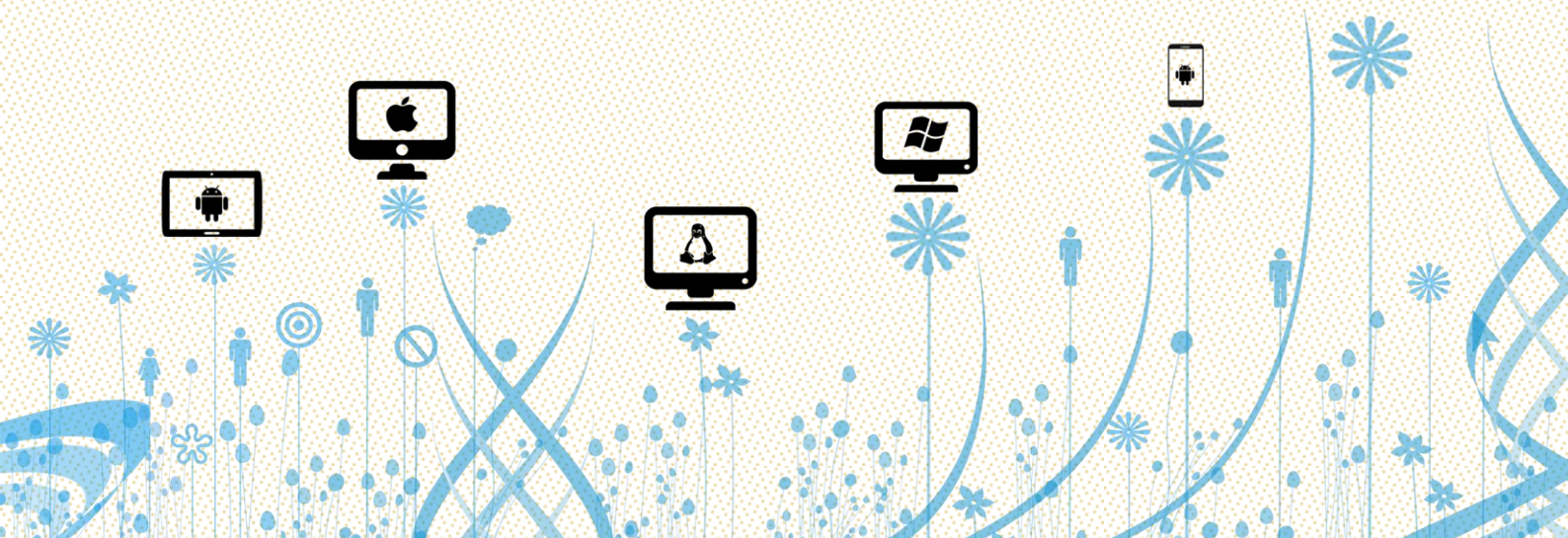
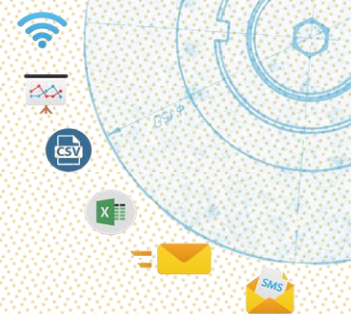
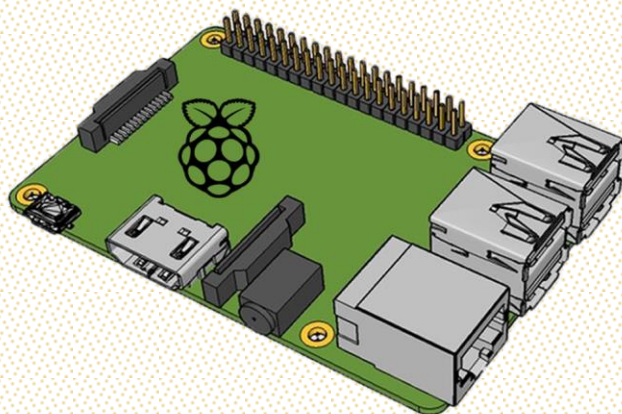


RASPBERRY PI
CONNECTING DEVICES



Luis Miguel Rocha Jacinto

Raspberry Pi controlling a process using I/O ports

Interconnection of multiple devices

Master Thesis submitted for partial approval of the requirements of Master
of Electrical and Computer Engineering Degree in Automation speciality

September 2015

• U • C •



UNIVERSIDADE DE COIMBRA

Raspberry Pi controlling a process using I/O ports

Interconnection of multiple devices

Luis Miguel Rocha Jacinto

A Dissertation
for Graduate Study in MSc Program
Master of Science in Electrical and Computer Engineering

Work Developed Under Supervision of
Prof. António Paulo Mendes Breda Dias Coimbra (DEEC-UC) and
Prof. Iztok Fajfar (FEE-UL)

Jury
Prof. Rui Alexandre de Matos Araújo (President)
Prof. António Paulo Mendes Breda Dias Coimbra (Vowel)
Prof. Lino José Forte Marques (Vowel)



Electrical and Computer Department
University of Coimbra
Portugal
September 2015

Acknowledge

"Success is not final, failure is not fatal: it is the courage to continue that counts."

Winston Churchill

I'm deeply thankful for all the support and strength my parents and brother gave me over the last years. Without them this work wouldn't be possible.

I would like to thank my coordinator Paulo Coimbra for guiding and supporting me over. Your discussion, ideas and feedback in this process have been absolutely invaluable to me.

Also I want to thank Iztok Fajfar for accepting me for this master thesis and for introducing this topic. I am very appreciated.

I also place on record, my sense of gratitude to my friends and all other persons, who, directly or indirectly, have helped me in this important step in my life.

Abstract

”It has become appallingly obvious that our technology has exceeded our humanity.”

Albert Einstein

The Raspberry Pi, a single-board computer, has never been fully explored as a server to connect multiple platforms and Operating Systems. In this work, it was developed a software compatible with multiple platforms and Operating Systems to acquire, process and deliver data to other devices either by wire or wireless. This software includes interfaces that allow the communication between Raspberry Pi and different Operating Systems, namely Windows, Linux and Mac for computers and Android for phones and tablets. Furthermore, to assure a better performance of the system, it was used multithreading and time-triggered routines. Moreover, some features and functionalities were added to allow an easy interaction of the user with the data. This includes receiving, analyzing, monitoring and storing the data, among others. The Android interface also includes the option to use a “SMS alert” service. The “SMS alert” service notifies the user once a preset value was reached allowing the user to react to critical situations.

A sensor of temperature and humidity was used for simulation purposes. Nevertheless, with the current software any other type of sensor can also be used, regardless of the numbers of output variables.

Keywords: Raspberry Pi, interfaces, multithreading, time-triggered routines, monitoring, functionalities, “SMS alert” and software compatible

Resumo

” Tornou-se bastante óbvio que a nossa tecnologia excedeu a nossa humanidade.”

Albert Einstein

O Raspberry Pi, um computador *single-board*, nunca foi completamente explorado como um servidor para conectar múltiplas plataformas e sistemas operativos. Neste trabalho, foi desenvolvido um *software* compatível com múltiplas plataformas e sistemas operativos para adquirir, processar e entregar dados a outros dispositivos por cabo ou sem cabo. Este software inclui interfaces que permitem a comunicação entre o Raspberry Pi e diferentes sistemas operativos, nomeadamente Windows, Linux e Mac para computadores e Android para telemóveis e tablets. Além disso, para assegurar uma melhor performance do sistema, foi usado *multithreading* e rotinas *time-triggered*. Mais ainda, alguns serviços e funcionalidades foram adicionados para permitirem uma fácil interação do utilizador com os dados. Isto inclui receber, analisar, monitorizar e armazenar os dados, entre outros. A interface Android inclui também uma opção para usar o serviço “Alerta SMS”. O serviço “Alerta SMS” notifica o utilizador que um valor predefinido foi atingido permitindo ao utilizador reagir a situações críticas.

Um sensor de temperatura e de humidade foi usado para fins de simulação. Porém, com o software atual qualquer outro tipo de sensor pode ser usado, independentemente do número de variáveis de saída.

Palavras-chave: Raspberry Pi, interfaces, rotinas *time-triggered*, monitorizar, funcionalidades, “Alerta SMS” e *software* compatível

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Summary of The Work Developed	2
1.4 Structure of the Thesis	3
2 Technical Background	6
2.1 Introduction to Raspberry Pi	6
2.2 State of art	8
2.2.1 Existing competitors	8
2.2.2 Summary	12
3 Hardware, Software and Technologies Used	15
3.1 Software and Technologies	15
3.1.1 TCP Protocol	15
3.1.2 TCP Sockets	15
3.1.3 VNC Protocol	15
3.1.4 SQLite	16
3.1.5 Multithreading	16
3.1.6 Timer Events	17
3.1.7 Android Preferences	17
3.1.8 Third-Party Libraries	18
3.2 Hardware	18
3.2.1 UART	18
3.2.2 Wireless USB for Raspberry Pi	19
3.2.3 DHT22 sensor	19
3.2.4 LCD	19

4	Development and Implementation	21
4.1	Preparation	21
4.1.1	Setting up VNC to support remote control of Raspberry Pi	21
4.1.2	Setting up Port Forwarding for External Connections	21
4.1.3	Setting up the UART	22
4.1.4	Setting up Plotting systems	22
4.1.5	LCD configuration for Raspberry Pi	23
4.1.6	Android Permissions	23
4.1.7	SQLite Tables Structure	23
4.2	Raspberry Pi Interface	24
4.2.1	Sensor Thread	25
4.2.2	TCP Clients Thread	25
4.2.3	UART Thread	25
4.2.4	Structure of the interface	26
4.2.5	Electrical connections between Raspberry Pi, sensor, LCD and UART	26
4.3	QT interface	27
4.3.1	Serial Port	27
4.3.2	TCP Client	28
4.3.3	View Data	29
4.3.4	Classes developed to handle important tasks	30
4.4	Android Interface	31
4.4.1	The Monitor Activity - TCP Client	31
4.4.2	Stored Data Activity	33
4.4.3	View Data Activity	34
4.4.4	Settings Activity	35
4.4.5	Classes developed to handle important tasks	36
4.5	Difficulties and Solutions	37
4.6	Summary	38
5	Results and Discussion	40
5.1	Android “SMS alert” Service Demonstration	40
5.2	Raspberry Pi Process Control Demonstration	41
5.3	CPU usage	45

5.3.1	Raspberry Pi	45
5.3.2	Android	47
5.3.3	QT	48
5.4	Discussion of the main results	49
6	Conclusions and Future Work	51
6.1	Conclusions	51
6.2	Future Work	52
A	Configurations and setting up features	54
A.1	Finding the IP of Raspberry Pi	54
A.2	VNC installation and configuration	54
A.3	VNC in the boot	54
A.4	Port Forwarding	56
A.5	Deactivate Raspberry Pi's UART	58
A.6	QT in Linux based Operating Systems	58
A.7	WiringPi instalation for C/C++ Development	59
A.8	RPi.GPIO installation for Python Development	59
B	Tutorials and examples	60
B.1	QT Examples	60
B.1.1	Serial Ports Names in QT	60
B.2	Python Examples	61
B.2.1	Email	61
B.2.2	Serial Port	62
B.2.3	LED Turn On/Off	62
B.2.4	LCD	63
B.3	C/C++ Examples	67
B.3.1	LED with button - Pull Up/Down pin:	67
B.3.2	PWM - LED brightness	68
B.3.3	LCD - 4-bit Mode	69
B.3.4	LED Blinking Effect	70
B.3.5	Makefile to compile the examples	71

C Useful information	72
C.1 TCP Connection	72
C.2 Multiple Readers/One Writer Problem	72
C.3 LCD Modes for Character Display	73
C.3.1 8 bits Mode	73
C.3.2 4 bits Mode	75
C.3.3 Comparison of the two modes	77
 Acronyms and symbols	 78

List of Figures

1	Raspberry Pi 2 [?]	7
2	Banana Pi [?]	8
3	BeagleBone Black [?]	9
4	Intel Galileo Gen 2 [?]	10
5	ODROID-C1 [?]	10
6	pcDuino3 [?]	11
7	UDOO Quad [?]	11
8	Database Structure	24
9	Raspberry Pi interface - Structure	26
10	Raspberry Pi interface - Connections	26
11	QT interface - Serial Port	28
12	QT interface - TCP Client	29
13	QT interface - View Data	30
14	Android interface - Monitor activity	32
15	Android interface - Stored Data activity	34
16	Android interface - View Data Activity	35
17	Android interface - Settings activity	35
18	Android interface - “SMS alert”	40
19	Raspberry Pi interface - UART and TCP Clients connected	41
20	Raspberry Pi interface - UART and TCP Clients disconnected	42
21	Raspberry Pi interface - Server shutdown	42
22	QT interface - Linux TCP Client	43
23	Android interface - TCP Client	43
24	QT interface - Windows Serial Port	44
25	Raspberry Pi interface - Running to test CPU usage	45
26	Raspberry Pi interface - CPU usage	46
27	Android interface - CPU usage	47
28	QT interface - Serial Port CPU usage	48
29	QT interface - TCP Client CPU usage	48
30	QT interface - View Data CPU usage	49

31	Setting up the port and the protocol	56
32	Association of the device to the port forwarding	57
33	Checking if port is open	57
34	Three-way handshake	72
35	LCD 8-bit Mode	74
36	LCD 4-bit Mode	76

List of Tables

1	Comparison between the single-board computers - 1/2	12
2	Comparison between the single-board computers - 2/2	13
3	UART Configuration Bits	22
4	Android Permissions [?]	23

Chapter 1

Introduction

*”I am just a child who has never grown up. I still keep asking these ‘how’ and ‘why’ questions. Occasionally, I find an answer.”
Stephen Hawking*

Over the years the use of microcontrollers has increased world wide. The appearance of the single-board microcontrollers, a microcontroller printed in a single circuit board providing all the circuitry necessary to control a task, introduced a new way to develop. This new way allowed the developers to focus on the development of applications rather than focusing in the development of the controller hardware. Also, they introduced a new way to teach electronic in education. Later, appeared the single-board computer, also printed in a single circuit board, introducing a high computational performance with some features similar to computers. This allowed its use as an embedded system (embedded computer controller), for educational purposes and also for the development of systems. With the appearance of the internet it became evident the need to connect different platforms and devices to single-board microcontrollers and single-board computers.

This work will focus on Raspberry Pi, a recent credit-card sized single-board computer that is revolutionizing the electronic community with its features [?]. The focus will be to show why it is so good and appropriate to use Raspberry Pi in order to create a system able to connect different platforms and devices. This work was mainly developed in Slovenia under the Erasmus program.

1.1 Motivation

The Raspberry Pi was introduced in 2006, and it has been a crucial tool for new students from all ages to learn how to program and to be introduced in the electronic environment. It has a powerful set of features that made it worldwide known and used by a vast number of users.

The main motivation is to make a first approach to a field, that has never been fully explored by the majority of Raspberry Pi’s users. The use of Raspberry Pi as a way to connect multiple

platforms and Operating Systems. Through this it should be able to handle the communications, to control processes, and to exchange data between the different devices. With this work it is intended to show that there is potential winnings if this device is used in the society.

1.2 Objectives

The main objective was to develop three interfaces: a server interface, the Raspberry Pi interface, and two endpoint interfaces, an Android interface and a QT interface. The focus was to allow the connection between Raspberry Pi and other devices. The Raspberry Pi interface is intended to allow the connection of multiple devices to exchange data. This connections would be either by wire, QT interface, or wireless, both Android and QT interfaces. Furthermore, the endpoint interfaces would allow the user to monitor, analyze and to manage the data using charts (graphical implementation). Moreover, these two interfaces have a GUI implemented that makes the interfaces user-friendly. Among other features and functionalities, the Android interface would include a service named “SMS alert” to notify a predefined phone number via SMS once a preset value is reached. Furthermore, all the interfaces would be developed using threading technology to guarantee a better usage of the resources, e.g. memory, and allowing a proper use of the CPUs available in the Raspberry Pi, computers, smartphones and tablets. Using threading mechanisms the complexity of the interfaces is increased.

Three different programming languages would be used in this work: C++, Java and Python. The first to be used in the QT interface, the second in Android interface and the last in Raspberry interface.

1.3 Summary of The Work Developed

At the beginning it was written guidelines on how to do the initial configuration such as finding out the Raspberry Pi’s IP address in order to configure a remote access, using VNC protocol, and on how to set up some needed features in this work, see appendix A. Also, for a brief introduction to Raspberry Pi, to know how to use it and how it works it was developed a few number of tutorials, see appendix B, on how to use the common basic electronic devices, how to interact with some new technologies and how to access the GPIO of this single-board computer.

After this short introduction, it started to be developed the Raspberry Pi, Android and QT interfaces. In the Raspberry Pi interface it was developed a TCP server, using TCP sockets,

and an UART communication. The first is intended to allow wireless communication, while the second allow wire communication. It was used a thread exclusively for the UART communication and for the TCP server, a dedicated thread for each TCP client connected. In terms of the sensor, one specific thread is responsible to read data from the sensor. This thread is able to send the data to the UART and TCP clients threads and when they receive it they send it to the particular user over wire and wireless respectively. In the Android interface it was developed a TCP client that is able to connect to the TCP server. It was developed the “SMS alert” and an “email sharing” services. The first allows to notify a predefined phone number that critical values have been detected and the second allows to send stored data to a predefined email address. In the QT interface it was developed a TCP client and a UART communication. It was developed a driver for the UART communication. The Android and QT interfaces were also developed to allow the user to monitor, analyze and process data in a graphical way. A database feature was also added in the two interfaces allowing the user to access stored data and do some actions with it. As an example exporting the data as a CSV or XLSX file format.

It is important to mention that this dissertation allowed me to consolidate the knowledge acquired throughout the course by exploiting the concepts of programming such as threading, TCP sockets, conditional objects, time-triggered routines, GUI development and also introduced me to new programming languages such as Java and Python that are a great acquisition to my portfolio.

In addition, this project was also very enriching for me regarding the programming domain, which is a very interesting business area that I would like to get involved into in my future job.

1.4 Structure of the Thesis

This dissertation contains five chapters apart from this one. They are as following:

- *Technical Background* - An introduction to Raspberry Pi is made, a state of art introducing the most know competitors, their advantages and disadvantages.
- *Hardware, Software and Technologies Used* - It is described and introduced the most relevant hardware, software and technologies used to support this dissertation.
- *Development and Implementation*- This chapter contains all the work developed, properly introduced, contextualized and explained with detail.

- *Results and Discussion* - In this chapter are showed and discussed the results of the work developed in the latter chapter.
- *Conclusions and Future Work* - It is presented the conclusions and the future work that can be done.

Chapter 2

Technical Background

*” You have enemies? Good.
That means you’ve stood up for
something, sometime in your
life.”*

Winston Churchill

There is a vast variety of single-board computers. The most appropriate for each application should be chosen after analyzing and comparing which will have the best performance and bring the best results for a desired application. With this in mind, this chapter starts with an introduction to Raspberry Pi and includes a comparison between some single-board computers.

2.1 Introduction to Raspberry Pi

Raspberry Pi is a worldwide known brand that gained its popularity after the success of its released single-board computers. In this work, it will be detailed the last Raspberry Pi release, the version 2 model B. This single-board computer has functionalities similar to a desktop computer and a powerful set of hardware components that allow its use to develop applications with high demands. Some of the main features are a System-on-Chip (SoC) Broadcom BCM2836, an ARM Cortex-A7 with 4 cores with 900MHz CPU, 1GB RAM (LPDDR2), HDMI port to connect a monitor, 4 USB ports to connect some devices like a keyboard and mouse, MicroSD card slot, CSI camera interface, combined audio jack and composite video, DSI display interface to connect touch screen display, 40 General Purpose I/O (GPIO) pins header and Ethernet port [?, ?]. The last one allows to connect Raspberry Pi to the internet and do operations such as browsing, connecting to ftp servers, creating tcp servers or clients to exchange data, updating the system via terminal and more. Regarding the USB ports, it is possible to connect a wireless extender to support wireless connections to the internet eliminating the need to connect the Raspberry Pi via wire to a router. Raspberry Pi runs on a Linux based Operating System and this is the biggest advantage of this device since it allows the use of all the Linux compatible programs in it, e.g. git (a distributed version control system, commonly used to manage versions of developed applications), wolfram mathematica for school work, apache and wireshark. Also, there is a

vast number of Hardware on Top (HAT) that can be added directly to the Raspberry Pi [?]. This hardware allows to plug to Raspberry Pi other devices, for instance a GPS Module, RGB Matrix, Piano keyboard, among others [?]. This single-board computer has some features that a single-board microcontroller has, for instance the GPIO pins, see figure 1. The GPIO are a set of input and output digital pins and they have an important role to the Raspberry Pi. Some of the electrical technologies available are I²C, SPI, PWM and UART support.

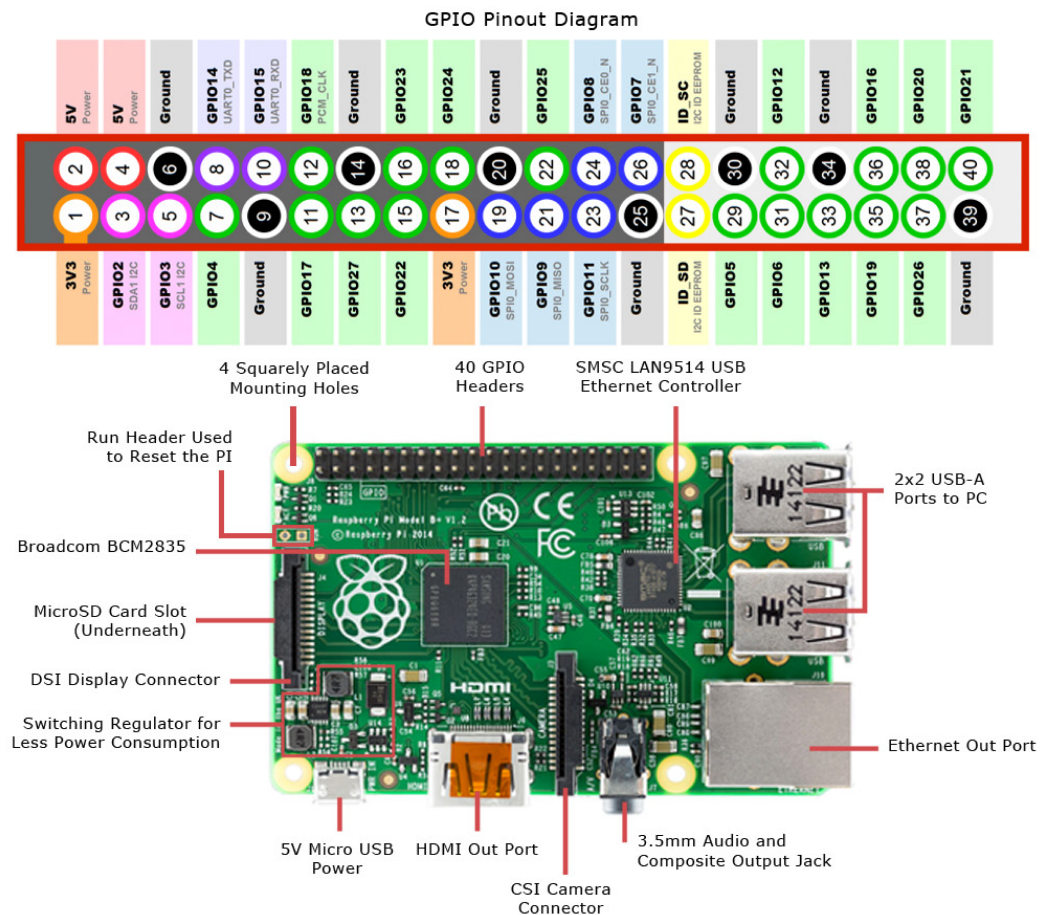


Figure 1: Raspberry Pi 2 [?]

In every device there is always some drawback and Raspberry Pi is no exception regarding to this topic. An example is that it does not support analog input since there is no ADC device. Fortunately, Raspberry Pi can be extended with a set of ADC devices to support analog input and allow the use of analog sensors. Another example is that there is only one pin that allows hardware PWM in this device. There are two solutions to this problem, either use software PWM or expand the Raspberry Pi with PWM circuits that use either SPI or I²C technology. The first solution is to use a software library¹. The drawback is that this library has low resolution (the minimum pulse width is 100 μ s [?]). This solution is not good for applications that require good

¹A C/C++ library is needed: WiringPi; See appendix A.7 for more information on how to install.

precision such as motors with encoders in robots or other devices where precision is absolutely mandatory. For these cases it is better the second solution. As a note, for outputting analog signals, PWM can be used regulating the Duty Cycle and applying a RC filter² to the output channel. In terms of Operating Systems it can run Linux based OS, Android, RISC OS and Windows 10 and the programming languages available are C/C++, Python, Node.js, Shell-script, Scratch, Java and other languages present in Linux Operating System.

2.2 State of art

This section will present and contextualize some single-board computers (SBCs) related to Raspberry Pi.

2.2.1 Existing competitors

Banana Pi

Banana Pi is a single-board computer with an Allwinner A20 SoC, an ARM Cortex-A7 2 cores with 1GHz CPU , 1GB (DDR3) RAM, it has an ethernet port, HDMI, CSI, SD card slot, support for audio I/O (stereo jack and microphone), 26 GPIO pins header, and 2 USB. It supports Hardware PWM (just one pin), SPI, I²C and also UART. This SBC is very similar to Raspberry Pi 1 model A (practically the same GPIO pins header) but has some functionalities that Raspberry Pi 1 does not have, for instance ethernet port (10/100/1000 Mbps). Banana Pi can run Linux based Operating System and Android. As programming languages there are available C/C++, Python and other languages available in Linux based Operating System [?].

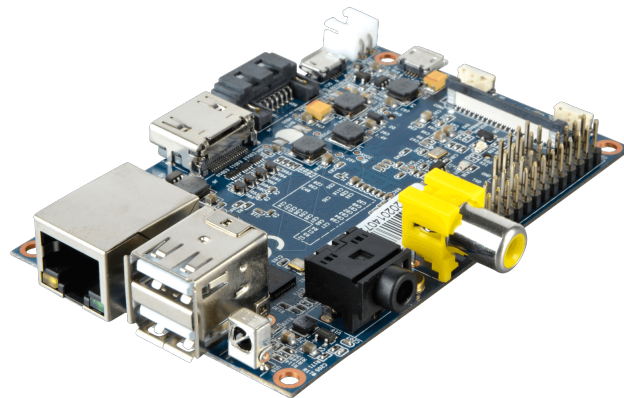


Figure 2: Banana Pi [?]

BeagleBone Black

²If it does not suffice a higher order active low-pass can be used or a higher PWM frequency [?].

Some of BeagleBone Black main features are a Texas Instrument Sitara AM3359 SoC, an ARM Cortex-A8 with 1 core and 1GHz CPU, 512MB (DDR3L) RAM, 4GB of memory (for the Operating System), microSD card slot, its support for microHDMI, 1 USB port , ethernet port (10/100 Mbps), a 46-pin dual-row expansion header including 7 analog input pins (ADC) and a large number of digital input and output pins[?]. It includes support for hardware PWM, I²C, SPI and UART. The main programming languages available are Python, C/C++, Java, Ruby and Node.js, among other languages available in a Linux based Operating System. As for Operating Systems there this SBC can run Android, Fedora, Angstrom Linux and also some other Linux based [?].

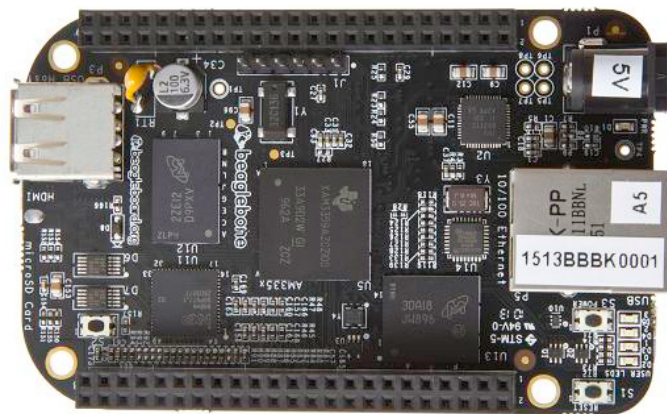


Figure 3: BeagleBone Black [?]

Intel Galileo Gen 2

This single-board computer has an Intel QuarkSoc X1000 SoC, a x86 Quark with 1 core and 400MHz CPU, 256MB (DDR3) RAM, 8MB NOR Flash and 8KB EEPROM onboard storage, a MicroSD card slot, 1 USB, ethernet port (10/100 Mbps) and an Arduino-compatible header (Arduino 1.0 header). This header has 20 digital I/O (12 fully native speed), 6 analog input (ADC) and it has support for 6 PWM (12-bit resolution), SPI master, I²C and UART. The Operating systems able to run in this single-board computer are Linux based Operating System and Windows. The programming languages available are Node.js, html5, C/C++ and Python [?, ?].

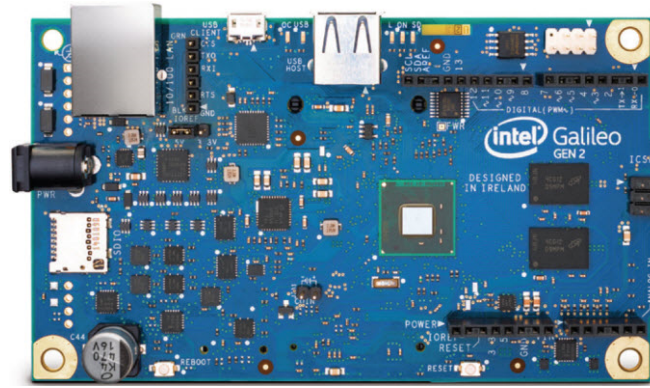


Figure 4: Intel Galileo Gen 2 [?]

ODROID-C1

ODROID-C1 is a single-board computer with an Amlogic S805 SoC, an ARM Cortex-A5 with 4 cores and 1.5GHz CPU, 1GB (DDR3) RAM, can be added an eMMC module as an onboard memory, microHDMI, a RTC, IR Receiver, MicroSD card slot, 40 GPIO pins header, supports analog input (ADC), 4 USB and ethernet port (10/100/1000 Mbps). Also it supports SPI, I²C and UART. This SBC can have the following Operating Systems: Linux based Operating Systems and Android. As for programming languages it can be used the languages available in the two Operating Systems mentioned [?].

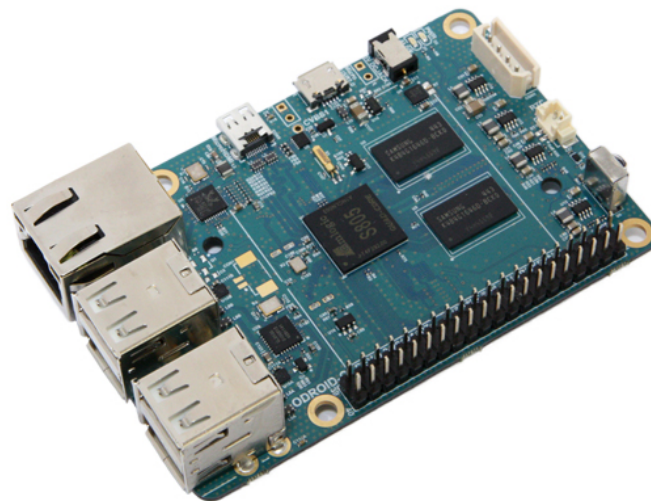


Figure 5: ODROID-C1 [?]

pcDuino3

This single-board computer has a Allwinner A20 SoC, an ARM Cortex-A7 with 2 cores and 1GHz CPU, 1GB (DRAM) RAM, it has a 4GB (Flash) onboard memory, HDMI support, MicroSD card slot, IR Receiver, 1 USB, ethernet port (10/100 Mbps) and an Arduino-compatible header (Arduino 1.0 header). This header has 14 digital I/O, 6 analog input (ADC) and it has

support for 2 PWM, SPI master, I²C and UART. The Operating Systems available for pcDuino3 are Linux based Operating System and Android. As for programming languages it can be used the languages available in the two Operating Systems mentioned [?, ?].

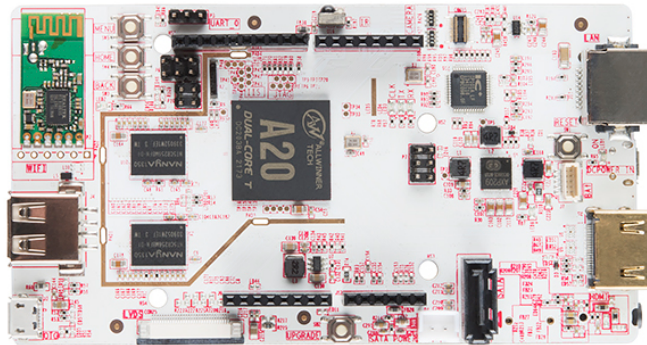


Figure 6: pcDuino3 [?, ?]

UDOO Quad

This single-board computer has a Freescale i.MX6 Quad SoC, an ARM Cortex-A9 with 4 cores and 1GHz CPU, 1GB (DDR3) RAM, HDMI support, MicroSD card slot, CSI, analog audio and microphone, Wifi module, sata support, FlexCan (Flexible Area Network) support, 2 USB, ethernet port (10/100/1000 Mbps), 76 GPIO pins with an Arduino-compatible header (Arduino 1.0 header) and support for analog input (ADC). It has support for PWM, I²C, SPI and UART. The Operating Systems available for this single-board computer are Linux based Operating System and Android. As for programming languages it can be used the languages available in the two Operating Systems mentioned [?, ?].

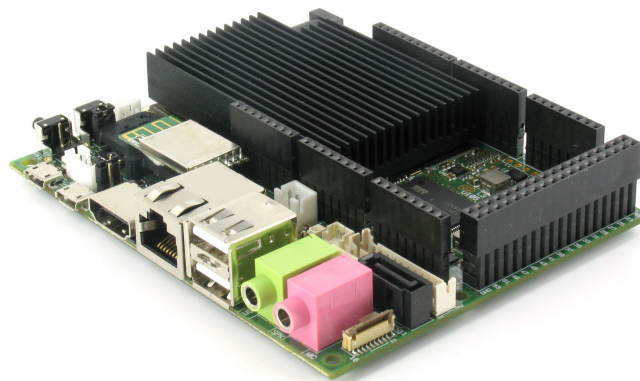


Figure 7: UDOO Quad [?, ?]

2.2.2 Summary

In the table bellow is summarized the Operating Systems, System-on-Chip (SoC), CPU and RAM that each single-board computer above mentioned have:

Table 1: Comparison between the single-board computers - 1/2

Single-Board Computer	Operating System	SoC	CPU	RAM
Banana Pi	Linux based OS Android	Allwinner A20	ARM Cortex-A7 2 cores - 1GHz	1GB (DDR3)
BeagleBone Black	Linux based OS Android	TI Sitara AM3359	ARM Cortex-A8 1 core - 1GHz	512MB (DDR3)
Intel Galileo Gen 2	Linux based OS Windows	Intel QuarkSoc X1000	x86 Quark 1 core - 400MHz	256MB (DDR3)
ODROID-C1	Linux based OS Android	Amlogic S805	ARM Cortex-A5 4 cores - 1.5 GHz	1GB (DDR3)
pcDuino3	Linux based OS Android	Allwinner A20	ARM Cortex-A7 2 cores - 1GHz	1GB (DRAM)
Raspberry Pi 2 model B	Linux Based OS Windows, Android	Broadcom BCM2836	ARM Cortex-A7 4 cores - 900MHz	1GB (LPDDR2)
UDOO Quad	Linux based OS Android	Freescale i.MX6 Quad	ARM Cortex-A9 4 cores - 1 GHz	1GB (DDR3)

The SoC, System-On-Chip, integrates all the components into a single silicon chip. It has the CPU, GPU, memory, USB controllers, power management circuits, among others. SoC main advantage is its very high level of integration and its little size. Since it has a shorter wiring it has a low energy consumption. The type of SoC in a single-board computer is a very important parameter. The CPU ranges are from 1 to 4 cores and the speed from 400MHz to 1.5GHz. In terms of RAM the ranges goes from 256MB to 1GB. In terms of Operating System all the SBCs have Linux based OS and some also support Android and Windows OS.

In terms of prices and the most important features, the table bellow summarizes all the information:

³Information available in: [?, ?, ?, ?, ?, ?, ?]

Table 2: Comparison between the single-board computers - 2/2

Single-Board Computer	Price (€) ³	Other interfaces
Banana Pi	38	HDMI,UART , 2 USB, SD card, CSI, Ethernet**
BeagleBone Black	48	MicroHDMI, MicroSD card, UART 4GB (eMMC), 1 USB, Ethernet*
Intel Galileo Gen 2	66	Arduino 1.0 headers, MicroSD card, UART,1 USB, JTAG 8MB NOR Flash and 8KB EEPROM, Ethernet*
ODROID-C1	31	MicroHDMI, 4 USB, RTC, (IR) Receiver support for eMMC module, MicroSD card,Ethernet**
pcDuino3	45	Arduino 1.0 headers, 4GB Flash, HMDI, UART 1 USB, IR Receiver, MicroSD card, Ethernet*
Raspberry Pi 2 model B	31	HMDI, CSI, DSI display, MicroSD card, UART 4 USB, HAT compatible,Ethernet*
UDOO Quad	87	Arduino 1.0 headers, MicroSD card, CSI HDMI, 2 USB, Ethernet**

* (10/100 Mbps)

** (10/100/100 Mbps)

In terms of price the range goes from 31 to 87 €. Comparing the features that these SBCs have to their prices it is clear to conclude that it is a good ratio. The most expensive SBC is UDOO Quad however, comparing its features to the price is quite acceptable. In terms of developing an application/system it should be analyzed first the needs and after choose from the available SBCs one in particular that fits the purpose needed, taking in considerations the parameters mentioned in the tables 1 and 2. Concluding, Raspberry Pi is a relatively cheap SBC that has some features and functionailites similar other SBCs.

Chapter 3

Hardware, Software and Technologies Used

” Whoever is careless with the truth in small matters cannot be trusted with important matters.”

Albert Einstein

In this chapter, it will be mentioned the hardware, the software and technologies that had contributed and supported significantly the work developed.

3.1 Software and Technologies

It will be mentioned all the software and technologies used in the development of the Raspberry Pi, Android and QT interfaces.

3.1.1 TCP Protocol

TCP is a transport layer protocol that provides reliable, ordered and error-checked delivery of a stream of data. It is mainly used by applications that require guaranteed delivery of packets [?].

3.1.2 TCP Sockets

Processes running on different machines communicate with each other by sending messages into sockets, using a TCP connection. Data can be exchanged as if there was a direct virtual pipe between client and server [?]. The TCP protocol will guarantee that all the data sent will be received and in the same order that was sent.

For more information about how to establish a TCP connection, using a TCP socket, please refer to C.1.

3.1.3 VNC Protocol

Virtual Network Computing is a graphical desktop sharing protocol to remotely control another computer. It can also transmit the keyboard and mouse events from one computer, the

client, to the one that is sharing the desktop, the server. There can be more than one client connected to the server. This technology allows for example remote maintenance, accessing files in a system and having control of a particular computer.

3.1.4 SQLite

SQLite is a database management system that complies with the SQL syntax. In contrast to other database systems, SQLite is not a client-server database. Instead, it is embedded into the application. Also, there is bindings for almost every programming language. Some of them are Java, C/C++, PHP, Python and Ruby.

3.1.5 Multithreading

Multithreading is a concept where multiple threads can exist within the context of a single process and run independently. This allows applications to be more responsive, faster executing and doing tasks and more importantly, makes use of the multicore and multi-CPU systems since it is possible to divide tasks into parallel tasks (threads) and let the OS decide if they run either concurrently, on a single core or, in parallel, on multiple cores.

Async task

In Android development, it is not allowed to access directly elements from the Android GUI thread directly by working threads. To be able to access them there are different solutions, one of them is the Async tasks. This solution allows to send arguments to a working thread and there are four main functions with an important role. In particular, the first one allow small configurations of variables and setting values, the second one does the work and is able to publish if new work is done, the third one receives the published work and allow to interact with the Android GUI thread and access its elements and the last one is called at the end of all the work to do the cleaning.

For more information about this class refer to [?].

QThread

QThread is a QT class that allow the use of working threads in the development of an application. There is some ways to use this class. The most used are to move a worker object to a thread using the *moveToThread()* method or to subclass the QThread and override the *run()* method. For both methods, to connect the work done with the QT GUI Thread it is necessary

to make use of signals and slots. In the working thread it should be used an *emit* and in the GUI thread the respective signal should be connected to a particular slot to handle the event. In this way it is possible to combine working threads that do some work and the QT GUI thread not blocking its graphical view.

For more information about this class refer to [?].

Conditional Objects

To deal with one of the writer/reader problems, the multiple readers and one writer, and to grant synchronization between threads, in Python language, there is a conditional variable that uses a lock mechanism to synchronize access to a shared state. This mechanism solves the known problems associated with shared variables. This conditional variable assures that threads interested in a particular change of state, the readers, will wait until the thread that change the state changes it, the writer. When the writer changes the state it notifies all the readers, awakening and allowing them to access the shared variable. Also, this conditional object can be used to solve other writer/reader sub-problems a part from the one explained above.

For more information about conditional objects and the multiple readers and one writer problem refer to [?] and appendix C.2 respectively.

3.1.6 Timer Events

In QT to deal with work that has to be done periodically, timers are a valuable asset. They allow a task to be executed every single millisecond. When there is a timer timeout, a timer event is raised and with the timer id it is possible to redirect the work to the task needed. Another way to redirect the work is to connect the timeout signal to a slot and have a specific function for each task.

3.1.7 Android Preferences

In order to provide an user preferences settings menu in an android application that is able to remember the values after being changed it is necessary to use the *preference* API from Android repository. This settings are available to be read in the application and with them it is possible to do some actions depending on the user configuration. Also it is important to highlight that even when the user closes the application, the settings will have the values that the user configured before.

For more information about this API refer to [?].

3.1.8 Third-Party Libraries

QcustomPlot

QT does not include support to plot graphs in an application. To add this functionality it is necessary to add a third-party library, *QCustomPlot*.

AchartEngine

Android does not have any built-in libraries to allow the use of graphs in an application. For being able to plot graphs it is necessary to include in the project a third-party library such as *AchartEngine*. Then add a JAR file containing the library to the project and inform the gradle system to compile it in order to be able to use its methods.

JExcelApi

In order to be able to handle excel file format in an Android application it is necessary to add a third-party library, *JExcelApi*. To be able to use this library it is necessary to do the same steps explained in the section 3.1.8 in the *AchartEngine* separator.

QtXlsXWriter

To add excel file format support to QT it was needed to add a third-party library, *QtXlsXWriter* to the project. This library allows to create a XLSX file, access the cells to add data, create charts and many more.

3.2 Hardware

3.2.1 UART

A universal asynchronous receiver/transmitter is a device that translate data between parallel and serial forms. Parallel interfaces transfer multiples bits at the same time over a bus and serial interfaces stream the data one single bit at time. Essentially, the UART acts as an intermediary between parallel and serial interfaces taking bytes of data and sending them bit by bit sequentially. Since the data is transferred without support from an external clock signal it is categorized as an asynchronous communication.

3.2.2 Wireless USB for Raspberry Pi

The wireless USB allow the Raspberry Pi to have internet access without having to connect an ethernet port to a router and settle up all the working environment taking in consideration the router location.

3.2.3 DHT22 sensor

It is a relative cheap sensor for measuring temperature and humidity. Inside there is a chip responsible to do analog to digital conversion and to output the data over digital signals. It uses a non-standard one wire digital signaling protocol.

For more information, please refer to the datasheet [?].

3.2.4 LCD

A Liquid Crystal Display is an electronic device that allow to display digits, words, characters, images and even videos in it.

Chapter 4

Development and Implementation

"I never did anything by accident, nor did any of my inventions come by accident; they came by work."

Thomas A. Edison

In this chapter, it will be explained all the work developed, how it was accomplished, which difficulties stood against the progress and how they were overcome.

4.1 Preparation

The purpose of this section is to describe all the steps taken to set-up all the tools and functionalities needed to provide a stable and consistent working base.

4.1.1 Setting up VNC to support remote control of Raspberry Pi

Due to the absence of a HDMI monitor it was necessary to use the VNC protocol to provide a graphical remote access to Raspberry Pi. First it is necessary to find out Raspberry Pi's IP address, see appendix A.1 to know how to, and use any SSH application to connect to it via terminal. After having access to Raspberry Pi it was necessary to install *Tight VNC Server*, configure and start it. See appendix A.2 for detailed information on how to do this step by step. And at last to establish a connection using a computer it was only needed to insert Raspberry Pi's IP address in any VNC application. To avoid to start the VNC server manually all the time the Raspberry Pi is turned on it was added to the boot order a script to start it automatically. The steps to do this are available in the appendix A.3.

4.1.2 Setting up Port Forwarding for External Connections

In order to allow external connections to Raspberry Pi it was necessary to port forward the port number *5000* and associate it to Raspberry Pi's IP address in the router interface. This was an important step since it allows TCP connections to be made outside a local network and

grant access to the server in any internet access point. All the steps needed are detailed in the appendix A.4.

4.1.3 Setting up the UART

First it was necessary to deactivate the main use of the UART from Raspberry Pi's Operating System in order to be able to have exclusive access to it. This step was needed since Raspberry Pi uses the UART to serve as a command line access, login purposes, and to provide diagnostic messages during the boot time. To deactivate it was necessary to do the steps explained in the appendix A.5.

To establish the UART communication it is was necessary to define the communications settings.

The table bellow shows the configuration settings needed:

Table 3: UART Configuration Bits

Baudrate	Data Bits	Parity	Stop Bit	Flow Control
115200	8	No Parity	One Stop	No Flow Control

The information about how to connect the UART to the Raspberry Pi is shown in the figure 10 in the section 4.2.5.

4.1.4 Setting up Plotting systems

AchartEngine configuration for Android

In order to use this service it was necessary to create a class named *Point* to manage points (a date and a double) and another class named *LineGraph* to use the *Point* class to add points to a graph. Also, it was developed in the *LineGraph* class methods to clear the graph, to repaint the view, to configure the layout and to add different series into the same plot. In the layout XML file it is mandatory to add a chart id inside a Linear Layout and in the java class it is necessary to attach the view to it. Also, it was replaced the default zoom buttons to make the layout prettier and user-friendly. It is important to mention that it was developed these two classes to help managing the plot and its features in a real time mode.

QCustomPlot Configuration for QT

In order to configure and use *QCustomPlot* in QT it was necessary to insert in the GUI main file a *QWidget* and then promote it to a *QCustomPlot*. It was customized the graphs layout and a zoom in and zoom out functionality was added being the last one triggered by the mouse wheel.

4.1.5 LCD configuration for Raspberry Pi

In order to use a LCD character display it was necessary to first choose the mode to use it. There is the 4bit and 8bit mode and in this work it was used the first. To be able to use it it was necessary to create a library, in Python language, that controls the LCD via commands using its registers. In addition this commands differ accordingly to the mode selected. The library created was based on an existent program to handle an LCD [?].

For more information about the two modes and the registers please refer to appendix C.3.

4.1.6 Android Permissions

To use some features that an android device has it is mandatory that in the *AndroidManifest* XML file the required permissions are included. This way the resources needed will be available. Also, for security purposes when the user tries to install an application in an android device it will be prompted being warned about the potential usage of this particular application.

In the table bellow it is shown the permissions used in this work:

Table 4: Android Permissions [?]

Permission	Description
android.permission.SEND_SMS	Allow the application to send SMS
android.permission.INTERNET	Internet access and sockets access
android.permission.WRITE_EXTERNAL_STORAGE	Write access to the external storage
android.permission.READ_EXTERNAL_STORAGE	Read access to the external storage
android.permission.ACCESS_NETWORK_STATE	Access to the state of the internet

Also, as an extra information, for adding support to export the application from the phone memory to a MicroSD card it is necessary to add in the XML file referred before the following line *android:installLocation="preferExternal"* in the *manifest* tag.

4.1.7 SQLite Tables Structure

In order to be able to manage a database it was necessary to define the tables structure and their relationship. This database consists in two different tables *listData* and *monitorData*. This

tables are connected to each other by a foreign key (FK). This key is the *idListData* field that is present in the *monitorData* table and that is connected to the *_id* field in the *listData* table. The *listData* table stores the begin/end times and dates for a particular set of monitored data and also an unique identifier, while the *monitorData* table stores the temperature, humidity, the timestamps and a *idListData* for all the monitored done. For each entry of the last table a particular *idListData* is associated to a particular set of monitored data in the *listData* table.

In the figure bellow it is shown the database structure with the two tables and their relationship:

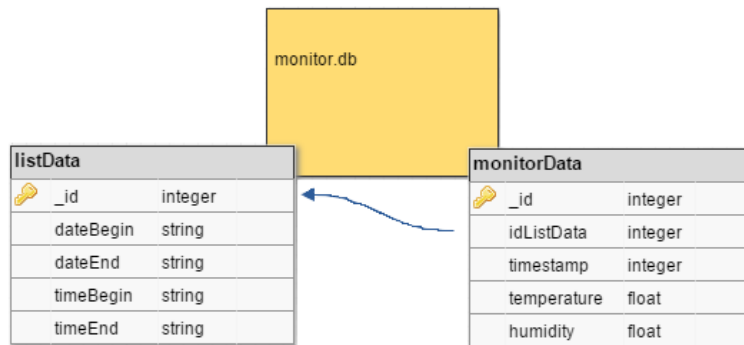


Figure 8: Database Structure

4.2 Raspberry Pi Interface

It was developed an interface for Raspberry Pi, using Python language, to allow TCP connections (a TCP server), to allow an UART connection and also an event manager to handle the readings of the sensor. The first uses wireless communication, while the second uses communication by wire. The TCP server can accept new TCP clients and start new threads for each client that successfully connects to the server. For the UART connection was developed a driver in order to communicate with devices able to communicate with this technology such as computers and microcontrollers. It was created a specific thread to handle the serial port communication, the sensor thread. For the event manager it was developed a specific thread to handle the sensor reading events. This thread is responsible to send the new data to the TCP clients and UART threads and also to print it in the LCD. There is a shared variable managed by a conditional object that is associated with a lock mechanism to handle synchronization between threads, refer to section 3.1.5 in the separator *Conditional Objects* for more information. This shared variable can be accessed by two different set of threads, the ones that need to read and the one that need to write to it. As readers there are the TCP clients and UART threads and as writer there is

the sensor thread. The last one can access the variable to write and when it does notifies all the readers, awakening and granting them access to read the shared variable and do their work. For safety reasons, when a CTRL+C is caught all the threads are safely terminated, the GPIO is cleaned up, the server shutdown and the program terminates.

See Appendix C.2 for more details about multiple readers and one writer problem.

4.2.1 Sensor Thread

This thread is responsible to read the data from the sensor, format the temperature and humidity to be with two decimal digits, to write and change the value of the shared variable and to notify all the TCP clients and UART threads that new data is available to be accessed. Also, in this thread the LCD is refreshed with the new data.

4.2.2 TCP Clients Thread

Each TCP client thread is able to access a shared variable that contains the new values from the sensor and send it to the according client over wireless. When a client disconnect, the connection is closed for that particular client and the thread terminates.

4.2.3 UART Thread

In this thread, first it is necessary to establish the connection with the right configuration, as it is shown in the table 3 in the section 3, and specify the port name. In this case the port name is `/dev/ttyAMA0`. It waits to receive in the buffer the message "Read\n", sent by the connected device. Then this thread enters in a loop accessing the shared variable to read and send it to the respective device over wire. To terminate the readings when it reads from the buffer the message "Stop\n" the communication is stopped. To start over the communication it is necessary to send again the message "Read\n".

4.2.4 Structure of the interface

In the figure below it is possible to see the structure of this interface, how it works and how it is connected with each thread:

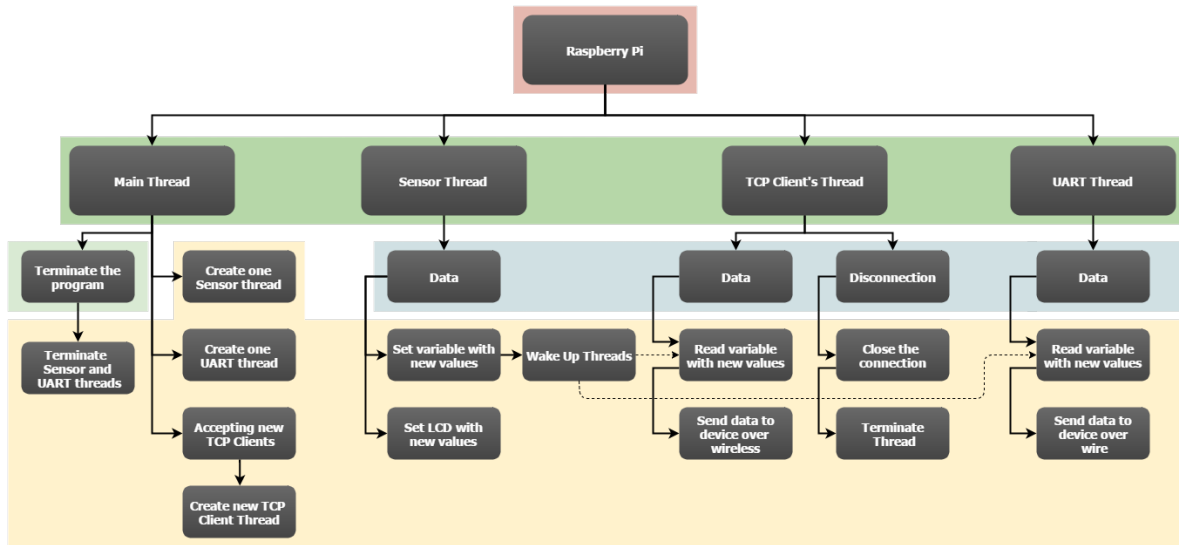


Figure 9: Raspberry Pi interface - Structure

4.2.5 Electrical connections between Raspberry Pi, sensor, LCD and UART

The figure below shows the connections between the Raspberry Pi, the LCD, the UART and the sensor. It is important to mention that the UART has a Micro-USB that needs to be connected to the device that will communicate over wire with the Raspberry Pi.

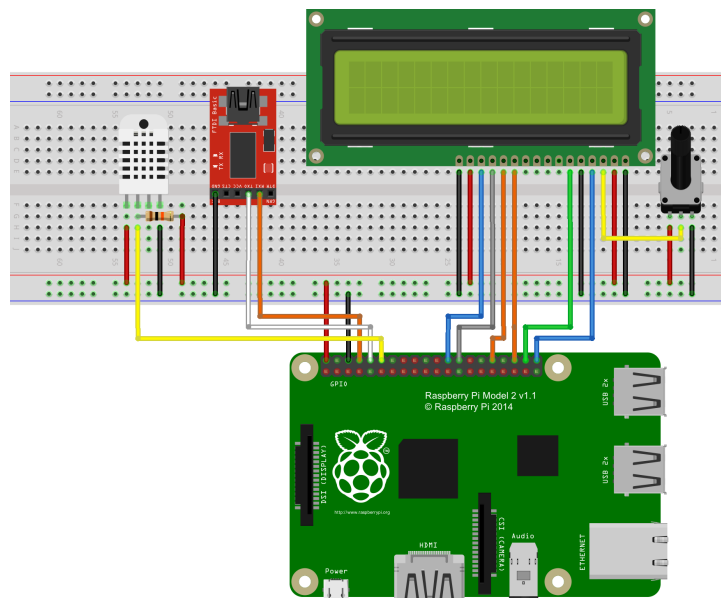


Figure 10: Raspberry Pi interface - Connections

4.3 QT interface

It was developed an interface that creates a TCP client, using a TCP socket, and a serial port communication, the UART connection, in order to connect to Raspberry Pi interface either by wire or wireless. This interface was build in QT to aim the compatibility with the most significant Operating Systems available in the market namely Windows, Linux and Mac. To use this interface in the Operating Systems mentioned it is necessary to install⁴ QT Creator in both systems, compile the code and the interface is ready to be used. It is important to mention that it is possible to use at the same time the two connections, the TCP client and the serial port communication. However there is no point in receiving the same data in different ways. For that reason it was disabled this service. This is done by disabling a tab when one connection is established. This tab is present in the GUI window and can be seen in the figure 11 in the section bellow. In this interface the UART connection does not make use of the database since it was chosen to develop only support for the TCP connection. At last it was developed a way to enable zoom in and zoom out of the plot in real time using the mouse wheel, exportation of stored data via XLSX or CSV file formats and also a way to save images of the graphs in PNG format file.

4.3.1 Serial Port

In this part, the first step to be done was to find out the serial port name of the connected device in the Operating System dynamically. To make the UART connection easier to the user, when the programs starts a combo box (multiple choice) is automatically filled with the available port names of the current connected devices to the computer. If for any case the user did not have the device connected before loading the interface it is possible to refresh and automatically the name will appear in the combo box. For more information about how to check the available ports in QT please refer to the appendix B.1.1. The second step was to develop a serial port communication being necessary to open the UART device and set up the configuration bits, see table 3 in the section 3. When the connect button is clicked the serial communication port is established and on success it is sent the message “Read\n” to the Raspberry Pi interface. When the Raspberry Pi interface receives this message the exchange of data starts. To prevent bad behavior in the QT interface the read button is enabled when the connection is established and

⁴For MAC Operating System it is necessary to install Xcode available on MAC installation CD [?]. For Linux refer to appendix A.6.

the stop button when the connection is active. For not colliding with the GUI thread in QT, when the read button is clicked a timer is instantiated to tick each second. Each timeout is handled in an event handler that reads the serial port, deals with the data formats and insert the data in the graph not affecting the GUI thread's performance. It is important to mention that it is necessary to know the id of the timer in order to handle the event. To terminate the readings, it is necessary to send the message "Stop\n" to Raspberry Pi interface. This is done when the user clicks in the stop button. In order to add the functionality to save the plotted graph as an image it was necessary to use the *savePNG* method available in the UI element of the graph and specify the file path to save it. As default it was chosen the computer desktop.

The figure 11 shows the GUI tab for the serial port communication.

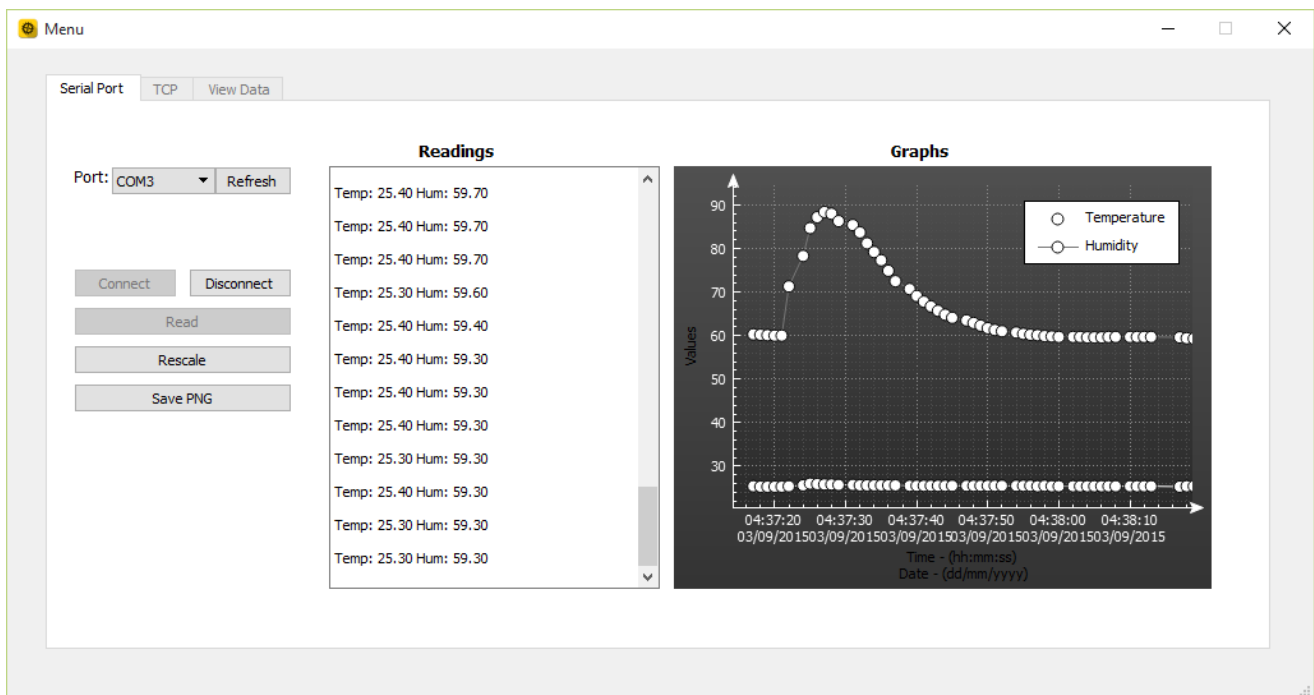


Figure 11: QT interface - Serial Port

4.3.2 TCP Client

The first step was to develop a method, using TCP sockets, to connect, by wireless, the QT interface with the Raspberry Pi interface, the TCP server. To do so it was necessary to provide the server's IP and port addresses. This two parameter are present in a text box, see figure 12, with default values that the user can change. To prevent bad behavior from the user, when the connection is established the disconnect button is enabled and the connect button is disabled. When the connection is established first time the graph is initialized and a timer is configured to tick each second until the user stops it. Each timeout event is handled by a task that reads

the data from the socket buffer, inserts the data in a graph and store the data in the database. To save the graph as an image the button save PNG has to be clicked.

The figure 12 shows the GUI tab for the TCP communication:

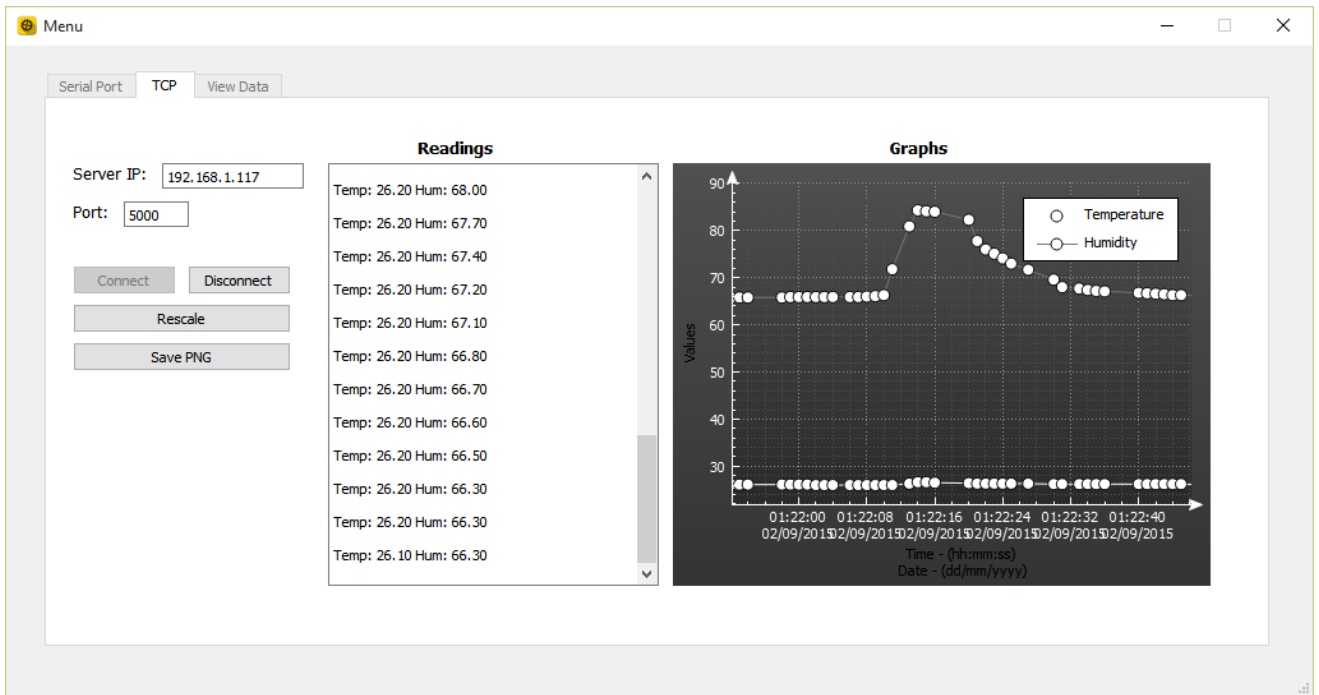


Figure 12: QT interface - TCP Client

4.3.3 View Data

In this part of the work, it was developed a way to access stored data. This allows the user to re-plot the graph and save images of it, export the data as a XLSX or CSV file format and delete either one or all the entries available in the database. For accessing a particular set of monitored data, it is necessary to double click in the respective entry of the table and the graph for that particular data is plotted, see figure 13. The data is fetched from the database by a working thread in the background of the interface. For this working thread it was necessary to develop a class called *MyThread* that is connected to the GUI thread via signals and slots. This is done in order to communicate the data already fetched from the database by the working thread to the GUI thread. When the GUI thread receives data it automatically adds it to the graph. For more information about this class please refer to the section 4.3.4 in the *MyThread* separator.

The figure 13 shows the GUI tab for the view data:



Figure 13: QT interface - View Data

It was also developed a *MyThreadExport* class that uses a thread to create the two files mentioned. They are stored in the computer desktop in use. This class is similar to *MyThread* class in terms of technology used and both uses *QThread* mechanism.

4.3.4 Classes developed to handle important tasks

MyThread class

In this class the *run* method from the *QThread* class was overridden in order to fetch data from the database using a working thread and sending it to the GUI thread. This working thread is connected to the GUI thread by a signal and a slot. A signal is emitted by the working thread when there is new work done to publish. When the GUI thread receives the work already done, via a slot, it is inserted the data in the graph. It is important to mention that this process was necessary to be able to communicate between this two threads. In the GUI thread, when a double click in a particular row of the table present in the View Data tab, see figure 13, is detected it is started a thread with a *QSqlQuery* as a parameter containing the result from a specific query. With this query it is fetched data from the database.

DataBaseManager class

In this class it was developed methods to create the database, the two tables needed for the interface and all the needed commands to interact with the tables, for instance to insert, delete, update and retrieve data. For methods that query the database to retrieve data it is returned a *QSqlQuery* variable that is after looped to extract the data. For methods that are exclusively for inserting, updating and deleting it is not returned any value. It is used binding arguments to make the SQL syntax easier to understand and to avoid complex strings with the required queries. Since this is a object-orient technology, it is only necessary to add a member of this class in the GUI header file to instantiate the database and use its methods.

4.4 Android Interface

This interface was developed to allow a wireless connection of all android devices to the Raspberry Pi interface. It is possible to receive data and store it in a database for later access, export data to XLSX and CSV file formats and send it via email, warn the user about a high value via SMS, the “SMS alert”, specify user preferences and settings, zoom in and zoom out the graph with multi-touch support, among other features.

4.4.1 The Monitor Activity - TCP Client

In the Monitor activity, it was developed the wireless connection between Android and Raspberry Pi interfaces trough the use of a TCP socket. Since it is required internet access for establishing the connection, when the user clicks in the connect button if there is no internet access a warning is thrown. Also if the socket gives the error: “cannot reach the host”, a warning is generated to warn the user that either the IP or port addresses provided in settings are wrong or the server is not active. When the connection is established successfully the connect button is changed to a disconnect button and the exchange of data starts.

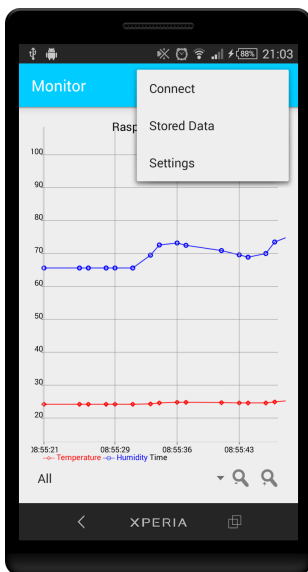
The following algorithm shows the logic behind this process:

Algorithm 1 Establishing Connection

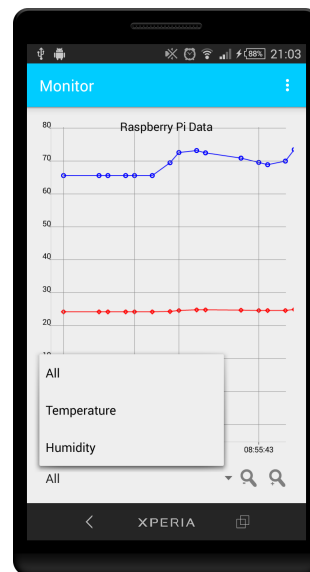
```
1: procedure CONNECT EVENT ▷ When the connect button is clicked
2:   if IP is valid then
3:     if there is internet access then
4:       create the socket
5:       if host is unreachable then
6:         warn either to start the TCP Server or that the IP or port addresses are wrong
7:       else
8:         establish connection
9:         switch the connect button with a disconnect button
10:        starts exchanging data
11:     else
12:       warn that there is no internet access in the android device
13:   else
14:     warn user that the IP address is invalid
```

It was developed also a class using SQLite technology to be able to store the data in a database for accessing the data in the future. See section 4.4.5 in the *Database* class separator for more information about the class structure and its features. Also, it was developed zoom and filter functionalities to allow zoom in and zoom out and to filter the data allowing to see better some areas of interest in the plot and also to see either all the plots, the temperature or the humidity.

In the figure bellow it is possible to see the layout of this activiry, the available actions and filters. The stored data and the settings buttons will be explained in detail in the bellow sections.



(a) Menu actions



(b) Filters in the graph

Figure 14: Android interface - Monitor activity

In this activity there is three important actions/services. They are as follow: “SMS alert”, store data in the database and add the new data to the graph. In terms of relationship between this three actions/services the following algorithms synthesises this process:

Algorithm 2 Relationship between SMS service, database and graph

```

1: procedure MONITORIZING PROCESS ▷ When the monitorizing is active
2:   if data is available then
3:     if data > threshold and “SMS alert” is allowed then
4:       if time elapsed since last sent SMS = time interval then
5:         send SMS with the high value detected
6:       insert data in the graph
7:       insert in the database

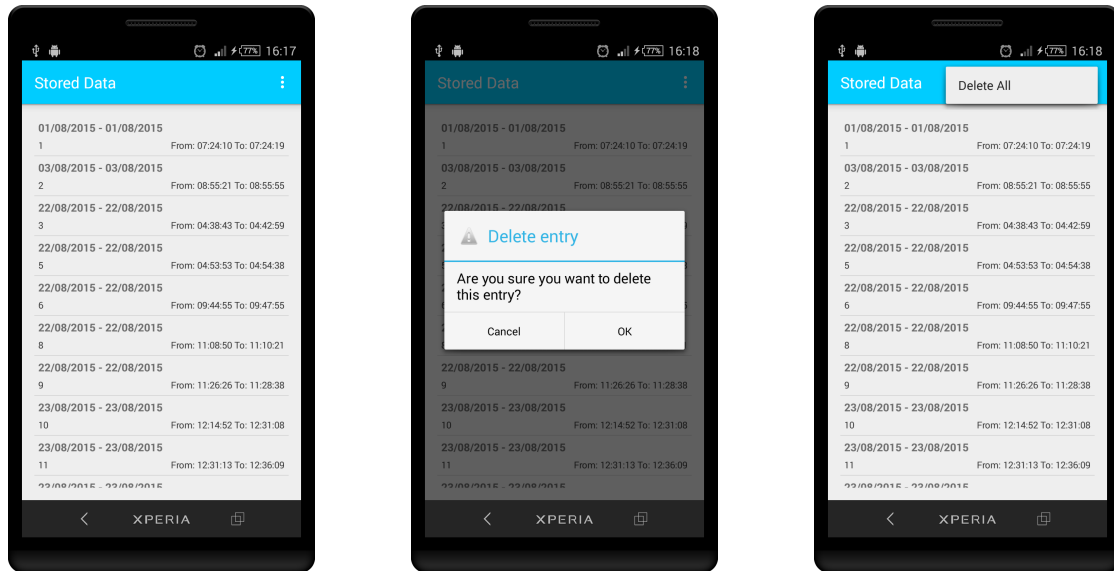
```

In the numbers 3 and 4 it is important to explain that the user is allowed to choose if the interface can use the “SMS alert” service, define a threshold to trigger it and the time interval to wait for the next SMS to be sent. The time interval is just a precaution to avoid sending a huge amount of unwanted SMS. This preferences that the user can define are explained in detail in the section 4.4.4.

4.4.2 Stored Data Activity

In this activity it is possible to see the stored data in form of a list. This list is created dynamically and when it gets wider, a vertical scroll view is inserted automatically. It is important to explain that this list is a custom list view developed in order to increase the functionality and to make the interface user-friendly. It was developed three main actions in this activity: long and simple press in a item and the delete all button. The first action, the long press, when it is triggered a pop up appears warning if the user wants to delete that entry, see figure 15b. If the user chooses the OK button the data is delete from the list view and from the database. The second, the simple press, goes to the View Data activity to display a graph with all the information about that entry. This activity is explained in the section 4.4.3. The third, the delete all button, deletes all the entries if the user clicks in the OK button in the pop up window. It has a confirmation pop up to avoid deleting all the data by mistake.

The figure bellow shows the custom list view, the long press in a item and the delete all button:



(a) Custom List View

(b) Long Press in the item

(c) Delete all entries

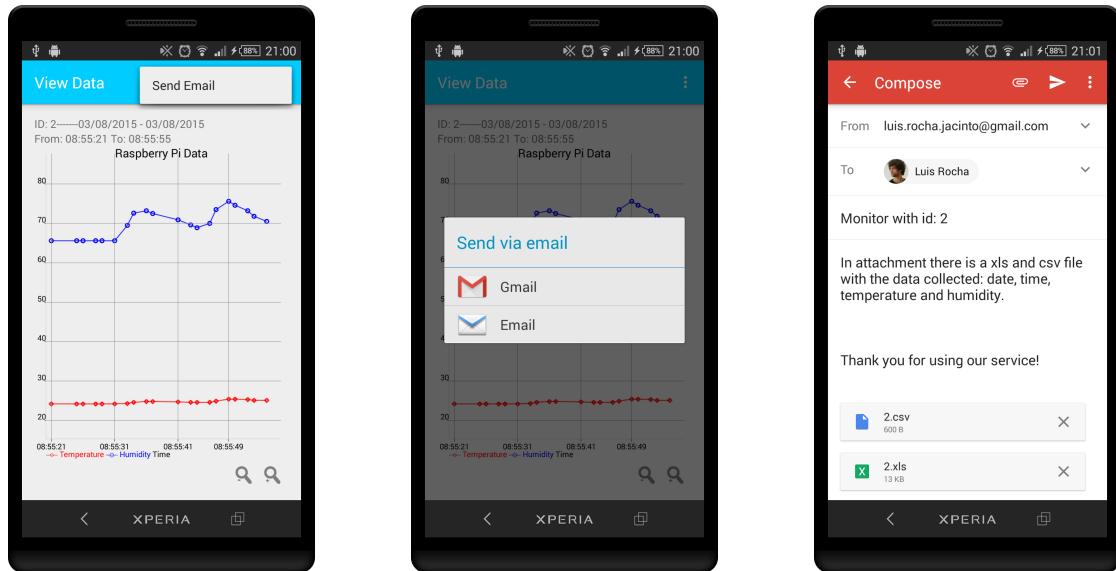
Figure 15: Android interface - Stored Data activity

4.4.3 View Data Activity

When a simple press in a item is detected in the Stored Data activity it is triggered the View Data activity that receives as parameter the id of the item selected. In this activity it is queried the database in order to plot the values in the graph for that particular element. In the graph it is possible to zoom in and zoom out. It was not developed in this activity the filter service. When the email button is clicked, see the figure 16a, a pop up window will appear to choose the email provider to be used, see figure 16b. When the user selects the provider an async task starts and query the database to retrieve the data for a particular id creating a XLSX and CSV file in their respective formats⁵. The name of the files generated is the id of the *listData* table since for each set of monitored data there is a different id generated by the database and therefore no conflict with the name of the files created. This two files are appended to the attachment of the email, a default subject is generated, the message is inserted and the recipient is the one configured in the settings of the interface. See figure 17 in the notifications header for understanding where it is configured the recipient. This process is fully automated and there is no need for the user to do anything other than clicking the send button. If by any chance the user wants to change some parameter it is possible to do that since it is allowed to the user to have full control of the email provider.

In the figure 16c it is shown the generated email:

⁵The CSV default format is in each row each value is separated by a comma. In the XLSX in each cell a value is stored.



(a) Send email action

(b) Choose the mail provider

(c) Generated Email

Figure 16: Android interface - View Data Activity

4.4.4 Settings Activity

The Settings activity has a main role in all the activities. The default actions and configurations to be used by the other activities are stored in this activity. This includes the default server's IP and port addresses, the phone number to send SMSs, the threshold to "SMS alert" and the time interval to avoid sending unwanted SMSs, a boolean flag to activate/deactivate the "SMS alert" and a default email address. It is important to highlight that this activity uses the preferences mechanism mentioned before in the section 3.1.7 and therefore it stores the preferences of the user. In the figure 17 it is shown the settings activity and its layout:

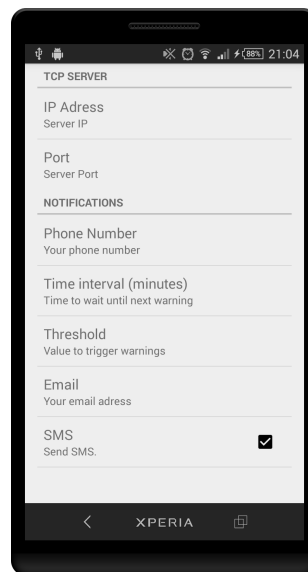


Figure 17: Android interface - Settings activity

4.4.5 Classes developed to handle important tasks

Graph classes

First it was developed a *Point* class that has two private variables, a date and a double, and their get and set methods. This class helps the *LineGraph* class allowing the encapsulation of data and handling the points of the graph in a easier way. In other words, the *LineGraph* class uses the *Point* class to create and manage points in the graph. The first class is also able to do all the configuration needed to start plotting. This two classes have an important role since they make it possible to plot smoothly in real time without having problems with the performance. For more information please refer to section 4.1.4 in the *AchartEngine configuration for Android* separator.

List View classes

It was developed the *CustomListAdapter* class to be able to modify the default android list view element and insert three text views showing the date, id and time for a particular set of monitored data in each row of the list. It was developed the *DataItem* class with three variables and their respectively set and get methods. The *CustomListAdapter* class uses the *DataItem* class to handle the data and associate the views in the layout file to the variables and then inflate the views. This layout file has the three text views mentioned before that will be present in each row of the list. In the *StoredData* activity it is used the *CustomListAdapter* class to populate the list view and to set up the long and simple press in a item. Explaining in a abstract way, the *CustomListAdapter* class uses the *DataItem* class to manage three items, the date, id and time and for each particular set of monitored data a dynamic row is added to the custom list in order to populate it. This classes are connected to each other and with them it is possible to obtain the final result present in the figure 15a.

Database class

Since almost every single activity in the Android interface need to access the database, it was decided to develop the *SQLiteManager* class that extends a *SQLiteOpenHelper* class from SQLite library. Some methods were overridden and other were created. The library developed is able to instantiate a database connection, to create the two tables if they don't exist already, to upgrade the version of the database and it has defined all the methods needed to query the database. Another purpose of this class is to assure that only one instance of the database is created and activities that need to write to the database will not collide with the activities that

need to read. This is done with *getWritableDatabase* and *getReadableDatabase* methods. Before executing the queries it is mandatory to call this methods. Doing this, the database and then the database main library will handle the requests and avoid conflicts. For retrieving the data from the queries a cursor is returned and a loop needs to be implemented in each activity that query the database.

4.5 Difficulties and Solutions

In this work, some difficulties got in the way of the progress. Some of the most important were:

- To understand how to configure the Raspberry Pi's VNC server to be able to use the graphical interface of this single-board computer.
- To start the communication using the UART since first it was discovered that the Raspberry Pi uses it to show boot-time messages and also for other services.
- It was difficult to get used to Python syntax since it is very different than C/C++.
- To use port forwarding it was necessary to understand the concepts behind this technology that are not straightforward.
- In QT it was hard to understand how to use the thread system.
- To understand how to develop Android applications.

The first difficulty that appeared was to access the Raspberry Pi's GUI interface. This was needed since it makes easier to program and develop. It was developed in this work tutorials on how to configure and use the Raspberry Pi, refer to the first three tutorials in the appendix A.

To use the UART it was necessary to do some configurations steps in order to disable the main function of Raspberry Pi's UART. For that it was necessary to research to learn how to do it.

The Python syntax was understood after a while and the abstraction level that comes with it made it possible to develop the Raspberry Pi interface.

To use and configure port forwarding it was necessary to research on how to open the port for the TCP protocol and to associate it with the Raspberry Pi's IP address. Also some research was made to understand better this concept.

To deal with threads in QT it is not easy to understand at first time. It is necessary to research the `QThread` class in QT documentation to know how to use this technology. In the section 3.1.5 in the *QThread* separator it is explained the two most used methods to use this technology. After understanding the mechanism it was straightforward the development.

To understand how to develop an Android application it was necessary to invest time in research. Some concepts are not easy to learn since there is a lot of classes, methods and technologies that need to be understood before starting the development phase. The development of the Android interface was slower than the QT interface.

4.6 Summary

Since this chapter is long, this section has the intention to summarize the work done. In a simple explanation, the Android and QT interfaces are intended to be endpoint interfaces that are able to connect to Raspberry Pi interface in order to exchange data. The Raspberry Pi interface is the “brain” of all the process and it works as a controller that handles the work, the requests and distribute the data either by wire or wireless to multiple devices. These devices can be either Android phones or tables or Windows, Linux or Mac computers.

The Android and QT interfaces are meant to simplify the information and to get an easier perspective of the data received. This is done by adding a graphical way to show the data in order to allow a better understanding about it. Also, there are mechanisms to make future analysis since it was implemented a database system in both interfaces.

Chapter 5

Results and Discussion

”However beautiful the strategy, you should occasionally look at the results.”

Winston Churchill

In this chapter, it is presented the results from experiments done in order to test the interfaces and verify their consistency. Also some demonstrations were done to show the most important services developed.

5.1 Android “SMS alert” Service Demonstration

To test the “SMS alert” service it was defined the threshold to be 75, see figure 18c. When the humidity value goes higher than this restriction, see figure 18a, it is sent a SMS to a default number, see figure 18b. After a time interval defined by the user, the interface is able to send again SMSs.

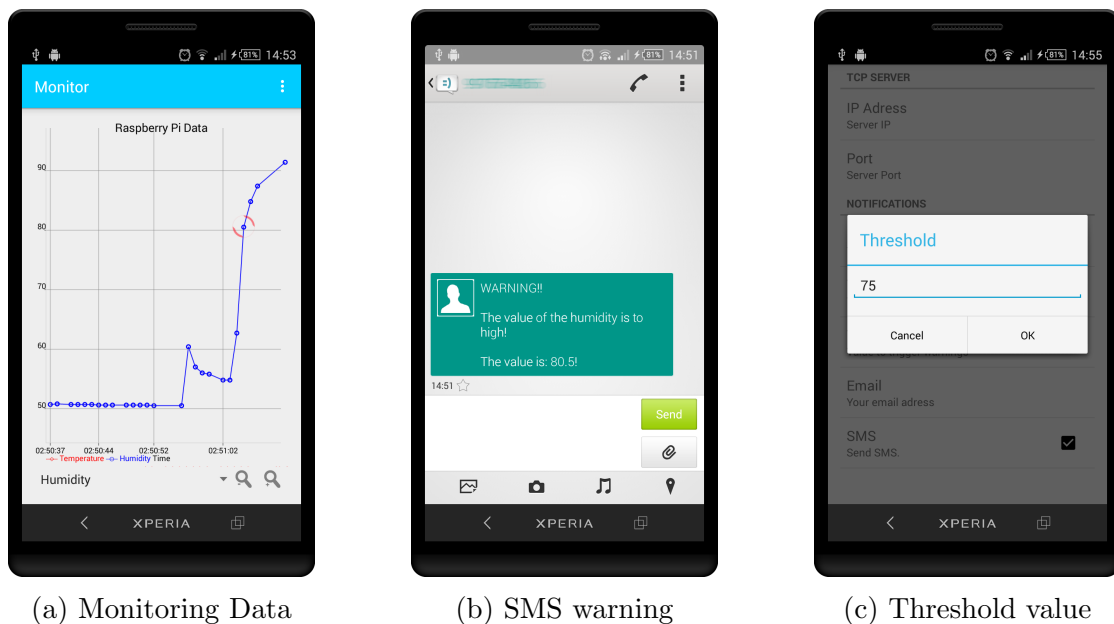
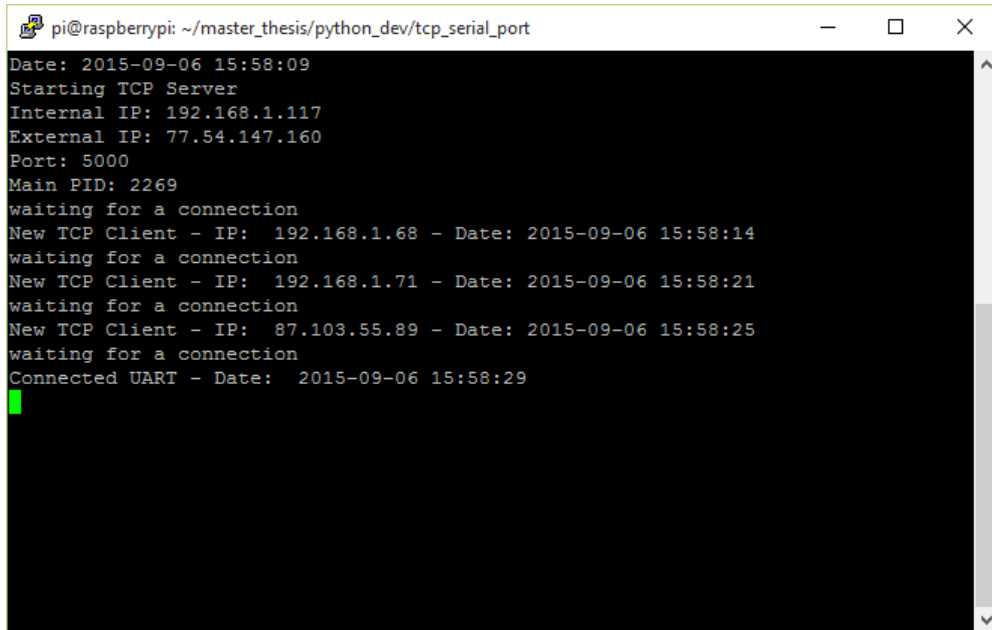


Figure 18: Android interface - “SMS alert”

5.2 Raspberry Pi Process Control Demonstration

To instantiate the Raspberry Pi's interface it was used *Putty* application. In this simulation it was used three TCP clients and the UART connection.

In the figure 19 it is shown the information about the server such as the server's external and internal IP and port addresses, the process identifier (PID), the date and the connections already accepted:

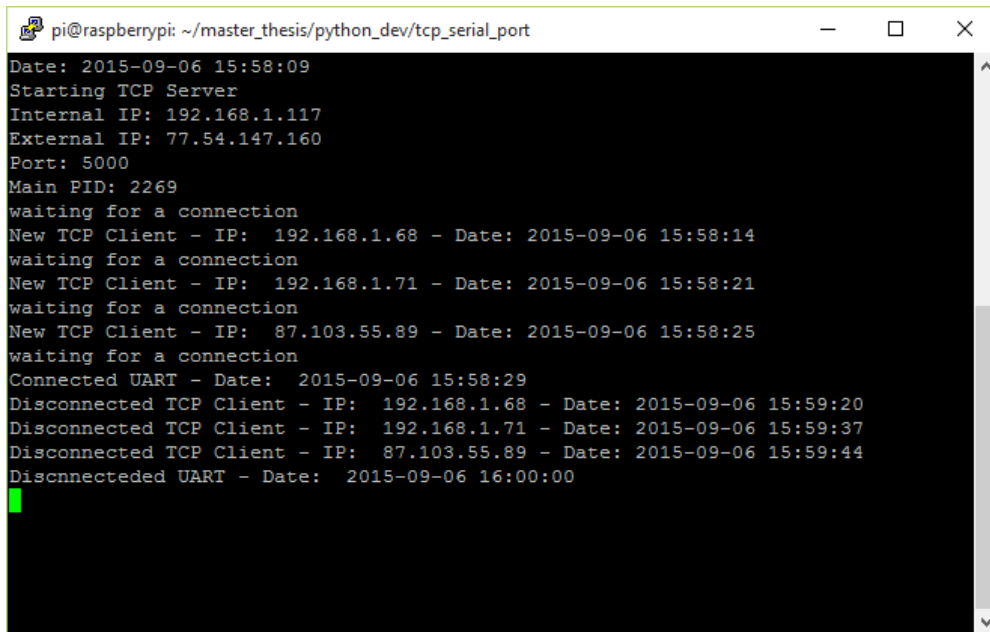


```
pi@raspberrypi: ~/master_thesis/python_dev/tcp_serial_port
Date: 2015-09-06 15:58:09
Starting TCP Server
Internal IP: 192.168.1.117
External IP: 77.54.147.160
Port: 5000
Main PID: 2269
waiting for a connection
New TCP Client - IP: 192.168.1.68 - Date: 2015-09-06 15:58:14
waiting for a connection
New TCP Client - IP: 192.168.1.71 - Date: 2015-09-06 15:58:21
waiting for a connection
New TCP Client - IP: 87.103.55.89 - Date: 2015-09-06 15:58:25
waiting for a connection
Connected UART - Date: 2015-09-06 15:58:29
```

Figure 19: Raspberry Pi interface - UART and TCP Clients connected

It is important to mention that the three TCP clients used were an android phone and two computers running Linux and Windows Operating Systems. The android phone was connected outside the local network using mobile data to access the internet, see figure 23. The Linux computer was running Freya OS (Ubuntu based OS), see figure 22. The last TCP connection is not shown in this demonstration. The UART connection was used by other computer running Windows OS, the one used to take the print-screens.

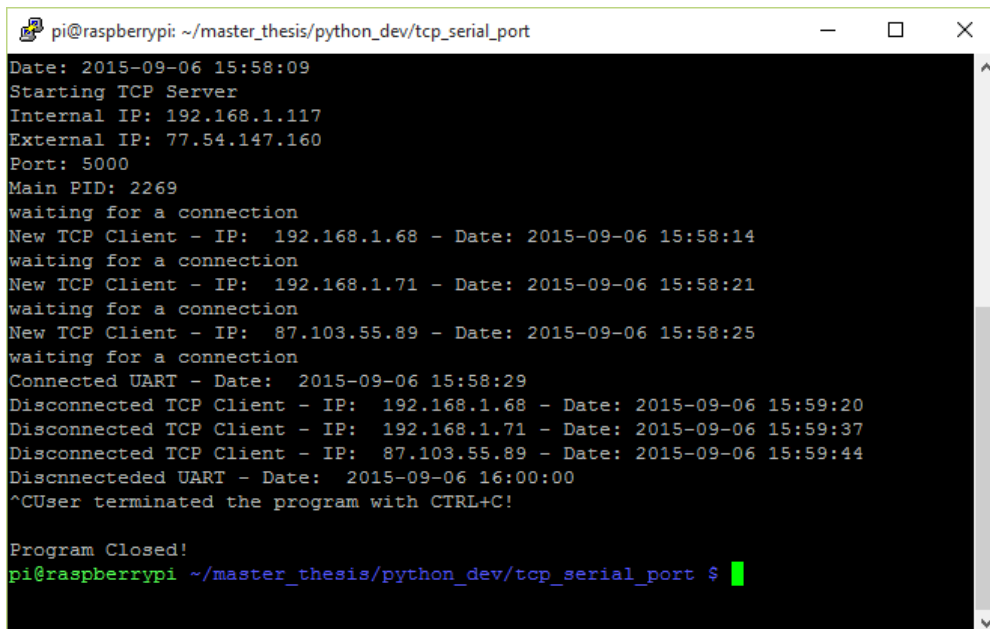
When the TCP Clients and the UART connection are disconnected, the server is still active waiting for new connections to happen. See figure bellow:



```
pi@raspberrypi: ~/master_thesis/python_dev/tcp_serial_port
Date: 2015-09-06 15:58:09
Starting TCP Server
Internal IP: 192.168.1.117
External IP: 77.54.147.160
Port: 5000
Main PID: 2269
waiting for a connection
New TCP Client - IP: 192.168.1.68 - Date: 2015-09-06 15:58:14
waiting for a connection
New TCP Client - IP: 192.168.1.71 - Date: 2015-09-06 15:58:21
waiting for a connection
New TCP Client - IP: 87.103.55.89 - Date: 2015-09-06 15:58:25
waiting for a connection
Connected UART - Date: 2015-09-06 15:58:29
Disconnected TCP Client - IP: 192.168.1.68 - Date: 2015-09-06 15:59:20
Disconnected TCP Client - IP: 192.168.1.71 - Date: 2015-09-06 15:59:37
Disconnected TCP Client - IP: 87.103.55.89 - Date: 2015-09-06 15:59:44
Discnecteded UART - Date: 2015-09-06 16:00:00
```

Figure 20: Raspberry Pi interface - UART and TCP Clients disconnected

And at last, when the *CTRL-C* is detected the server shutdown terminating all the working threads and the program. See figure bellow:



```
pi@raspberrypi: ~/master_thesis/python_dev/tcp_serial_port
Date: 2015-09-06 15:58:09
Starting TCP Server
Internal IP: 192.168.1.117
External IP: 77.54.147.160
Port: 5000
Main PID: 2269
waiting for a connection
New TCP Client - IP: 192.168.1.68 - Date: 2015-09-06 15:58:14
waiting for a connection
New TCP Client - IP: 192.168.1.71 - Date: 2015-09-06 15:58:21
waiting for a connection
New TCP Client - IP: 87.103.55.89 - Date: 2015-09-06 15:58:25
waiting for a connection
Connected UART - Date: 2015-09-06 15:58:29
Disconnected TCP Client - IP: 192.168.1.68 - Date: 2015-09-06 15:59:20
Disconnected TCP Client - IP: 192.168.1.71 - Date: 2015-09-06 15:59:37
Disconnected TCP Client - IP: 87.103.55.89 - Date: 2015-09-06 15:59:44
Discnecteded UART - Date: 2015-09-06 16:00:00
^CUser terminated the program with CTRL+C!

Program Closed!
pi@raspberrypi ~/master_thesis/python_dev/tcp_serial_port $
```

Figure 21: Raspberry Pi interface - Server shutdown

In the figure bellow, it is shown the Linux TCP Client used in the demonstration and its monitored data:

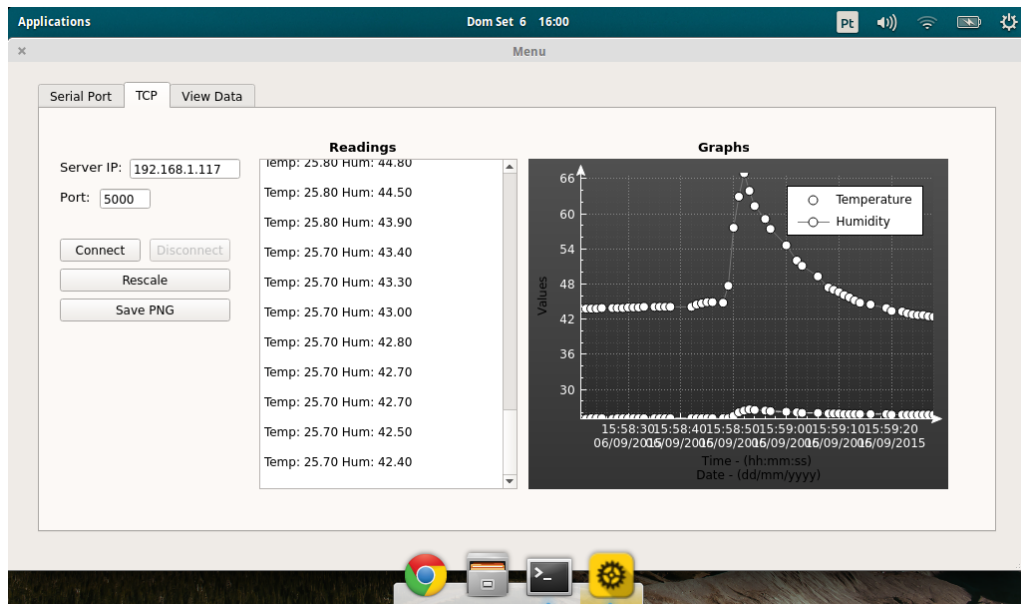


Figure 22: QT interface - Linux TCP Client

The android phone connected with mobile data to access the Raspberry Pi's server is shown in the figure bellow. It is important to mention, that for connecting to the server outside the local network it is strictly necessary to use the External IP address given in the Raspberry Pi terminal interface, see figure 19.

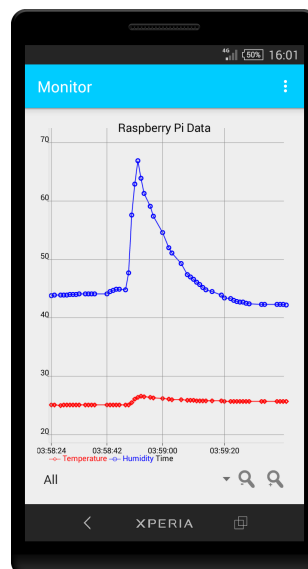


Figure 23: Android interface - TCP Client

Bellow it is shown the Windows computer used to connect via UART to Raspberry Pi interface:

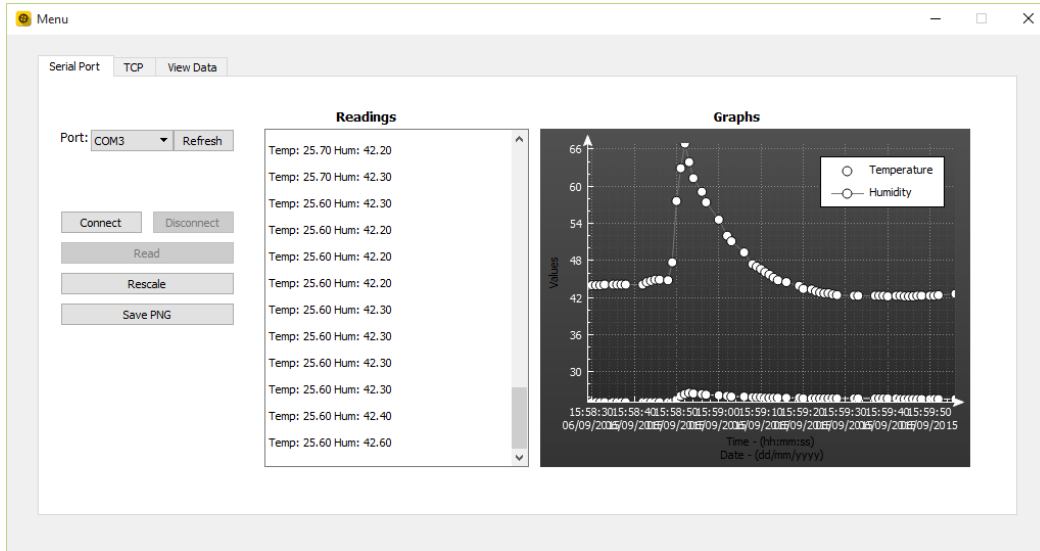


Figure 24: QT interface - Windows Serial Port

Testing the responsiveness of the interface:

The Raspberry Pi interface was also simulated in order to check if the server has a good response when new connections and disconnections of older connections are requested in a frequent way. For testing this it was used the same amount of devices used in the demonstration above and it was connected and disconnected devices frequently. The results were positive, the interface didn't slow down the communications and the data was delivered as it was expected. The interface was responsive.

Testing the interface when the internet is being used

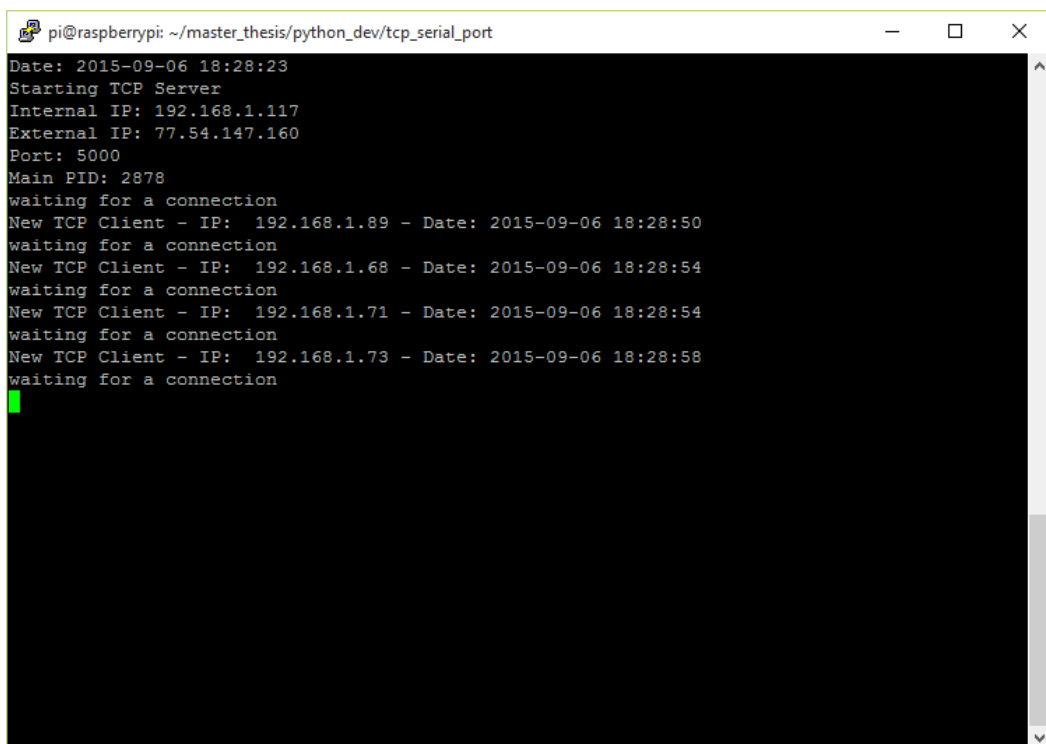
It was tested the interface when the internet network was being used by other users, downloading files and data. The wireless communication, the TCP server, worked as expected. However the transference of the packets was slower comparing the case without that usage. The wire communication, the UART connection, was not affected by this test since it does not depend on the internet usage and therefore it had a stable performance being faster than the wireless solution. At last, a good internet connection between the Raspberry Pi and the router is required to avoid the scenario in which Raspberry Pi is disconnected from the internet and due to that the server is not able to communicate with its TCP Clients.

5.3 CPU usage

The interfaces developed run smoothly without CPU overloads. This test was done to analyze the CPU usage to verify if the interfaces developed are consuming more resources than they should.

5.3.1 Raspberry Pi

In this test, it was used four TCP clients, connected in the local network, to the Raspberry Pi interface. The figure bellow shows the clients connected and most important the process identifier (PID) that will be used to test the CPU usage.

A terminal window titled 'pi@raspberrypi: ~/master_thesis/python_dev/tcp_serial_port'. The output shows the server starting at 18:28:23, listing internal IP (192.168.1.117) and external IP (77.54.147.160), and port (5000). The main PID is 2878. It then shows four new TCP clients connecting at 18:28:50, 18:28:54, 18:28:54, and 18:28:58, each with a different IP address (192.168.1.89, 192.168.1.68, 192.168.1.71, 192.168.1.73). The terminal is mostly black with white text, and a green cursor is visible at the bottom left.

```
pi@raspberrypi: ~/master_thesis/python_dev/tcp_serial_port
Date: 2015-09-06 18:28:23
Starting TCP Server
Internal IP: 192.168.1.117
External IP: 77.54.147.160
Port: 5000
Main PID: 2878
waiting for a connection
New TCP Client - IP: 192.168.1.89 - Date: 2015-09-06 18:28:50
waiting for a connection
New TCP Client - IP: 192.168.1.68 - Date: 2015-09-06 18:28:54
waiting for a connection
New TCP Client - IP: 192.168.1.71 - Date: 2015-09-06 18:28:54
waiting for a connection
New TCP Client - IP: 192.168.1.73 - Date: 2015-09-06 18:28:58
waiting for a connection
```

Figure 25: Raspberry Pi interface - Running to test CPU usage

Having the Raspberry Pi interface running and executing in another terminal the following command: `top -n 40 -d 2 -c -p 2878 | grep tcp_serial_port.py` where `-n 40` is the number of iterations, `-d 2` is the sample rate, two per second, `-c` is to show the absolute path of the running process and `-p 2878` is the process identifier (PID) it is possible to verify the CPU usage of this interface. The `top` (table of processes) command is a task manager available in many UNIX-like Operating Systems that provide the CPU usage of the processes. Using the `-p 2878` the `top` will only show results from the process with that PID. The `grep` is used to extract only the lines where the CPU usage is.

The values of the CPU usage, the yellow column, are low and there is no overload of the interface. The figure below shows the output from the `top` command:

```

pi@raspberrypi ~ $ clear && top -n 40 -d 2 -c -p 2878 | grep tcp_serial_port.py
2878 root    20    0 32808   12m 6088 S    0.0  1.3  0:01.43 python tcp_serial_port.py
2878 root    20    0 32808   12m 6088 S    3.5  1.3  0:01.50 python tcp_serial_port.py
2878 root    20    0 41000   12m 6088 S    3.5  1.3  0:01.57 python tcp_serial_port.py
2878 root    20    0 49192   12m 6088 S    2.0  1.3  0:01.61 python tcp_serial_port.py
2878 root    20    0 57384   12m 6088 S    2.0  1.3  0:01.65 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:01.72 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.5  1.3  0:01.77 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:01.85 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.0  1.3  0:01.91 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    0.0  1.3  0:01.91 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:01.99 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:02.07 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:02.14 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.5  1.3  0:02.19 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:02.26 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:02.34 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:02.41 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.0  1.3  0:02.45 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:02.53 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:02.61 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.0  1.3  0:02.67 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.0  1.3  0:02.73 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:02.80 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:02.88 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.0  1.3  0:02.92 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:03.00 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:03.07 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:03.15 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.0  1.3  0:03.19 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    1.0  1.3  0:03.21 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:03.29 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.0  1.3  0:03.33 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    1.0  1.3  0:03.35 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:03.43 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.0  1.3  0:03.49 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.5  1.3  0:03.54 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:03.61 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    4.0  1.3  0:03.69 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    2.0  1.3  0:03.73 python tcp_serial_port.py
2878 root    20    0 65576   12m 6088 S    3.5  1.3  0:03.80 python tcp_serial_port.py
pi@raspberrypi ~ $

```

Figure 26: Raspberry Pi interface - CPU usage

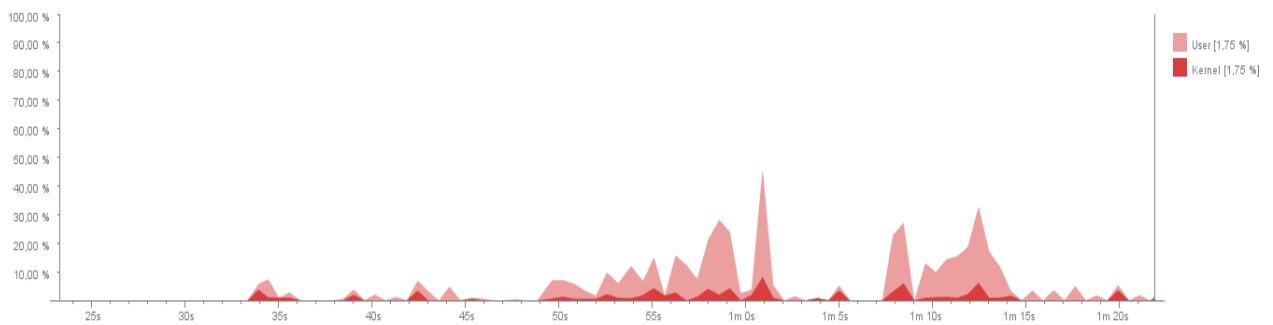
It is important to mention that the CPU usage is the percentage of the CPU that is being used by the process. By default `top` displays the percentage of a single CPU. On multi-core systems it is possible to have CPU usage greater than 100%. Since Raspberry Pi has 4 cores, the maximum value of CPU usage that is possible to have is 400% meaning that the four CPUs are being used at its maximum. To show the overall percentage of the available CPUs in use it is necessary to hit the `Shift i` while `top` is running. Since the command used above uses the `grep` to pipe the output and grab only the necessary data to print in the terminal it is not possible to hit `Shift i` while `top` is running. Despite that, it is known that Raspberry has 4 cores so the values shown in the figure 26 divided by four will give the overall percentage of the available CPUs being used. The maximum value in the figure above shown was 4% and therefore the maximum

overall CPU usage was 1%. The meaning of this number is that assuming that the four cores were one this interface would be using 1% of its time of execution which is a pretty good result. Also, there was no visual increase in the CPU usage when the TCP connections were established.

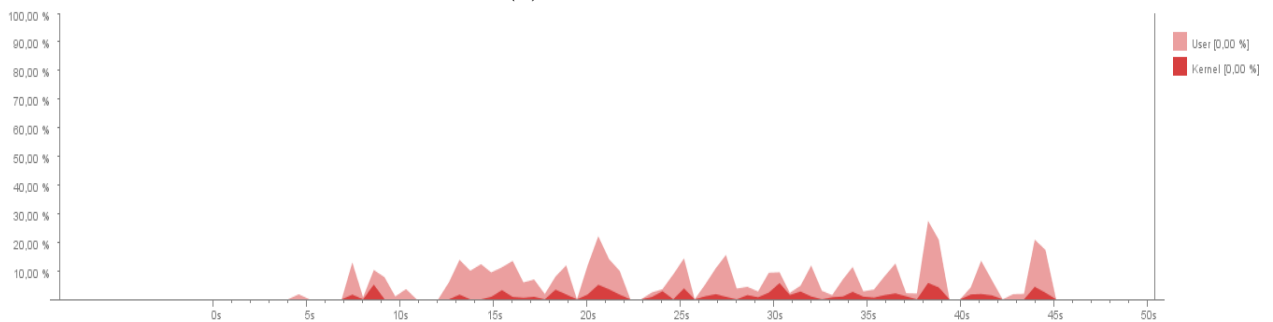
5.3.2 Android

For the android interface, since there is a GUI interface, the CPUs are used more than with Raspberry Pi's interface. However, this usage is not compromising the interface.

The figure bellow shows two CPU usage for two different activities:



(a) Monitor activity



(b) Stored Data Activity

Figure 27: Android interface - CPU usage

There are spikes present in the figures 27a and 27b. The spikes in the first figure are caused either by touching the graph, zoom in and zoom out or changing the filters, while in the second figure they are caused when the user clicks in an item in the list view being a particular data from the database loaded in the graph. It is important to mention that the other activities present in the Android interface showed even less CPU usage than this values shown. This interface does not overload the CPUs available.

5.3.3 QT

For the QT interface, the CPU usage is low for the Serial Port connection. In the figure bellow it is shown their values:

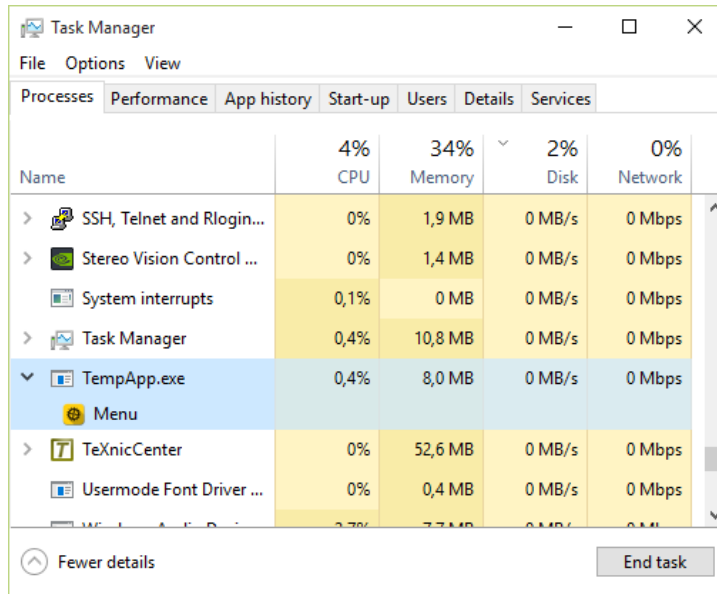


Figure 28: QT interface - Serial Port CPU usage

In the TCP connection the network usage is not zero like in figures 28 and 30 since there is a TCP connection established. In the figure bellow it is shown the CPU usage for this connection. It is higher than the UART but still in a low level.

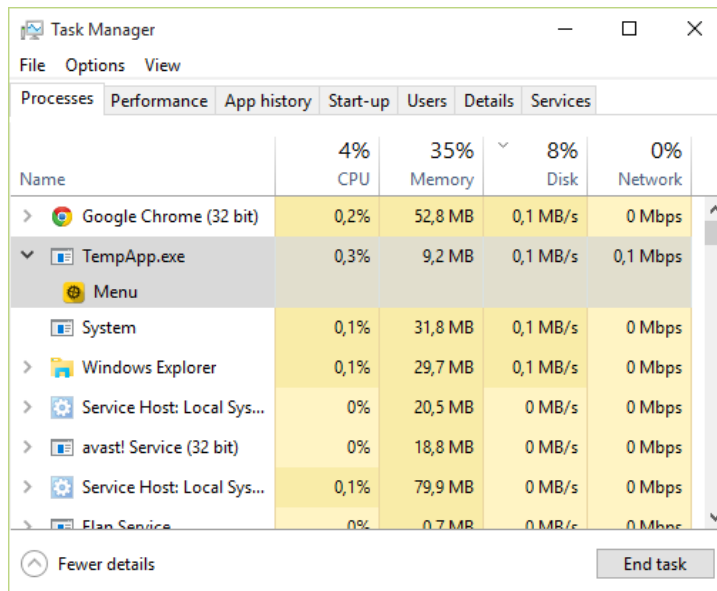


Figure 29: QT interface - TCP Client CPU usage

The figure 30 shows the maximum CPU usage obtained in the QT interface. This happens when a user double clicks in a row of the *QTableView*. This event triggers a *QThread* to retrieve

the values from the database for a particular id and insert them in the graph. Even though they are higher than the other two figures above shown it is still considerably low.

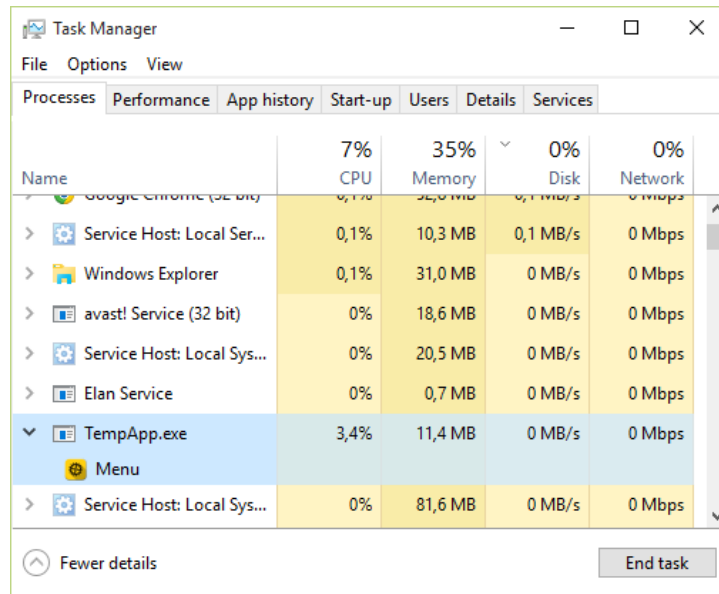


Figure 30: QT interface - View Data CPU usage

5.4 Discussion of the main results

The main results obtained were to be expected. It was accomplished a Raspberry Pi interface connecting different devices simultaneously in real time exchanging data. In addition, the “SMS alert” service demonstration was simulated showing that when a value is higher than a certain condition the SMS is sent to the default phone number. This demonstration was positive and the detection worked as expected. In terms of CPU usage the three interfaces were tested and the results were good since there is almost no overload of the CPUs. Also all the interfaces run smooth and allow a bunch of features that introduce great functionalities to the user.

In this work it was used different technologies, programming languages, platforms and Operating Systems. Most of the knowledge used in this work was introduced or acquired in this master degree. The rest of it was self-taught and improved during this work.

The objectives in this work have been achieved with great success. From a practical view, the main goal to use Raspberry Pi to connect multiple devices, with different platforms and Operating Systems was accomplished. Moreover, it is important to highlight that the work developed is a first approach in an area that has a high growing potential and that in terms of the main goal of this dissertation achieved its purpose.

Chapter 6

Conclusions and Future Work

”In the end, it’s not the years in your life that count. It’s the life in your years.”

Abraham Lincoln

In this chapter is presented the conclusion of this project and the future work that can be developed.

6.1 Conclusions

Three interfaces were developed in this study, a server interface, the Raspberry Pi interface, and two endpoint interfaces, the Android and the QT interfaces. The Raspberry Pi interface showed an incredible good performance allowing the connection between the server and multiple devices to exchange data. In the wire connection it was used serial port communication, while in the wireless connection it was used a TCP socket. The TCP protocol assures a reliable, ordered, and error-checked stream of data. Both the Android and QT interfaces were very efficient monitoring, analyzing and processing data in a graphical way.

The Raspberry Pi interface uses threading technology and synchronization methods to optimize the use of the Quad-Core CPU available in the Raspberry Pi. Although the type of sensor used to test this interface only included two types of variables, temperature and humidity, other type of sensors, which can include two or more types of variables, can also be used.

The Android interface uses the `AsyncTask` class from Android APIs to allow a proper and easy use of the working threads. This allows efficient communication with the GUI thread. With this interface, one can share the stored data with other devices via email. Moreover, a service named as “SMS alert” was created to notify the user once a preset value is reached. This allows the user to react to critical situations.

The QT interface uses a customized class that overrides the `QThread` class to manage threads, and a time-triggered routine to deal with the data coming from the Raspberry Pi. This interface is compatible with Windows, Linux and Mac Operating Systems. In opposite to

the Android interface, which only uses wireless communication, the QT interface allow the use of either wire or wireless communications.

6.2 Future Work

For future work it is considered two categories: i) The development of the software; ii) Considering other possible applications.

i) Development of a GUI for the Raspberry Pi interface; Use either an android phone or an USB 3G modem integrated with the Raspberry Pi to send the “SMS alert”. This integration would also permit the development of a new functionality, the “SMS info”. This functionality would allow the user to obtain information from the Raspberry Pi interface, by his request.

ii) Monitoring the security of a building by integrating a smoke or a motion sensor to the Raspberry Pi interface; Monitoring the energy consumption of a building by linking the Raspberry Pi with a power meter.

References

- [1] R. P. Brasil, “Raspberry pi 2 modelo b,” http://rasberrypibra.com/wp-content/uploads/2015/08/raspberry_pi_2_modelo_b_circuit_especificacoes_pinout.jpg, [Online; accessed 11-May-2015].
- [2] B. Pi, “What is banana pi?” <http://www.bananapi.org/p/product.html>, [Online; accessed 12-September-2015].
- [3] T. Instruments, “Beaglebone black development board,” <http://www.ti.com/tool/beaglebk>, [Online; accessed 12-September-2015].
- [4] M. Electronic, “Intel galileo gen 2 development board,” http://www.mouser.com/images/microsites/Intel_Galileo2_lrg.jpg, [Online; accessed 11-September-2015].
- [5] H. Kernel, “Odroid-c1,” http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141578608433&tab_idx=1, 2015, [Online; accessed 12-September-2015].
- [6] S. pcDuino3, “pcduino3,” <https://cdn.sparkfun.com//assets/parts/9/7/2/2/12856-01.jpg>, [Online; accessed 11-September-2015].
- [7] U. Q. Products, “Udoo quad,” <http://shop.udoo.org/eu/product/udoo-quad.html>, [Online; accessed 12-September-2015].
- [8] Manifest.permission, “Android apis reference,” <http://developer.android.com/reference/android/Manifest.permission.html>, [Online; accessed 31-May-2015].
- [9] R. P. Foundation, “What is a raspberry pi?” <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>, [Online; accessed 03-September-2015].
- [10] R. P. Products, “Raspberry pi 2 model b,” <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>, [Online; accessed 12-September-2015].
- [11] E. technical specifications, “Raspberry pi 2 model b,” <http://www.element14.com/community/docs/DOC-73827/1/the-new-raspberry-pi-2-model-b-1gb-technical-specifications>, [Online; accessed 12-September-2015].
- [12] J. Adams, “Introducing raspberry pi hats,” <https://www.raspberrypi.org/blog/introducing-raspberry-pi-hats/>, [Online; accessed 11-May-2015].

- [13] Adafruit, “Raspberry pi hats and plates,” <http://www.adafruit.com/category/286>, [Online; accessed 11-September-2015].
- [14] G. Henderson, “Software pwm library,” <https://projects.drogon.net/raspberry-pi/wiringpi/software-pwm-library/>, [Online; accessed 31-May-2015].
- [15] A. Palacherla, “Using pwm to generate analog output,” *Microchip*, 2002, datasheet reference: DS00538.
- [16] G. Coley, “Beaglebone black system reference manual,” *Beagle Bone*, 2013.
- [17] Intel, “Intel galileo gen 2 development board datasheet,” <http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-g2-datasheet.html>, 2015, [Online; accessed 12-September-2015].
- [18] I. Galileo, “Intel galileo gen 2 development board,” <http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html>, 2015, [Online; accessed 12-September-2015].
- [19] Sparkfun, “pduino3 - dev board,” <https://www.sparkfun.com/products/12856>, 2015, [Online; accessed 12-September-2015].
- [20] A. Jin, “Explanation of pduino3 headers,” <http://learn.linksprite.com/pduino/quick-start/explanation-of-pduino3-headers/>, 2014, [Online; accessed 12-September-2015].
- [21] U. Quad, “Udoo.rev.d gpio,” http://elinux.org/images/1/1f/Udoo_pinoutext.jpg, [Online; accessed 12-September-2015].
- [22] Ewell, “Banana pi dual core raspberry-like development board,” <http://www.amazon.com/Raspberry-Pi-like-devepment-Gigabit-ethernet/dp/B00LGXINGS>, [Online; accessed 12-September-2015].
- [23] Sparkfun, “Beaglebone black,” <https://www.sparkfun.com/products/12857>, [Online; accessed 12-September-2015].
- [24] I. G. G. 2, “Intel galileo gen 2,” <https://www.sparkfun.com/products/13096>, 2015, [Online; accessed 12-September-2015].
- [25] E. Upton, “Raspberry pi 2 on sale,” <https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/>, [Online; accessed 12-September-2015].

- [26] F. K. James and W. R. Keith, *Computer Networking - A Top-Down Approach*, 5th ed. Pearson, 2009.
- [27] A. References, “Async task class,” <http://developer.android.com/reference/android/os/AsyncTask.html>, [Online; accessed 09-August-2015].
- [28] Q. Documentation, “Qthread class,” <http://doc.qt.io/qt-4.8/qthread.html>, [Online; accessed 09-August-2015].
- [29] C. Objects, “Threading library,” <https://docs.python.org/2/library/threading.html#condition-objects>, [Online; accessed 09-August-2015].
- [30] A. A. Guides, “Android preferences,” <http://developer.android.com/guide/topics/ui/settings.html>, [Online; accessed 09-August-2015].
- [31] *Humidity and Temperature Sensor*, <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>, SparkFun.
- [32] M. Hawkins, “16x2 lcd module control with backlight switch,” <http://www.raspberrypi-spy.co.uk/2012/08/16x2-lcd-module-control-with-backlight-switch/>, [Online; accessed 31-May-2015].
- [33] Q. Documentation, “Qt for mac os x requirements,” <http://doc.qt.io/qt-4.8/requirements-mac.html>, [Online; accessed 31-May-2015].
- [34] L. registers, “Lcd modes and registers,” <http://esd.cs.ucr.edu/labs/interface/interface.html>, [Online; accessed 09-August-2015].

Configurations and setting up features

A.1 Finding the IP of Raspberry Pi

There is four options available: having a HDMI monitor, connecting the UART to a computer, accessing the router DHCP table or having a program to search all the IP addresses in use for a particular router. The first and second options show the boot messages from Raspberry Pi and it is printed its IP there. The third option is to access the router DHCP table, if it is allowed for the particular router, and search for the entry that has the name *raspberrypi* and its IP is there available. For the fourth option it can be used for example the program *Advanced IP Scanner*⁶.

A.2 VNC installation and configuration

To configure the VNC Server the following steps are needed:

1. Using any ssh program connect to Raspberry Pi
2. Provide the user (pi) and the password (raspberrypi)
3. Execute the command: `sudo apt-get install tightvncserver`
4. Execute the command: `tightvncserver` and then set a password
5. To start the server execute: `vncserver :0 -geometry 1600x900 -depth 24`

A.3 VNC in the boot

The steps are as follow:

1. Execute the command: `sudo bash`
2. Execute the command: `nano /etc/init.d/vncboot`
3. Copy and Paste the code bellow
4. Press CTRL+X to save the file and name it vncboot

⁶Available in: <http://www.advanced-ip-scanner.com/br/>.

5. Change the permissions of the file with: `chmod 755 /etc/init.d/vncboot`
6. Execute the command: `update-rc.d vncboot defaults`
7. Next time the Raspberry Pi is turned on the VNC Server will start automatically.

The following code is the one referred in the step 3:

```
1 #!/bin/sh
2 ### BEGIN INIT INFO
3 # Provides: vncboot
4 # Required-Start: $remote_fs $syslog
5 # Required-Stop: $remote_fs $syslog
6 # Default-Start: 2 3 4 5
7 # Default-Stop: 0 1 6
8 # Short-Description: Start VNC Server at boot time
9 # Description: Start VNC Server at boot time.
10 ### END INIT INFO
11
12 USER=root
13 HOME=/root
14
15 export USER HOME
16
17 case "$1" in
18   start)
19     echo "Starting VNC Server"
20     /usr/bin/vncserver :1 -geometry 1600x900 -depth 24
21     ;;
22
23   stop)
24     echo "Stopping VNC Server"
25     /usr/bin/vncserver -kill :0
26     ;;
27
28   *)
29     echo "Usage: /etc/init.d/vncboot {start|stop}"
```



```

30  exit 1
31  ;;
32  esac
33
34  exit 0

```

A.4 Port Forwarding

The first step is to connect to the router with an Ethernet cable or via wireless. It was used a TG585 v8 router and to access its GUI interface it is needed to use a browser and type *192.168.1.254*. After the page is loaded go to the separator *Game & Application Sharing*. Then it is necessary to open the port needed and to choose the type of protocol to open. In this case, as an example it was opened the port *5000* for TCP and UDP protocols as it is shown in the figure bellow:

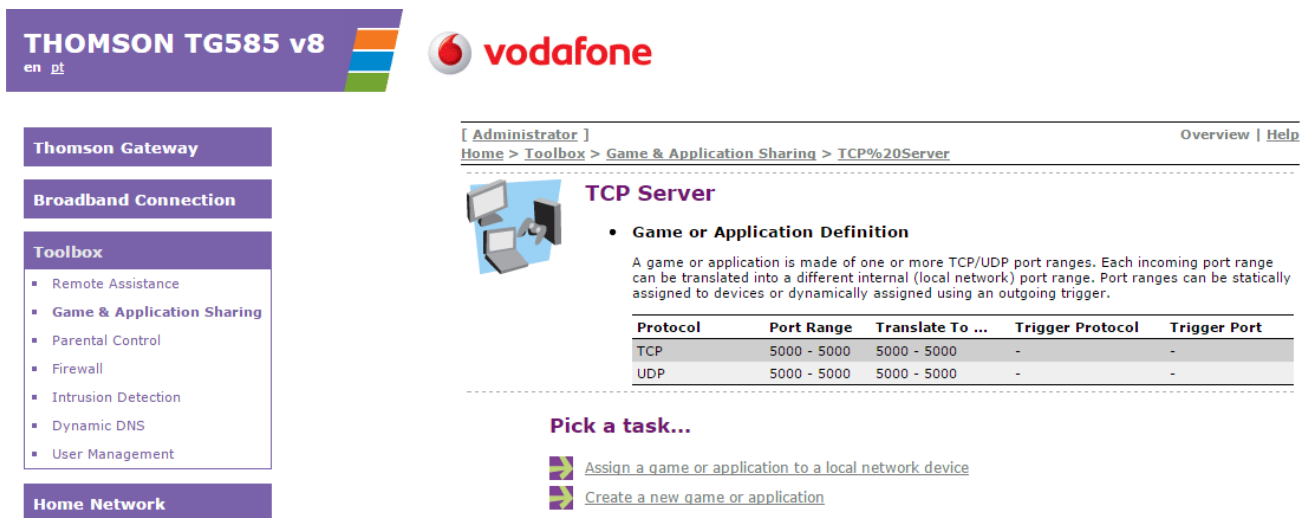


Figure 31: Setting up the port and the protocol

The second step is to link the IP address that will be associated with this service. For doing this click in the *Assign a game or application to a local network device* and choose the device needed. In this case it was chosen the Raspberry Pi, see figure 32. To verify that the port is open⁷ it is necessary to use the external address of the network (given in the website) and insert the port. If everything is well configured in the router it should appear in the end that the port is open as it is shown in the figure 33:

⁷Use this website: <http://www.yougetsignal.com/tools/open-ports/>.

Thomson Gateway

Broadband Connection

Toolbox

- Remote Assistance
- **Game & Application Sharing**
- Parental Control
- Firewall
- Intrusion Detection
- Dynamic DNS
- User Management

Home Network

[Administrator]

Overview | [Configure](#) | [Help](#)

Home > [Toolbox](#) > [Game & Application Sharing](#)



Game & Application Sharing

This page summarizes the games and applications defined on your Thomson Gateway. Each game or application can be assigned to a device on your local network.

- **Universal Plug and Play**

Universal Plug and Play (UPnP) is a technology that enables seamless operation of a wide range of games and messaging applications.

Use UPnP: Yes
Use Extended Security: No

- **Assigned Games & Applications**

The table below shows the games and applications that are allowed to be initiated from the Internet.

You need to configure such games or applications if you like to act as a game server or share a server located on your local network with other people.

If you are simply a player or simply accessing the Internet, you don't need to configure games or applications.

Game or Application	Device	Log
TCP Server	raspberrypi	Off

Pick a task...

- [Assign a game or application to a local network device](#)
- [Create a new game or application](#)
- [Modify a game or application](#)

Figure 32: Association of the device to the port forwarding

you get signal

Port Forwarding Tester

your external address
77.54.147.160

open port finder

Remote Address Port Number

Use Current IP

Port 5000 is open on 77.54.147.160.

Is your router causing you massive grief? Try picking up a cheap [Netgear N600](#) on [Amazon](#) or [Newegg](#). Since I bought one last year, I've never had to reboot it. Port forwarding is a breeze to setup.

If my tool has been helpful to you, check out my [desktop wallpaper](#) site or follow me on Twitter [@kirkouimet](#). :)

about

The open port checker is a tool you can use to check your external IP address and detect open ports on your connection. This tool is useful for finding out if your port forwarding is setup correctly or if your server applications are being blocked by a firewall. This tool may also be used as a port scanner to scan your network for ports that are commonly forwarded. It is important to note that some ports, such as port 25, are often blocked at the ISP level in an attempt to prevent malicious activity.

For more a comprehensive list of TCP and UDP ports, check out [this Wikipedia article](#).

If you are looking for a software solution to help you configure port forwarding on your network, try using this powerful [Port Forwarding Wizard](#).

help me pay for school (PayPal)

common ports

- 21 FTP
- 22 SSH
- 23 TELNET
- 25 SMTP
- 53 DNS
- 80 HTTP
- 110 POP3
- 115 SFTP
- 135 RPC
- 139 NetBIOS
- 143 IMAP
- 194 IRC
- 443 SSL
- 445 SMB
- 1433 MSSQL
- 3306 MySQL
- 3389 Remote Desktop
- 5632 PCAnywhere
- 5900 VNC
- 6112 Warcraft III
- Scan All Common Ports

Figure 33: Checking if port is open

A.5 Deactivate Raspberry Pi's UART

To deactivate the UART it is necessary to do the following commands:

1. Execute the command: *sudo nano /boot/cmdline.txt*
2. Change the line 1 to 2
3. Execute the command: *sudo nano /etc/inittab*
4. Comment (insert a #) or delete the line 3
5. Reboot the Raspberry Pi.

The lines above referred are:

1. *dwc_otg.lpm_enable = 0 console = ttyAMA0,115200 kgdboc = ttyAMA0,115200 console = tty1 root = /dev/mmcblk0p2 rootfstype = ext4 elevator = deadline rootwait*
2. *dwc_otg.lpm_enable = 0 console = tty1 root = /dev/mmcblk0p2 rootfstype = ext4 elevator = deadline rootwait*
3. *T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100*

A.6 QT in Linux based Operating Systems

To install QT 5.3.1 on 32-bit Linux computer it is needed to do:

1. Execute the command: *wget*
download.qt-project.org/official_releases/qt/5.3/5.3.1/qt-opensource-linux-x86-5.3.1.run
2. Execute the command: *sudo chmod +x qt-opensource-linux-x86-5.3.1.run*
3. Execute the command: *./qt-opensource-linux-x86-5.3.1.run*
4. Follow the instructions given in the installation wizard

After, execute the following commands to install a compiler and some libraries:

1. Execute the command: *sudo apt-get install g++*
2. Execute the command: *sudo apt-get install mesa-common-dev*
3. Execute the command: *sudo apt-get install libgl1-mesa-dev libglu1-mesa-dev*

A.7 WiringPi instalation for C/C++ Development

This is a library that allow to program the GPIO, available in the Raspberry Pi, in C/C++ language. It has a lot of features and it is very useful for developing.

The steps are as follow:

1. Execute the command: *sudo apt-get update*
2. Execute the command: *sudo apt-get upgrade*
3. Execute the command: *sudo apt-get install git-core*
4. Execute the command: *git clone git://git.drogon.net/wiringPi*
5. Execute the command: *cd wiringPi*
6. Execute the command: *./build*

A.8 RPi.GPIO installation for Python Development

This library allows to program the GPIO available in the Raspberry Pi in Python language. To install it follow the steps bellow:

1. Execute the command: *sudo apt-get update*
2. Execute the command: *sudo apt-get upgrade*
3. Execute the command: *sudo apt-get install python-dev python-rpi.gpio*

Tutorials and examples

B.1 QT Examples

B.1.1 Serial Ports Names in QT

In this section, it is shown a simple example on how to create a function to insert in a QComboBox the list of available serial port names connected to a certain computer. This function should be called either in the creating time of the GUI window or in a button click event to allow a user to refresh the list.

The code bellow shows that function:

```
1 ...
2#include <QtSerialPort/QSerialPort>
3#include <QtSerialPort/QSerialPortInfo>
4
5//Assuming that in the GUI file there is a
6//QComboBox with the name port_name
7
8 ...
9void refresh_serial_port ()
10 {
11     // Clears the QComboBox
12     ui->port_name->clear ();
13
14     // Inserts the serial port names found
15     foreach (const QSerialPortInfo &info ,
16             QSerialPortInfo::availablePorts ())
17     {
18         ui->port_name->addItem (info.portName ());
19     }
20 }
21 ...
```

B.2 Python Examples

B.2.1 Email

This example shows how to send emails using Raspberry Pi and a gmail account. To be able to use this example it is necessary to allow in the security setting of the gmail account this type of usage.

```
1 #Import smtplib fot the sending function
2 import smtplib
3
4 # Import email modules
5 from email.mime.text import MIMEText
6
7 to = 'target_to_send_email@gmail.com'
8 gmail_user = 'temp.app.server@gmail.com'
9 gmail_password = 'temperatura'
10 smtpserver = smtplib.SMTP('smtp.gmail.com',587)
11
12 smtpserver.ehlo()
13 smtpserver.starttls()
14 smtpserver.login(gmail_user , gmail_password)
15
16
17 msg = MIMEText(" Hello! This is an automatic email from Raspberry Pi.")
18 msg['Subject'] = "Temperature Application"
19 msg['From'] = gmail_user
20 msg['To'] = to
21 smtpserver.sendmail(gmail_user , [to], msg.as_string())
22
23 smtpserver.quit()
```

B.2.2 Serial Port

In this example the serial port is opened and on success it send the numbers from 1 to 49. On failure or if it is detected a *CTRL-C* the program is terminated.

```
1 import serial
2
3 #To read a line use the line bellow
4 #serial_port.readline()
5
6 try:
7   #Open Serial Port
8   serial_port = serial.Serial("/dev/ttyAMA0", baudrate=115200,
9 timeout=3.0)
10
11   for i in range(1,50):
12     serial_port.write(str(i)+"\n")
13
14 except KeyboardInterrupt:
15   print " User_terminated_the_program_with_CTRL+C!\n"
16
17 except serial.SerialException:
18   print " Serial_Exception!"
19
20 finally:
21   serial_port.close
22   print " Closed_Serial_Port!"
```

B.2.3 LED Turn On/Off

In this example it is shown how to turn on a LED and then after four seconds turn it off.

```
1 import RPi.GPIO as GPIO
2 import time
3
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(4, GPIO.OUT) ## GPIO4 Output
```

```

6 GPIO.output(4,True) ## Turn on
7
8 try:
9     time.sleep(4) ## Wait 4 segundos
10    GPIO.output(4,False) ## Turn off
11
12 except KeyboardInterrupt:
13     print " User_terminated_the_program_with_CTRL+C!\n"
14
15 except:
16     print " Other_exceptions_were_caught"
17
18 finally:
19     GPIO.cleanup()

```

B.2.4 LCD

In this example it imported a developed library called *lcd.py*. The following code is the *main.py* and below this code is the library code.

```

1 import lcd
2 import time
3 import RPi.GPIO as GPIO
4
5 def main():
6
7     lcd.lcd_init()
8     lcd.lcd_string_line_one(" Raspberry_Pi")
9     lcd.lcd_string_line_two(" is_the_best:P")
10    time.sleep(3)
11
12 if __name__ == '__main__':
13
14     try:
15         main()
16     finally:

```



```
17 lcd.clear()
18 lcd lcd_string_line_one(" Bye_Bye!")
19 GPIO.cleanup()
```

Here is the library, *lcd.py*, to use the LCD:

```
1 import RPi.GPIO as GPIO
2 import time
3 # Timing constants
4 E_PULSE = 0.0005
5 E_DELAY = 0.0005
6
7 #GPIO pins
8 LCD_RS = 21
9 LCD_E = 20
10 LCD_D4 = 16
11 LCD_D5 = 12
12 LCD_D6 = 7
13 LCD_D7 = 8
14
15 LCD_WIDTH = 16 # Maximum characters per line
16 LCD_CHR = True # Character Mode
17 LCD_CMD = False # Comand Mode
18 LCD_LINE_1 = 0x80 # LCD RAM address - 1st line
19 LCD_LINE_2 = 0xC0 # LCD RAM address - 2nd line
20
21 def lcd_init():
22     GPIO.setmode(GPIO.BCM)
23     GPIO.setup(LCD_E, GPIO.OUT)
24     GPIO.setup(LCD_RS, GPIO.OUT)
25     GPIO.setup(LCD_D4, GPIO.OUT)
26     GPIO.setup(LCD_D5, GPIO.OUT)
27     GPIO.setup(LCD_D6, GPIO.OUT)
28     GPIO.setup(LCD_D7, GPIO.OUT)
29
30 # Initialise display
```

```

31 lcd_byte(0x33,LCD_CMD)
32 lcd_byte(0x32,LCD_CMD)
33 lcd_byte(0x06,LCD_CMD) # Cursor move direction
34 lcd_byte(0x0C,LCD_CMD) # Display On, Cursor Off, Blink Off
35 lcd_byte(0x28,LCD_CMD) # Data length , number of lines , font size
36 lcd_byte(0x01,LCD_CMD) # Clear display
37 time.sleep(E_DELAY)
38
39 def lcd_byte(bits , mode):
40     # Send byte to data pins
41     # bits = data
42     # mode = True for character
43     #       False for command
44
45     GPIO.output(LCD_RS, mode)
46
47     # High nibble
48     GPIO.output(LCD_D4, False)
49     GPIO.output(LCD_D5, False)
50     GPIO.output(LCD_D6, False)
51     GPIO.output(LCD_D7, False)
52     if bits&0x10==0x10:
53         GPIO.output(LCD_D4, True)
54     if bits&0x20==0x20:
55         GPIO.output(LCD_D5, True)
56     if bits&0x40==0x40:
57         GPIO.output(LCD_D6, True)
58     if bits&0x80==0x80:
59         GPIO.output(LCD_D7, True)
60
61     lcd_toggle_enable() # Toggle Enable
62
63     # Low nibble
64     GPIO.output(LCD_D4, False)
65     GPIO.output(LCD_D5, False)

```

```

66 GPIO.output(LCD_D6, False)
67 GPIO.output(LCD_D7, False)
68 if bits&0x01==0x01:
69     GPIO.output(LCD_D4, True)
70 if bits&0x02==0x02:
71     GPIO.output(LCD_D5, True)
72 if bits&0x04==0x04:
73     GPIO.output(LCD_D6, True)
74 if bits&0x08==0x08:
75     GPIO.output(LCD_D7, True)
76
77 lcd_toggle_enable() # Toggle Enable
78
79 def lcd_toggle_enable():
80     # Toggle enable
81     time.sleep(E_DELAY)
82     GPIO.output(LCD_E, True)
83     time.sleep(E_PULSE)
84     GPIO.output(LCD_E, False)
85     time.sleep(E_DELAY)
86
87 def lcd_string(message, line):
88     message = message.ljust(LCD_WIDTH, " ")
89     lcd_byte(line, LCD_CMD)
90
91     for i in range(LCD_WIDTH):
92         lcd_byte(ord(message[i]), LCD_CHR)
93
94 def lcd_string_line_one(message):
95     lcd_string(message, LCD_LINE_1)
96
97 def lcd_string_line_two(message):
98     lcd_string(message, LCD_LINE_2)
99
100 def clear():

```

```
101 lcd_byte(0x01, LCD_CMD)
```

B.3 C/C++ Examples

Using the library wiringPi, see appendix A.7 to know how to install it, this appendices are meant to show some examples on how to use it.

B.3.1 LED with button - Pull Up/Down pin:

In this example, instead of using an external resistor to pull up/down the button it is used an internal resistor inside Raspberry Pi's pins. The pull up/down makes sure that the potential of the button will be either 0 or 5 volts. This is used to avoid the floating phenomena (unknown state) and assure that the pin will be in either low or high state.

This example turns on the led depending on wich pull system was used.

```
1#include <stdio.h>
2#include <wiringPi.h>
3
4const int ledPin = 17; //GPIO17
5const int butPin = 22; //GPIO22
6
7int main(void)
8{
9
10 wiringPiSetupGpio();
11 pinMode(ledPin, OUTPUT);
12 pinMode(butPin, INPUT);
13
14 // PULL-UP Resistor
15 // When it is clicked it reads 0V
16
17 // PULL-DOWN resistor
18 // When it is clicked it reads 5V
19 //pullUpDnControl(butPin, PUD_DOWN);
20
21 pullUpDnControl(butPin, PUD_UP); // PULL-UP
```

```

22
23 while(1)
24 {
25     if(digitalRead(butPin))
26     {
27         digitalWrite(ledPin , LOW);
28     }
29     else
30     {
31         digitalWrite(ledPin , HIGH);
32     }
33 }
34
35 return 0;
36 }

```

B.3.2 PWM - LED brightness

In this example, it is written in the PWM register a value from 0 to 1023 that changes the duty cycle of the wave. The effect is that we see the LED brightness change.

```

1 #include <stdio.h> // printf()
2 #include <stdlib.h>
3 #include <wiringPi.h>
4
5 const int ledPin = 18;
6
7 int main(void)
8 {
9     int bright;
10
11     wiringPiSetupGpio();
12
13     pinMode(ledPin , PWM_OUTPUT);
14
15     while(1)

```

```

16 {
17     for (bright = 0 ; bright < 1024 ; bright++)
18     {
19         pwmWrite(ledPin , bright) ;
20         delay(1);
21     }
22
23     for (bright = 1023 ; bright >= 0 ; bright--)
24     {
25         pwmWrite(ledPin , bright) ;
26         delay(1);
27     }
28 }
29
30 return 0;
31 }

```

B.3.3 LCD - 4-bit Mode

In this example, it is shown how to use the LCD to print to its screen the string “Hello World!”. It was used the 4-bit mode, for using the 8-bit mode it is necessary to use more four wires, change the variable *LCD BIT MODE* to 8 and specify the pins for the extra four wires.

```

1#include <stdio.h>
2#include <wiringPi.h>
3#include <lcd.h>
4
5const int LCD_ROWS = 2;
6const int LCD_COLS = 16;
7const int LCD_BIT_MODE = 4; // 4-bit Mode
8const int LCD_RS = 7; //GPIO7
9const int LCD_E = 8; //GPIO8
10const int LCD_D4 = 17; //GPIO17
11const int LCD_D5 = 18; //GPIO18
12const int LCD_D6 = 27; //GPIO27
13const int LCD_D7 = 22; //GPIO22

```

```

14
15 int main(void)
16 {
17
18     int lcd; // handle
19     wiringPiSetupGpio();
20
21     lcd = lcdInit(LCD_ROWS, LCD_COLS, LCD_BIT_MODE, LCD_RS, LCD_E,
22 LCD_D4, LCD_D5, LCD_D6, LCD_D7, 0, 0, 0, 0);
23
24     if(lcd == -1)
25     {
26         printf("lcdInit_failed!\n");
27         return -1;
28     }
29
30     lcdPosition(lcd,0,0); // posiciona na posicao 0x0
31     lcdPuts(lcd,"Hello World!");
32     getchar();
33     lcdClear(lcd);
34
35 return 0;
36 }

```

B.3.4 LED Blinking Effect

This example shows how to use digital ports to turn on and off a LED and make a blinking effect.

```

1 #include <stdio.h>
2 #include <wiringPi.h>
3
4 const int ledPin = 17; //GPIO17
5
6 int main(void)
7 {

```

```

8
9  wiringPiSetupGpio ();
10
11 pinMode(ledPin , OUTPUT);
12
13 printf(" Press _CTRL+C_ to _quit_." );
14
15 while (1)
16 {
17     digitalWrite(ledPin , HIGH);
18     delay(75); // wait 75ms
19     digitalWrite(ledPin , LOW);
20     delay(75); // wait 75ms
21 }
22
23 return 0;
24 }

```

B.3.5 Makefile to compile the examples

This is the makefile that can be used to compile the programs above mentioned:

```

1 main: main.o
2     gcc -Wall -W -Werror -o main main.o -l wiringPi
3 main.o: main.c
4     gcc -Wall -W -Werror -c main.c

```


Useful information

C.1 TCP Connection

The client establish the initial contact with the server and initiates a TCP connection, this is done by creating a socket. Once the socket has been created in the client program, TCP in the client initiates a three hand-shake. During the three-way handshake, the server creates a socket dedicated to a particular client. In the end of the handshaking phase, a TCP connection exist between the client's socket and the server new socket and data can be exchanged.

Inside the TCP header, there are two important bits ACK and SYN. This two have an important role in this process. In the three-way handshake, see the figure 34, three packets are exchanged. The first one is a TCP packet with the SYN bit on (set to 1) from the client to the server, when the server receives it sends a packet to the client with the SYN and ACK bits on. When the client receives this packet sends a last packet with the ACK bit on and when the server receives this last message the connection is established.

For more information about three hand-shake, please refer to [?].

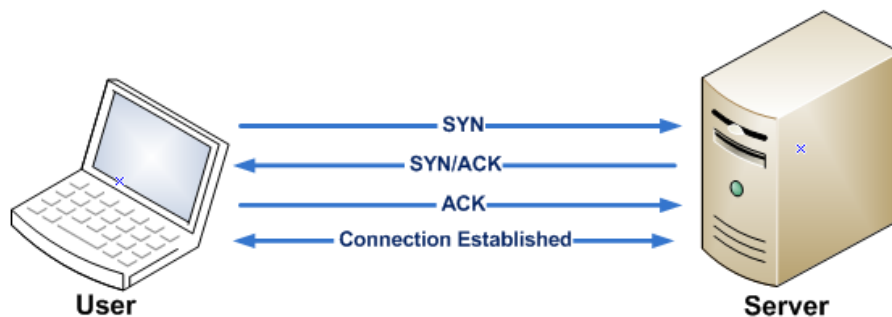


Figure 34: Three-way handshake

C.2 Multiple Readers/One Writer Problem

The well-known problem of readers and writers has a vast number of sub-problems, one of them is the multiple readers and one writer. This problem happen when concurrent programming is used and synchronization is strictly necessary between readers and writers.

The easy way to understand this problem is to imagine a scenario in which there are two sets of threads, the writer and the readers and there is a shared variable between them. The first set can write data and the second read it accessing the shared variable.

If the shared variable is open for reading, no reading thread should be kept waiting to access the variable to be able to read. If the writer is writing no reader is allowed access the variable.

In this problem there is a vast number of approaches to avoid starvation that happens when a thread waits indefinitely for some resources, but other threads are using it, to increase fairness or to favor or readers or writers.

C.3 LCD Modes for Character Display

In order to operate with LCD displays, it is necessary to choose which mode we want to operate with the LCD. There is two different ways to communicate with it. There's the 4 and 8 bits.

C.3.1 8 bits Mode

In 8-bit mode, it is required 8 wires for sending a whole byte of data at once and three control lines: enable(EN), register Select (RS) and read/write(RW).

The basic procedure is to prepare all other lines, and then pulse the enable line high for a short while in which LCD reads the command sent (when RW is low) or writes data (when RW is high). For control messages, RS line is low, and for writing characters, RS line is high. In this work the RW is connected to the Ground since it is only needed to send commands.

First it is necessary to send the following commands to initialize the LCD: 0x30, 0x30 and again 0x30; which means three times 00110000 in binary.

It is important to mention that after this three commands it is required to send the next byte in the following format: 0 0 1 DL N F; where DL is to set the interface (1 in binary for 8-bit mode - DB4), N for number of lines of the display (2 bits) and F for the font (2 bits).

The figure bellow shows the procedure to do the initialization and to operate the LCD in this mode:

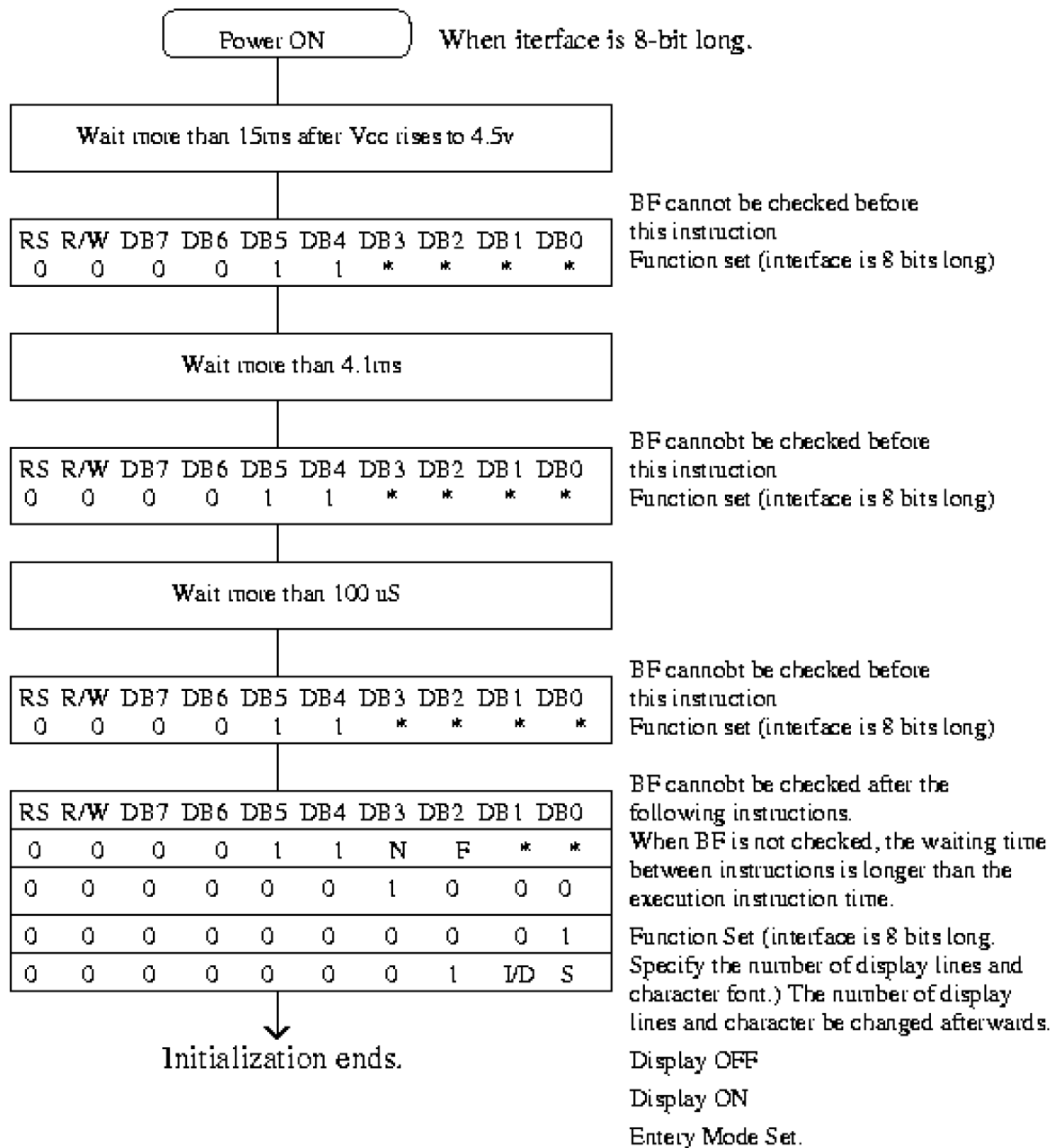


Figure 35: LCD 8-bit Mode

C.3.2 4 bits Mode

This mode is an approach that uses 4 wires to do the communication. The basic principle is the same as 8-bit but instead of sending a whole byte at once it is necessary to send first the higher nibble and then the lower nibble and the reconstruction of the bits to make a byte is done by the LCD.

Having a function that receives a whole byte and then sends the first nibble, toggle the enable line and after sends the lower nibble is advised.

So the first commands are 0x33 and then 0x32. To configure it to work in 4-bit and to set up a 2x16 LCD, two lines with sixteen characters each, the following byte needs to be sent: 0 0 1 DL N F; where DL is to set the interface (0 in binary for 4-bit mode - DB4), N is the number of lines of the display (two lines so 10 in binary) and F is the font and we choose the default one (00 in binary). So the upper nibble is 0x02 and the lower nibble is 0x08 and the whole byte is 0x28.

To use the LCD after those steps it is only necessary to know the registers associated with each action.

The figure bellow shows the procedure to do the initialization and to operate the LCD in this mode:

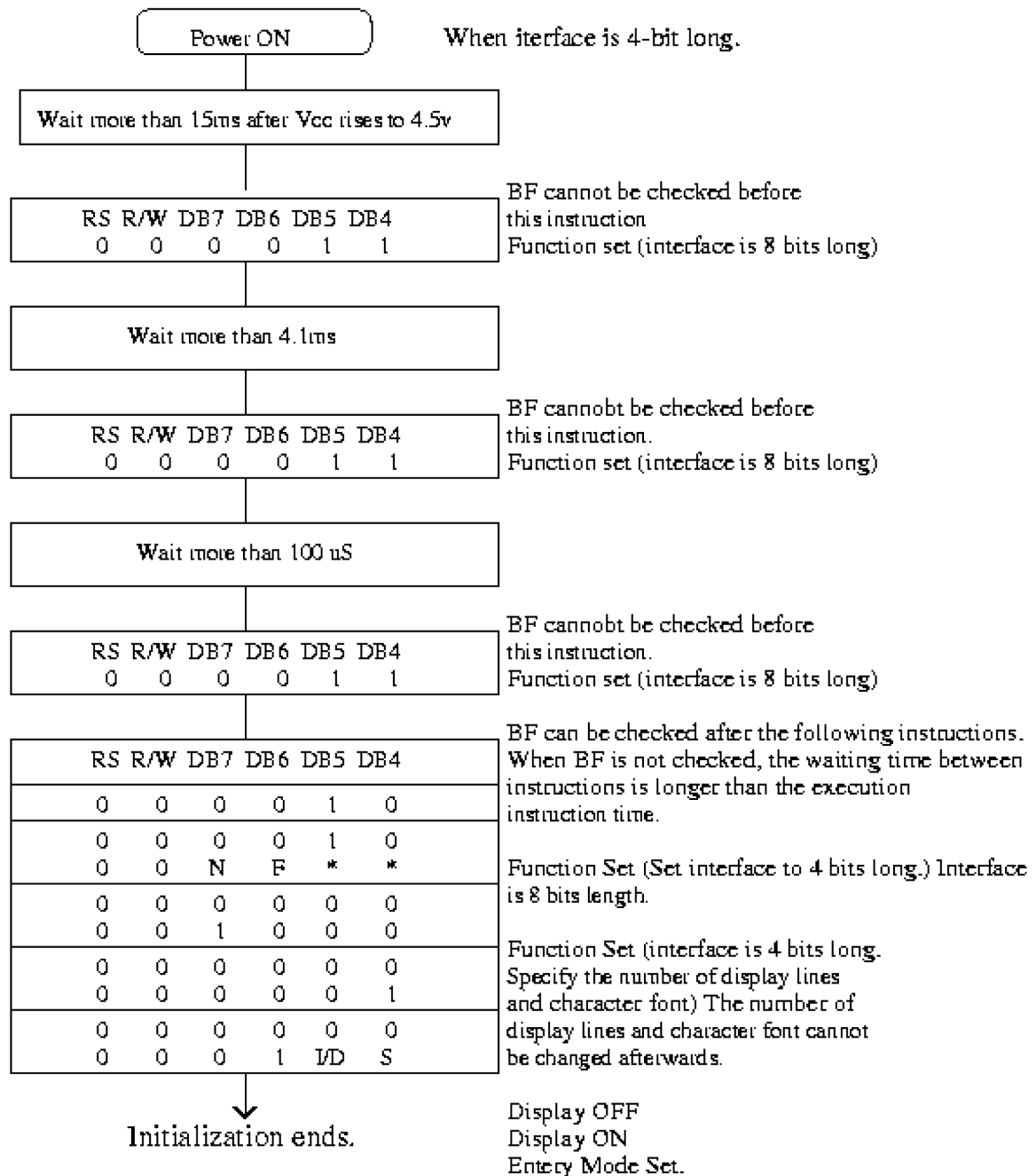


Figure 36: LCD 4-bit Mode

C.3.3 Comparison of the two modes

Comparing the two modes, 8-bit is slightly faster however it is impossible to notice the difference to human eyes since the transmission time is relatively low.

In terms of wires 4-bit has only four so it is better to use this method for devices with limited I/O ports.

For information about the registers please refer to [?].

Acronyms and symbols

Abbreviation	Meaning
ACK	Acknowledge
ADC	Analog-to-Digital Converter
API	Application Programming Interface
CAN	Controller Area Network Protocol
COM	Communication
CSI	Camera Serial Interface
DAC	Digital-to-Analog Converter
DHCP	Dynamic Host Configuration Protocol
HAT	Hardware Attached on Top
FK	Foreign Key
GPIO	General - Purpose Input/Output
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
I ² C	Inter-Integrated Circuit
JAR	Java Archive
LCD	Liquid Crystal Display
OS	Operating System
PHP	Hypertext Preprocessor
PK	Primary Key
PNG	Portable Network Graphics
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SD Card	Secure Digital Card
SMS	Short Message Service
SoC	System-on-Chip
SQL	Structured Query Language
SSH	Secure Shell
SYN	Synchronize
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
USB	Universal Serial Bus
VNC	Virtual Network Computing
XLSX	Microsoft Excel Open XML Spreadsheet
XML	Extensible Markup Language