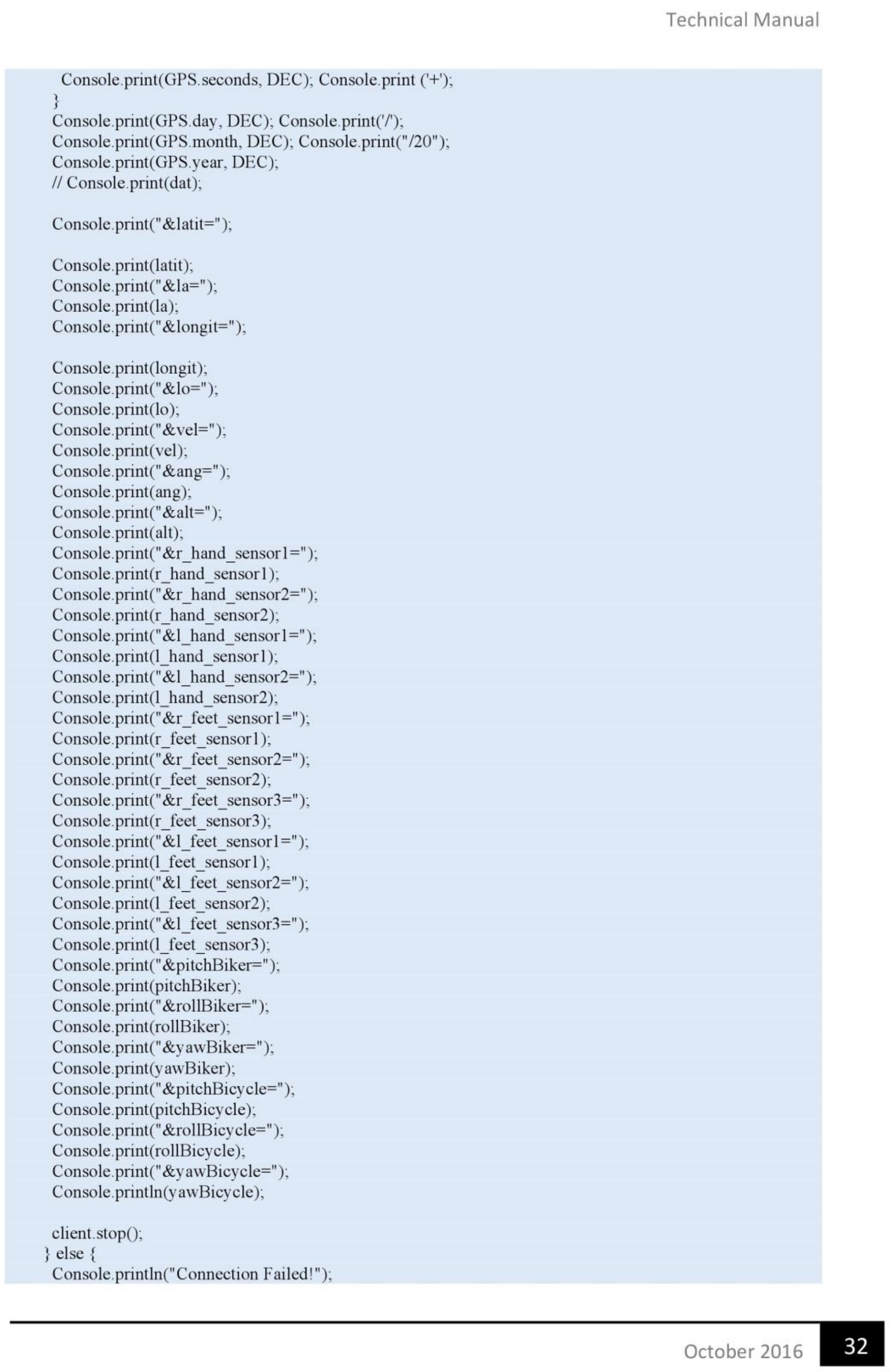
```
            unite1 = '\0';
            unite2 = '\0';
            unite3 = '\0';
            unite4 = '\0';
            unite5 = '\0';
            unite6 = '\0';
    */
    break;
  case 'e':
    unite1 = rx16.getData(1);
    l_feet_sensor1 = unite1;
    unite2 = rx16.getData(2);
    l_feet_sensor2 = unite2;
    unite3 = rx16.getData(3);
    l_feet_sensor3 = unite3;
    unite1 = 0;
    unite2 = 0;
    unite3 = 0;
    /*
            unite1 = rx16.getData(1);
            unite2 = rx16.getData(2);
            l_feet_sensor1 = word(unite1, unite2);
            unite3 = rx16.getData(3);
            unite4 = rx16.getData(4);
            l_feet_sensor2 = word(unite3, unite4);
            unite5 = rx16.getData(5);
            unite6 = rx16.getData(6);
            l_feet_sensor3 = word(unite5, unite6);
            unite1 = '\0';
            unite2 = '\0';
            unite3 = '\0';
            unite4 = '\0';
            unite5 = '\0';
            unite6 = '\0';
    */
    break;

  case 'f':
    //Angles************
    stuff.b[0] = rx16.getData(1);
    stuff.b[1] = rx16.getData(2);
    stuff.b[2] = rx16.getData(3);
    stuff.b[3] = rx16.getData(4);
    pitchBicycle = stuff.f;

    stuff.b[0] = rx16.getData(5);
    stuff.b[1] = rx16.getData(6);
    stuff.b[2] = rx16.getData(7);
    stuff.b[3] = rx16.getData(8);
    rollBicycle = stuff.f;

    stuff.b[0] = rx16.getData(9);
    stuff.b[1] = rx16.getData(10);
    stuff.b[2] = rx16.getData(11);
    stuff.b[3] = rx16.getData(12);
    yawBicycle = stuff.f;

    break;
```



```
  case 'c':
    //Angles************
    stuff.b[0] = rx16.getData(1);
    stuff.b[1] = rx16.getData(2);
    stuff.b[2] = rx16.getData(3);
    stuff.b[3] = rx16.getData(4);
    pitchBiker = stuff.f;

    stuff.b[0] = rx16.getData(5);
    stuff.b[1] = rx16.getData(6);
    stuff.b[2] = rx16.getData(7);
    stuff.b[3] = rx16.getData(8);
    rollBiker = stuff.f;

    stuff.b[0] = rx16.getData(9);
    stuff.b[1] = rx16.getData(10);
    stuff.b[2] = rx16.getData(11);
    stuff.b[3] = rx16.getData(12);
    yawBiker = stuff.f;

    break;
  default:
    //Serial.println("Another Value!!!");
    break;
}

latit = GPS.latitude;
la = GPS.lat;
lo = GPS.lon;
longit = GPS.longitude;
vel = GPS.speed;
ang = GPS.angle;
alt = GPS.altitude;

if (client.connect(server, 80)) {

  // Console.println("Connected!");

  client.print("GET /sendData.php?");
  client.print("biker_bikerId=");
  client.print(biker_bikerId);
  client.print("&rideId=");
  client.print(rideId);
  client.print("&dat=");
  client.print(GPS.hour, DEC); client.print(twoPoints);
  client.print(GPS.minute, DEC); client.print(twoPoints);
  if ((int)GPS.seconds >= 0 && (int)GPS.seconds <= 9) {

    client.print('0'); client.print(GPS.seconds, DEC); client.print ('+');
  } else {
    client.print(GPS.seconds, DEC); client.print ('+');
  }
  client.print(GPS.day, DEC); client.print('/');
  client.print(GPS.month, DEC); client.print(yearGPS);
  client.print(GPS.year, DEC);
  //client.print(dat);

  client.print("&latit=");
```



```
client.print(latit);
client.print("&la=");
client.print(la);
client.print("&longit=");

client.print(longit);
client.print("&lo=");
client.print(lo);
client.print("&vel=");
client.print(vel);
client.print("&ang=");
client.print(ang);
client.print("&alt=");
client.print(alt);
client.print("&r_hand_sensor1=");
client.print(r_hand_sensor1);
client.print("&r_hand_sensor2=");
client.print(r_hand_sensor2);
client.print("&l_hand_sensor1=");
client.print(l_hand_sensor1);
client.print("&l_hand_sensor2=");
client.print(l_hand_sensor2);
client.print("&r_feet_sensor1=");
client.print(r_feet_sensor1);
client.print("&r_feet_sensor2=");
client.print(r_feet_sensor2);
client.print("&r_feet_sensor3=");
client.print(r_feet_sensor3);
client.print("&l_feet_sensor1=");
client.print(l_feet_sensor1);
client.print("&l_feet_sensor2=");
client.print(l_feet_sensor2);
client.print("&l_feet_sensor3=");
client.print(l_feet_sensor3);
client.print("&pitchBiker=");
client.print(pitchBiker);
client.print("&rollBiker=");
client.print(rollBiker);
client.print("&yawBiker=");
client.print(yawBiker);
client.print("&pitchBicycle=");
client.print(pitchBicycle);
client.print("&rollBicycle=");
client.print(rollBicycle);
client.print("&yawBicycle=");
client.println(yawBicycle);


Console.print("biker_bikerid=");
Console.print(biker_bikerId);
Console.print("&rideId=");
Console.print(rideId);
Console.print("&dat=");
Console.print(GPS.hour, DEC); Console.print(':');
Console.print(GPS.minute, DEC); Console.print(':');
if ((int)GPS.seconds >= 0 && (int)GPS.seconds <= 9) {

  Console.print('0'); Console.print(GPS.seconds, DEC); Console.print ('+');
} else {
```



```
      Console.print(GPS.seconds, DEC); Console.print ('+');
    }
    Console.print(GPS.day, DEC); Console.print('/');
    Console.print(GPS.month, DEC); Console.print("/20");
    Console.print(GPS.year, DEC);
    // Console.print(dat);

    Console.print("&latit=");

    Console.print(latit);
    Console.print("&la=");
    Console.print(la);
    Console.print("&longit=");

    Console.print(longit);
    Console.print("&lo=");
    Console.print(lo);
    Console.print("&vel=");
    Console.print(vel);
    Console.print("&ang=");
    Console.print(ang);
    Console.print("&alt=");
    Console.print(alt);
    Console.print("&r_hand_sensor1=");
    Console.print(r_hand_sensor1);
    Console.print("&r_hand_sensor2=");
    Console.print(r_hand_sensor2);
    Console.print("&l_hand_sensor1=");
    Console.print(l_hand_sensor1);
    Console.print("&l_hand_sensor2=");
    Console.print(l_hand_sensor2);
    Console.print("&r_feet_sensor1=");
    Console.print(r_feet_sensor1);
    Console.print("&r_feet_sensor2=");
    Console.print(r_feet_sensor2);
    Console.print("&r_feet_sensor3=");
    Console.print(r_feet_sensor3);
    Console.print("&l_feet_sensor1=");
    Console.print(l_feet_sensor1);
    Console.print("&l_feet_sensor2=");
    Console.print(l_feet_sensor2);
    Console.print("&l_feet_sensor3=");
    Console.print(l_feet_sensor3);
    Console.print("&pitchBiker=");
    Console.print(pitchBiker);
    Console.print("&rollBiker=");
    Console.print(rollBiker);
    Console.print("&yawBiker=");
    Console.print(yawBiker);
    Console.print("&pitchBicycle=");
    Console.print(pitchBicycle);
    Console.print("&rollBicycle=");
    Console.print(rollBicycle);
    Console.print("&yawBicycle=");
    Console.println(yawBicycle);

    client.stop();
  } else {
    Console.println("Connection Failed!");
```

```
        client.stop();
       }

       printInSD();
       dataFile.flush();
     } else {
       // not something we were expecting
       //  flashLed(errorLed, 1, 25);
     }
   } else if (xbee.getResponse().isError()) {
     //nss.print("Error reading packet.  Error code: ");
     //nss.println(xbee.getResponse().getErrorCode());
     // or flash error led
   }
  }
}
}
```

In the programming of the RFID sensor, at the moment it is possible to have twelve bikers registered and this are defined in the following lines of codes:

```
const int arraySize = 11;
File idFile;
int ids[arraySize];
```

But if there is the need to register more bikers, the only need is to increase the variable arraySize. For example, if the objective is to increment the double of the bikers the changes are as follows:

```
const int arraySize = 23;
File idFile;
int ids[arraySize];
```

With the increase of this variable the file authids.txt will support twenty-four registered bikers.

4. Cloud Server

The Server is responsible for the storage of every training performed by each biker. There is the place were the data is all saved for further analysis through a mobile application, this is, the application access to the webserver through php modules and present the data in the application with a user friendly view. The main objective is to centralize all the information in one place. This could be a problem if something went wrong with the server, but some mechanisms were created, like a local database and the possible in the mobile application to export each training.

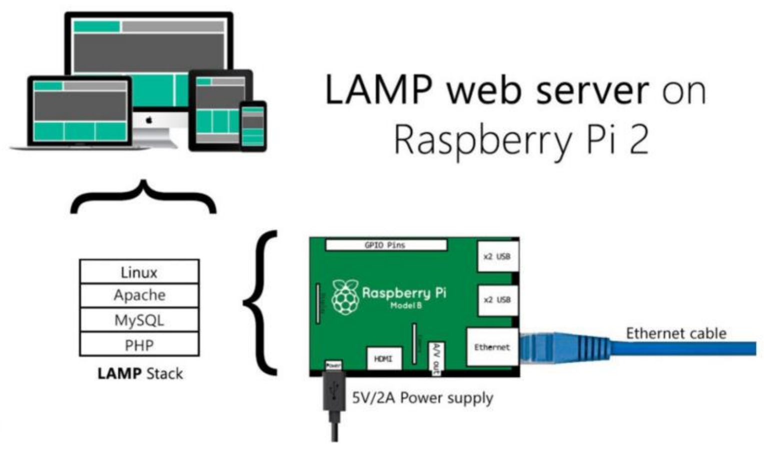


Figure 16 - LAMP Web Server Overview

The LAMP term is one of the most common configuration for webservers which standard for:

Linux – operation System;

Apache- webserver (http) software;

Mysql – database server;

PHP or Perl – programming languages;

The main reason to use the raspberry pi as a webserver is because he is dedicated network device, and there is a equipment very accessible in terms of costs and efficiency, making is job perfectly. The operating system and all mechanisms needed is freeware.

4.1. Setup of the LAMP Server "Cloud"

To the following setup of the Cloud is necessary the operation system, the WebServer one mechanism to create databases and one language to guarantee the connection and insertion of data in the database.

4.1.1. WebServer

To have a cloud, it is need to have a webserver to guarantee that the information is accessible in the Internet.

```
sudo apt-get install apache2
```

4.1.2. Mysql

The MySQL is an open-source relational database management system. MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open-source web application software.

```
sudo apt-get install mysql-server
```

4.1.3. PHP

Is a server-side scripting language designed for web development but also used as a general-purpose programming language. PHP code may be embedded into HTML code, or it can be used in combination with various web template systems. The following commands will install PHP version 5 and the MySQL libraries to allow PHP to access the MySQL database.

```
sudo apt-get install php5
sudo apt-get install php5-mysql
```

After all setup is complete is possible to access to the WebServer through a remote connection ssh (Secure Shell).

The Database was created in the program MySQL workbench, to allow visualization of the entire DB:

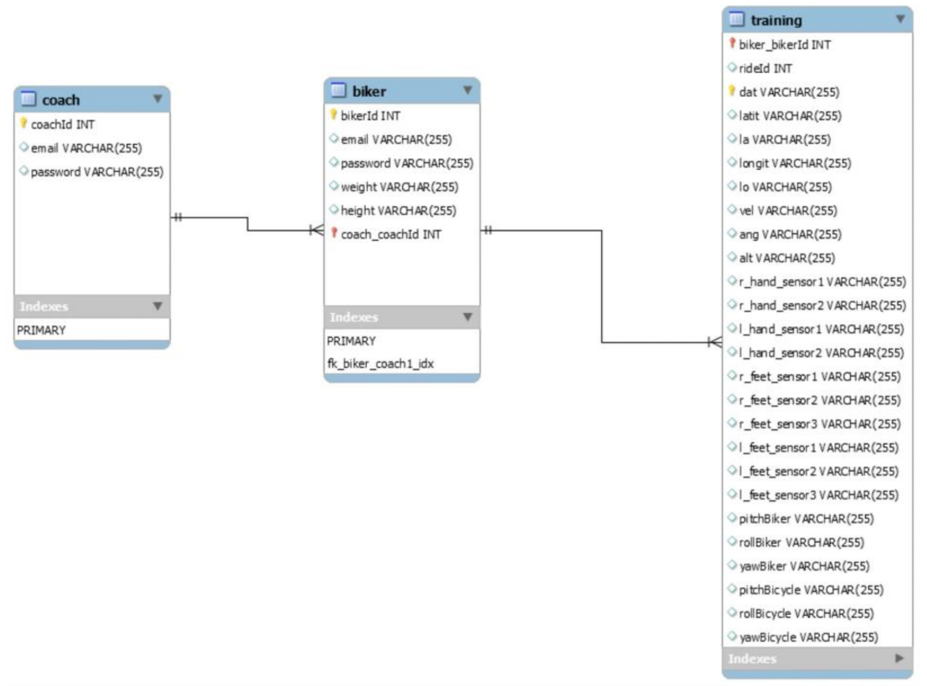


Figure 17 - Database mountainBikeDB

The DB and the above tables were created in the Database with the following Script in SQL:

```
CREATE SCHEMA IF NOT EXISTS `mountainBikeDB` DEFAULT CHARACTER SET utf8 ;
USE `mountainBikeDB` ;
-- -----------------------------------------------------
-- Table `mountainBikeDB`.`coach`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mountainBikeDB`.`coach` (
  `coachId` INT NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(255) NULL,
  `password` VARCHAR(255) NULL,
  PRIMARY KEY (`coachId`))
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `mountainBikeDB`.`biker`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mountainBikeDB`.`biker` (
  `bikerId` INT NOT NULL,
  `email` VARCHAR(255) NULL,
  `password` VARCHAR(255) NULL,
  `weight` VARCHAR(255) NULL,
  `height` VARCHAR(255) NULL,
  `coach_coachId` INT NOT NULL,
  PRIMARY KEY (`bikerId`, `coach_coachId`),
  INDEX `fk_biker_coach1_idx` (`coach_coachId` ASC),
  CONSTRAINT `fk_biker_coach1`
```

```
    FOREIGN KEY (`coach_coachId`)
    REFERENCES `mountainBikeDB`.`coach` (`coachId`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `mountainBikeDB`.`training`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mountainBikeDB`.`training` (
  `biker_bikerId` INT NOT NULL,
  `rideId` INT NULL,
  `dat` VARCHAR(255) NOT NULL,
  `latit` VARCHAR(255) NULL,
  `la` VARCHAR(255) NULL,
  `longit` VARCHAR(255) NULL,
  `lo` VARCHAR(255) NULL,
  `vel` VARCHAR(255) NULL,
  `ang` VARCHAR(255) NULL,
  `alt` VARCHAR(255) NULL,
  `r_hand_sensor1` VARCHAR(255) NULL,
  `r_hand_sensor2` VARCHAR(255) NULL,
  `l_hand_sensor1` VARCHAR(255) NULL,
  `l_hand_sensor2` VARCHAR(255) NULL,
  `r_feet_sensor1` VARCHAR(255) NULL,
  `r_feet_sensor2` VARCHAR(255) NULL,
  `r_feet_sensor3` VARCHAR(255) NULL,
  `l_feet_sensor1` VARCHAR(255) NULL,
  `l_feet_sensor2` VARCHAR(255) NULL,
  `l_feet_sensor3` VARCHAR(255) NULL,
  `pitchBiker` VARCHAR(255) NULL,
  `rollBiker` VARCHAR(255) NULL,
  `yawBiker` VARCHAR(255) NULL,
  `pitchBicycle` VARCHAR(255) NULL,
  `rollBicycle` VARCHAR(255) NULL,
  `yawBicycle` VARCHAR(255) NULL,
  PRIMARY KEY (`biker_bikerId`, `dat`),
  INDEX `fk_training_biker1_idx` (`biker_bikerId` ASC),
  CONSTRAINT `fk_training_biker1`
    FOREIGN KEY (`biker_bikerId`)
    REFERENCES `mountainBikeDB`.`biker` (`bikerId`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

All the php files are in the the folder /var/www/html/.

This php files are housed in the cloud, every time is needed to get some information from database or to insert some information these files are called. In the coordinator is used two files:

- getRiderId.php: This file is important to know how many trainings already was performed. Let's see a simple case one biker already perform two trainings, he wants to perform another one, this simple php file goes to the database and see the number of last training.
- sendData.php: Like name indicates this file will send all data collected from sensors.

In case of the application there are a few files that are important for the proper functioning of the application:

- connectToDB.php: Perform the connection to the database, every single php file uses this file.
- login.php: When a coach tries to access to the application, this will see if the coach is registered in the database.
- loginBiker.php: A biker tries to access to the application, this file will see if the biker is registered on the database. It's important that the biker is registered to perform a training.
- createBiker.php: Entering in the application the user will see all is bikers and has the option to register others.
- getCoachIdUsingEmail.php: At the beginning, the users enter is username (email) and with this information the file will get the CoachId.
- getCoachEmailByUsingCoachId.php: Get the email using the CoachId.
- getBikersByCoachId.php: This file will retrieve a list of the bikers that a coach is responsible.
- getTrainingArrayByBikerId.php: Will retrieve a list in a json format of all training performed for a given BikerId.
- getTrainingArrayByRiderId.php: When the coach or biker choose which training want to analyze this will get the individual training number performed.

  All this programming could be done in a single file, but for a good comprehension and good debugging it was decided to make it by step by step. If any problem occurs is easy to detect the fault.

5. Android application Mountain Bike App

The application is to allow very friendly access to each training session, with this application every coach and biker can access to every training session that already perform.

5.1. Sequence Diagram

The sequence of activities in the application have two options, depending which user is logged Coach or Biker.

5.1.1. Coach Sequence Diagram

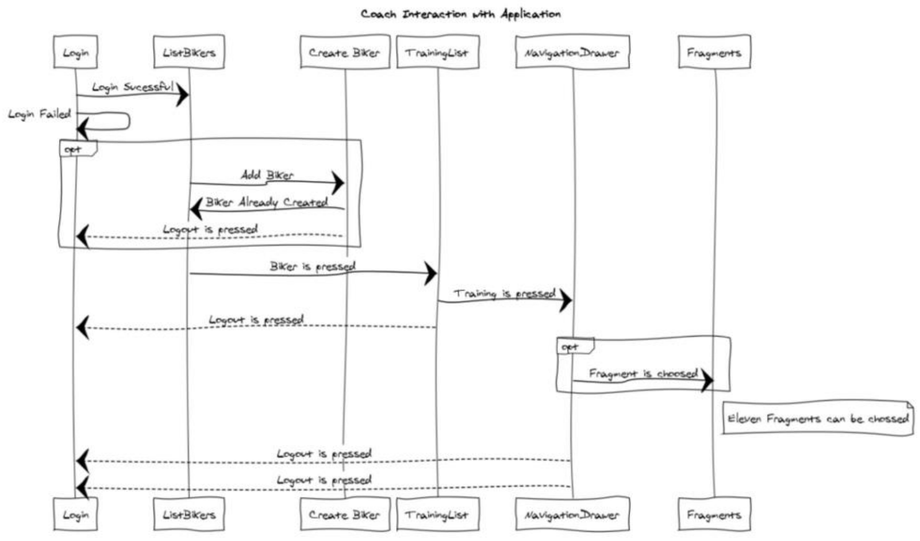


Figure 18 - Sequence Diagram (Coach interaction)

5.1.2. Biker Sequence Diagram

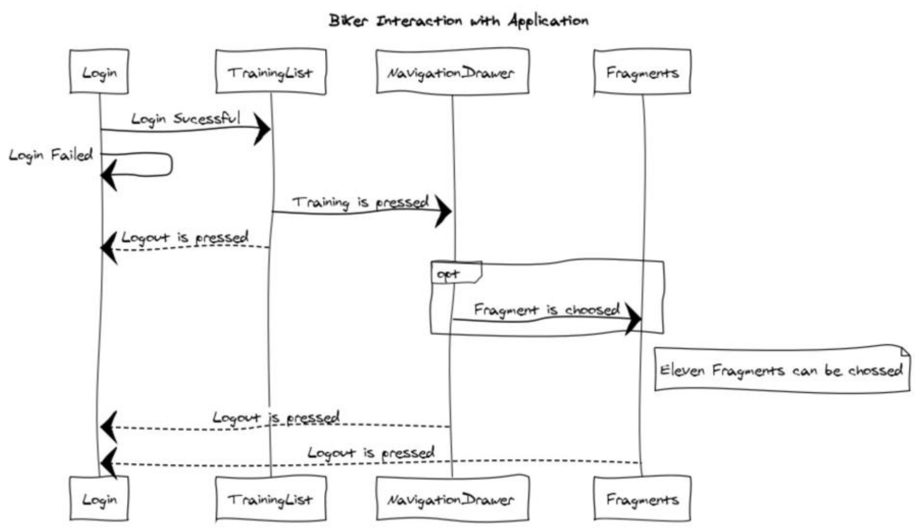


Figure 19 - Diagram Sequence (Biker interaction)

Well in the diagrams is possible to see every interaction that a coach or a user may make in the mobile application. In the login Activity the user must choose if he had coach or biker account. By default, is selected the coach account.

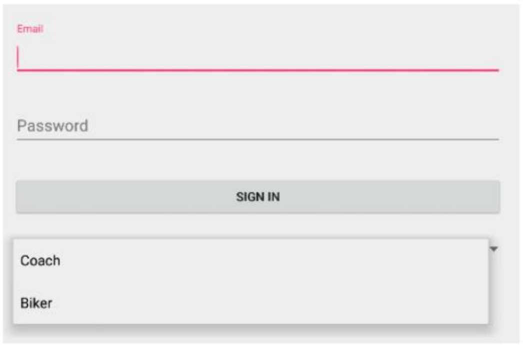


Figure 20 - Login in Mountain Bike App

Depending of this choice the mobile application will behave according to the diagrams presented. In the case of the fragments is where the analyze is performed, this is, the report and graphs are presented in the Fragment in total there is eleven fragments.

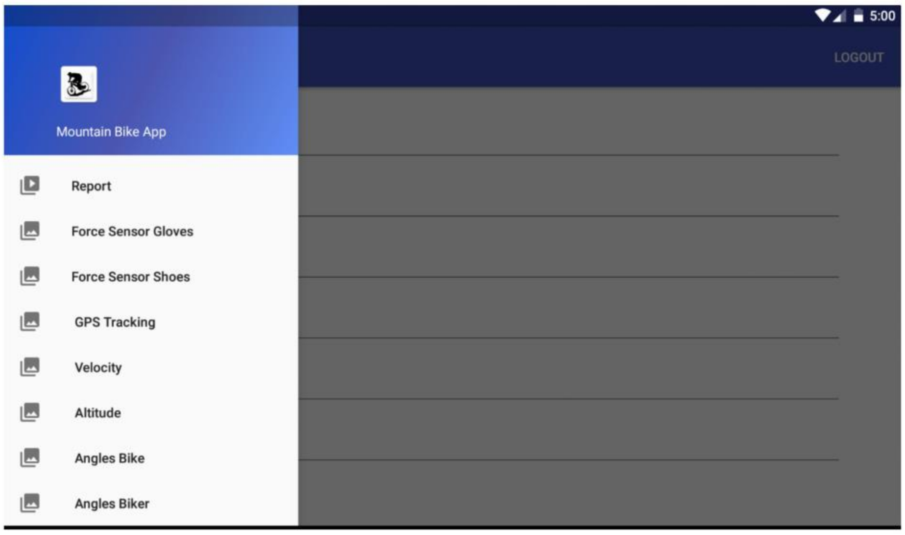
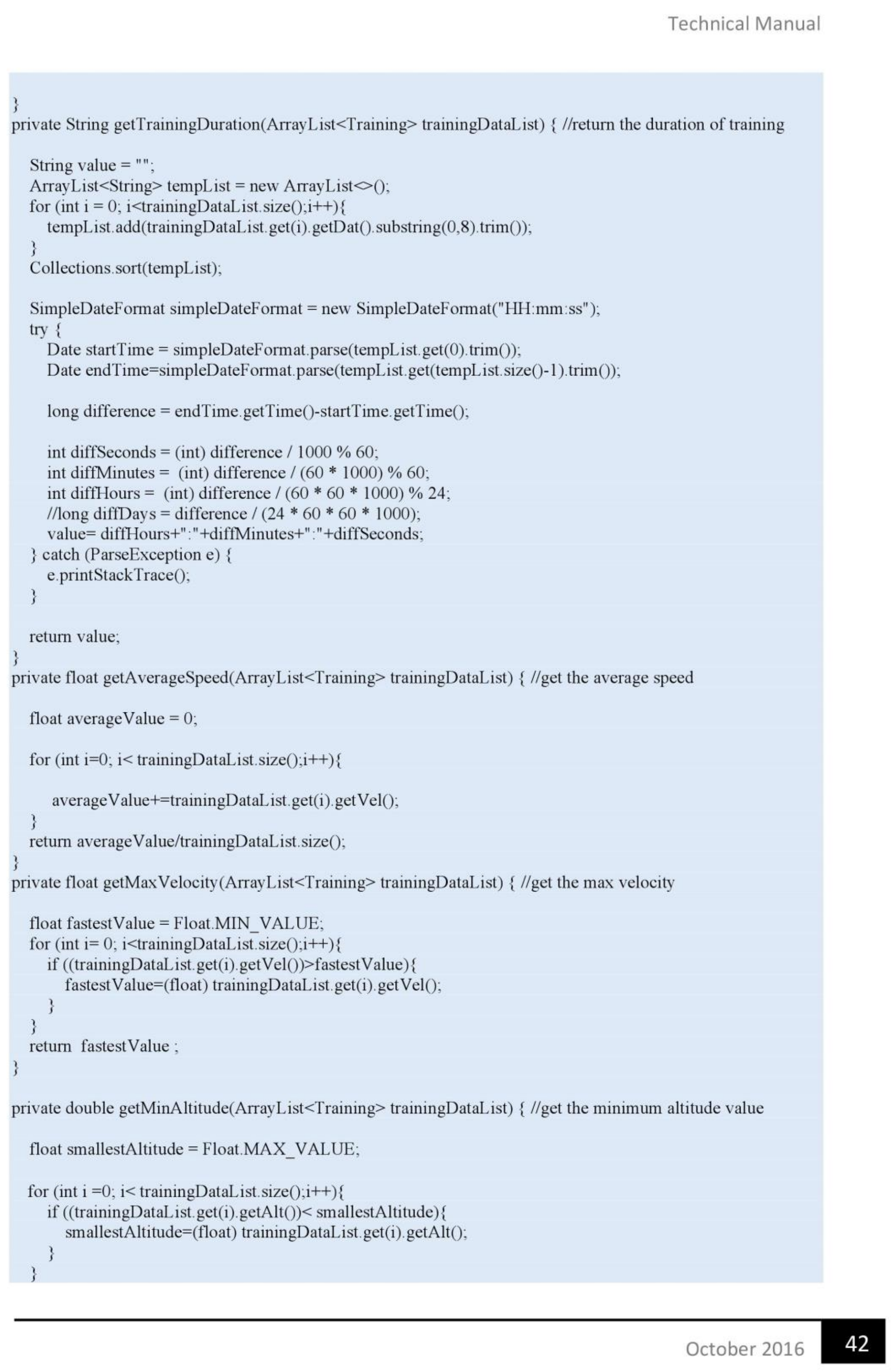


Figure 21 - Navigation Drawer (Show Fragments)

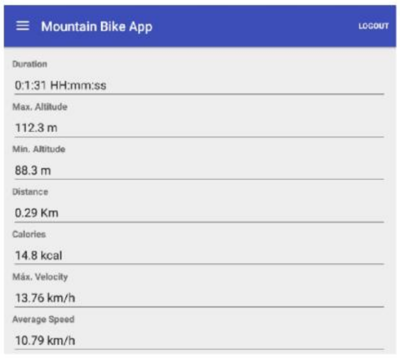
1. Report: present some results like the duration, maximum altitude, minimum altitude, distance, calories, maximum velocity and average speed;

Here is presented some of the functions that permit to calculate some values about the training session performed by the bikers. All the data is collected from the database.

```
private float getCaloriesBurned(ArrayList<Training> trainingDataList) { //calculate the time spent in hours

  float result=0;
  ArrayList<String> tempList = new ArrayList<>();
  for (int i = 0; i<trainingDataList.size();i++){
    tempList.add(trainingDataList.get(i).getDat().substring(0,8).trim());

  }
  Collections.sort(tempList);

  SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
  try {
    Date startTime = simpleDateFormat.parse(tempList.get(0).trim());
    Date endTime=simpleDateFormat.parse(tempList.get(tempList.size()-1).trim());
    long difference = endTime.getTime()-startTime.getTime();

    int diffSeconds = (int) difference / 1000 % 60;
    int diffMinutes =  (int) difference / (60 * 1000) % 60;
    int diffHours =  (int) difference / (60 * 60 * 1000) % 24;
    //long diffDays = difference / (24 * 60 * 60 * 1000);
    result= diffHours+diffMinutes*MINUTETOHOUR+diffSeconds*SECONDSTOHOUR;
  } catch (ParseException e) {
    e.printStackTrace();
  }
  return result;
```



```
}
private String getTrainingDuration(ArrayList<Training> trainingDataList) { //return the duration of training

    String value = "";
    ArrayList<String> tempList = new ArrayList<>();
    for (int i = 0; i<trainingDataList.size();i++){
        tempList.add(trainingDataList.get(i).getDat().substring(0,8).trim());
    }
    Collections.sort(tempList);

    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
    try {
        Date startTime = simpleDateFormat.parse(tempList.get(0).trim());
        Date endTime=simpleDateFormat.parse(tempList.get(tempList.size()-1).trim());

        long difference = endTime.getTime()-startTime.getTime();

        int diffSeconds = (int) difference / 1000 % 60;
        int diffMinutes =  (int) difference / (60 * 1000) % 60;
        int diffHours =  (int) difference / (60 * 60 * 1000) % 24;
        //long diffDays = difference / (24 * 60 * 60 * 1000);
        value= diffHours+":"+diffMinutes+":"+diffSeconds;
    } catch (ParseException e) {
        e.printStackTrace();
    }

    return value;
}
private float getAverageSpeed(ArrayList<Training> trainingDataList) { //get the average speed

    float averageValue = 0;

    for (int i=0; i< trainingDataList.size();i++){

         averageValue+=trainingDataList.get(i).getVel();
    }
    return averageValue/trainingDataList.size();
}
private float getMaxVelocity(ArrayList<Training> trainingDataList) { //get the max velocity

    float fastestValue = Float.MIN_VALUE;
    for (int i= 0; i<trainingDataList.size();i++){
        if ((trainingDataList.get(i).getVel())>fastestValue){
            fastestValue=(float) trainingDataList.get(i).getVel();
        }
    }
    return  fastestValue ;
}

private double getMinAltitude(ArrayList<Training> trainingDataList) { //get the minimum altitude value

    float smallestAltitude = Float.MAX_VALUE;

   for (int i =0; i< trainingDataList.size();i++){
        if ((trainingDataList.get(i).getAlt())< smallestAltitude){
            smallestAltitude=(float) trainingDataList.get(i).getAlt();
        }
    }
    return smallestAltitude;
}

private float getMaxAltitude(ArrayList<Training> trainingDataList) {//get the maximum altitude value
    float biggestAltitude = Float.MIN_VALUE;

    for (int i=0; i< trainingDataList.size();i++){
        if (( trainingDataList.get(i).getAlt())>biggestAltitude){
            biggestAltitude=(float) trainingDataList.get(i).getAlt();
        }
    }

    return biggestAltitude;
}
private float convertDegreeMinuteMToDecimal(float value) { //convert the latitude and longitude to degrees

    int indexOfPoint;
    indexOfPoint = new Double(value).toString().indexOf('.');

    String value1 = String.valueOf(value);

    String aux1=value1.substring(0,indexOfPoint-2);
    String aux2=value1.substring(indexOfPoint-2,value1.length());

    float result=Float.parseFloat(aux1)+(Float.parseFloat(aux2)/60);

    return result;
}

private float getDistance( ArrayList<Training> listOfRides){ //get distance of the biker

    ArrayList<Training> tempList = new ArrayList<>();
    double distance = 0;
    for (int i =0; i<listOfRides.size();i++){
        Training training = new Training(listOfRides.get(i).getDat(),convertDegreeMinuteMToDecimal((float)
listOfRides.get(i).getLatit()),listOfRides.get(i).getLa(),convertDegreeMinuteMToDecimal((float)
listOfRides.get(i).getLongit()),listOfRides.get(i).getLo());
        tempList.add(training);

    }
    Collections.sort(tempList, new TrainingComparator());
    for (int i =0; i<tempList.size();i++){
        // Log.d("tempListDat",""+tempList.get(i).getDat()+" "+tempList.get(i).getLatit()+"
"+tempList.get(i).getLongit());
        if (i!=0){

            float dLat= (float) Math.toRadians(tempList.get(i).getLatit()-tempList.get(i-1).getLatit());
            float dLong = (float) Math.toRadians(tempList.get(i).getLongit()-tempList.get(i-1).getLongit());

            float a = (float) (Math.sin(dLat/2)*Math.sin(dLat/2)+Math.cos(Math.toRadians(tempList.get(i-
1).getLatit()))*Math.cos(Math.toRadians(tempList.get(i).getLatit()))*
                    Math.sin(dLong/2)*Math.sin(dLong/2));
            float c = (float) (2*Math.atan2(Math.sqrt(a),Math.sqrt(1-a)));

            distance+=RADIUSEARTH*c;
        }
    }
    return (float)train.round(distance,2);
}
```



Mountain Bike App LOGOUT

Duration	0:1:31 HH:mm:ss
Max. Altitude	112.3 m
Min. Altitude	88.3 m
Distance	0.29 Km
Calories	14.8 kcal
Máx. Velocity	13.76 km/h
Average Speed	10.79 km/h

Figure 22 - Report Fragment

2. Force Sensor Gloves: this fragment is responsible for presenting the graphs of the four sensors in gloves athlete;

```
private float convertValue(int sensorValue) { //Function to convert % to ADC values
   float result=0;
   result = (sensorValue*654)/100; //convert the values in percentage to ADC
   return result;
}
private float convertADCValueToForceValue(float adcValue) { convert ADC value to Kg
   float gain = 1500/172;  // 1.5Kg /ADCvalue for the 1.5kg
   float forceValue= (float) (gain*adcValue*0.001);  // convert ADC to g and to kg
   return (float) train.round(forceValue,3);
}
```

3. Force Sensor Shoes: In this fragment is possible to see the graphs that translate the force of the six mounted sensors;
4. GPS Tracking: Gives an idea of the route that the biker performed on is workout;
5. Velocity: present the velocity graph in km/h;
6. Altitude: shows an altitude graph in meters.
7. Angle: Demonstrate the angle that the GPS sensor is making with Satellite.
8. Angles Bike: In the chest of the biker the IMU calculate the pitch, roll and yaw of the biker and in the fragment is presented the graph of that values.
9. Angles Biker: Like in the Biker, the bicycle has another IMU. So to see the movements of the bicycle the graph is presented here.

10. IMU Bicycle and Biker difference: To see the difference between the biker and the bicycle movements this fragment present a graph of that.
11. Export: All the data is saved in the database, but if the user wants to export the training data here is possible, only need to give a file name and press a button.

In the AndroidManifest.xml is important to write the following code, because the application needs to have access to the Internet and to the SD card.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

The following code represents all the libraries used in the build.gradle (Module: app) of the android studio:

```
repositories {
   maven { url "https://jitpack.io" }
}

dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   testCompile 'junit:junit:4.12'

  compile 'com.android.support:appcompat-v7:23.0.0'
  compile 'com.android.support:design:22.2.1'
  compile 'com.mcxiaoke.volley:library:1.0.19'
  compile 'com.android.support:support-v13:23.0.+'
  compile 'com.google.android.support:wearable:+'
  compile 'com.google.android.gms:play-services-wearable:+'
  compile 'com.github.PhilJay:MPAndroidChart:v2.2.5'
  compile 'com.jjoe64:graphview:4.1.0'
  compile 'com.google.android.gms:play-services:9.0.1'
}
```

5.2. Main Features

This application uses a two secure authentication, if any user wants to use this application he needs to be register in the database. Only registered users can access to this application, in order to protect user data. This mechanism is based in username and password to identify the user. At the coordinator only bikers that have a card or tag registered can perform a training session.

The communication with the database in the application is made through HTTP requests using the Volley library. Volley is an HTTP library that makes possible the networking for Android easier and faster.

For the visual presentation of the data collected form the database, is used two libraries GraphView and MPAndroidChart with these two libraries is possible to create the line graphs, bar charts and the points Charts that is possible to observe in the fragments. With these libraries is possible to display to the user's multiple series of data, scroll with a finger touch move gesture and read the values from the graphs. Allowing the user analyze every aspect of the graph created.