

Carlos García Martínez

# PROTOTYPING DELAYED MIRROR USING RASPBERRY PI

Bachelor's thesis  
Information Technology

2017



South-Eastern Finland  
University of Applied Sciences

<b>Author (authors)</b>  Carlos García Martínez	<b>Degree</b> Bachelor of Information Technology	<b>Time</b>  December 2017
<b>Title</b>  Prototyping delayed mirror using Taspberry Pi.		67 pages
<b>Commissioned by</b>  Universidad de Málaga		
<b>Supervisor</b>  Osmo Ojamies		
<b>Abstract</b>  <p>This thesis has a main goal, to make technology more reachable for users related with performing arts, and sports environments. These users, will be able to interact with the developed model based on embedded system as Raspberry Pi 3B and C++ based software, as openFrameworks, combination.</p> <p>The objective is also to improve the knowledge in electronic technologies which can be associated to Raspberry, and then to expand the functions of it, to be able to apply them into the main goal previously described.</p> <p>Also the learning about SPI connection and how can be used in this situations, it is definitely a good way to grow up in technologies. This method will be used in order to connect the devices which produce bus signals and have them into the Raspberry Pi, having, in this way, the option to work with them.</p> <p>Finally it is important in this Thesis the way to receive analog signals into Raspberry. In order to do this, it will be used a analog/digital converter chip, MCP3008, to get them, as digital ones, and have the information into our ARM system. Then process these signals and translate them into new signals through the different components of the system.</p> <p>In particular, this system will show the image recorded by a USB camera connected to it, but with a certain delayed which can be controlled, by a physical interface designed to it.</p>		
<b>Keywords</b>  OpenFrameworks, Raspberry Pi, ARM, Mirror, SPI, MCP3008		

# CONTENTS

LIST OF TABLES .....	5
1 INTRODUCTION.....	6
1.1 Motivation.....	6
1.2 Main Objectives.....	6
1.2.1 Target output.....	7
1.2.2 Learning objectives .....	8
1.2.3 Methodologic objectives.....	9
2 STATUS OF TECHNIQUE.....	10
2.1 Tools.....	10
2.1.1 Raspberry.....	10
RASPBERRY PI 3 MODEL B.....	12
G.P.I.O.....	13
RASPBIAN .....	16
2.1.2 Open Frameworks.....	17
ADD-ONS.....	19
2.1.3 Analog / Digital Conversion .....	21
OPTION I <sup>2</sup> C.....	21
OPTION SPI.....	23
3 REQUIREMENTS .....	28
3.1 What is really a requirement .....	28
3.1.1 Definition and attributes of requirements.....	28
3.1.2 Requirements groups: functionals and no functionals.....	30
3.2 System requirements .....	31
3.3 Architecture .....	35
4 HARDWARE DEVELOPEMENT.....	36

4.1	Physic architecture .....	36
4.1.1	Camera subsystem .....	38
4.1.2	User interface subsystem .....	39
4.1.3	SPI Subsystem .....	40
4.1.4	Processor subsystem .....	41
4.1.5	Display output System.....	42
4.2	Implementation.....	42
5	SOFTWARE DEVELOPMENT .....	45
5.1	Introduction .....	45
5.2	Program Structure .....	45
5.3	Implementation.....	46
5.3.1	Input Functions.....	50
5.3.2	Processing functions .....	57
5.3.3	Output functions .....	58
6	TESTING.....	59
7	CONCLUSSIONS AND FUTURE LINES.....	63
7.1	Conclussions.....	63
7.2	Future lines .....	65
	REFERENCES.....	66

## LIST OF TABLES

Table 2-1 Improvements of Raspberry Pi 3.....	13
Table 2-2 Transmission modes of I2c [13] .....	23
Table 2-3 Operating modes in SPI .....	25
Table 3-1 Table of Requirements .....	34
Table 6-1 Test 1. Requirement 1.....	60
Table 6-2 Test 2. Requirement 1.1.....	60
Table 6-3 Test 3. Requirement 1.4.....	60
Table 6-4 Test 4. Requirement 1.5.....	61
Table 6-5 Test 5. Requirement 2.....	61
Table 6-6 Test 6. Requirement 2.1.....	62
Table 6-7 Test 7. Requirement 2.2.....	62
Table 6-8 Test 8. Requirement 2.3.....	63

## **1 INTRODUCTION**

### **1.1 Motivation**

The occidental world is in the middle of a technologic era. Nevertheless, there are still multiple areas where electronics are still not involved or are starting to be in there, but anyway, so much to explore is left. For instance, technology in sportive areas, and its relationship with health is one of the fields with more technological development nowadays. [1]

Artistic scenarios are also one of those environments where there is so much to do regarding technology. There are, actually, some applications which support activities that are developed in a dance academy, but most of them are related with the business or administrative area of it, but not with the part of learning or teaching flows of the dance itself. This situation is quite different in other fields as plastics arts or music, where there are plenty of tools what are able to support every related activity.

With this point of view, and taking in value the technological *hole* in this area, currently, we can say that technologies which can be developed in order to help in this environment, have a big margin of evolution, because of the absence of precedents that can be used as base points, then it is possible to say that all is still about development.

### **1.2 Main Objectives**

This project aims to make the technology more accessible in certain fields of the society, where its use is still at a really basic level or not even used at all. This could be because of the need has not appeared still or a specific use is not found until now. A good example of this could be the one regarding the practice of a sport in where the own-learning is involved, but there is not any interactive element which help with the development of the subject. It is in this case, where this thesis is more related, especially in those fields which have a relationship with the dance studios,

and the technology what can be used there to improve the practice in them from users. Also, this could be extendible to another areas as sports or improvement in the individual technique or another physical activities

### **1.2.1 Target output**

As main objective, with this prototype it is pretended to give to the user, autonomously, a direct feedback of his/her own development in the learning of a technique.

In summary, this device will generate a repetition of the movements from user with a variable delay, so in that way, the user could be able to have a more detailed vision of his/her exercise, and then to be able to recognize the parts what can be fixed or improved, in the same way than the ones that have reached the requirements, in a faster and trustable way.

This repetition of movements will be projected in some way that the user will be able to see his/her own work in a quick sight, without changing the positions which is being practiced and then, without lose the work flow. This will help to the learning too, because there is no time lost regarding post checking.

Another objective of this thesis is to make the prototype as portable as possible. In this way, the user could be take it and move it easily in order to use it in whatever environment available without the need of being in a specific environment designed for its use, hence, providing to the user a more comfortable feeling.

Following this way, the goal that is being tried to reach is to demonstrate that thanks to embed-based systems technology, users can have a greater learning capacity. This is, to optimize aspects as:

Speed. It pretends to improve the time when the user can see by itself how big is the evolution of his or her work, because this pretends to make the user more self-sufficient. Before this, the space was limited to the environments designed for this

kind of activities. After the implementation of this prototype, the spaces available will grow to the ones where the image can be projected.

### 1.2.2 Learning objectives

In general, this thesis aims to advance the learning of the use of a software programming environment, to achieve the development of tasks related with electronic devices, which facilitates the development of various activities carried out by the individual, more specifically activities in environments artistic or sports, where there will be a clear user interaction machine.

On the other hand, one of the main objectives in terms of learning is to extend the use of commercial systems based on microcontrollers, deepening their study and possible outputs or uses in environments still unknown to them, such as are the sports or artistic facilities. The ability to be programmable is mainly required, in such a way that its functioning can be adapted to what is sought in this project. For this, it is intended to deepen in the use of Rasberry Pi. To deepen in the study of its different configurations, as well as in the code development adapted to it and thus allowing the use of the device with different external elements.

In this last aspect, it is about increasing the knowledge in programming languages oriented to objects suitable for this system, as is the case of C ++. And, more specifically, it is intended to increase the use of tools based on this type of language, in order to achieve more user-friendly programming environments for the end user, that is, to try that, at least from the point of view of its programming, this project can be developed, or expanded, in the most pleasant way possible, also for users who are further away from the electronic world. A tool that quite well fulfills this type of feature can be *openFrameworks*.

In addition, the knowledge of the electronics that make up this project, as well as basic principles of it, are indispensable tools for the development of a prototype that meets these characteristics, since they must take into account various factors that allow proper use and operation of the different components that are part of said system. That is, continuing to grow in the knowledge of electronic devices,



their use, and their study is undoubtedly part of one of the objectives that are intended to be carried out.

On the other hand, at present, the world of the 3D Printer is becoming more and more important for the development of prototypes, and small projects. That is why, another of the objectives to be reached is the use of this type of technology for projects related to electronics. This includes, therefore, the previous study and design in different software to arrive at the final result that will be an element printed by this type of printers, and that is also useful for the prototype.

In summary, the aim is to extend the knowledge in the programming of microcomputer systems, by means of which, combining them with the appropriate electronics, it is possible to carry out activities in artistic or sports facilities in the simplest way for the end user. That is, to make a development in code with the necessary complexity such that the use of the device is facilitated to the average user, and in this way facilitate the possible commercialization of the same.

### **1.2.3 Methodologic objectives**

In the development of this project, we have tried to follow a methodology based on requirements. Which allow to the developer to have a clear idea of how far it is intended, that is how much is possible to reach following this line of study. Requirements, can give us even the guidelines that must be followed for this project to meet the needs that were initially intended to cover.

When talking about requirements, it is necessary to divide into two groups, functional, and non-functional.

The functional requirements, as its own name indicates, are those that the final device must comply with in terms of its operation, while the non-functional requirements are based on the different functionalities that the system must possess or fulfill internally or externally, but that do not affect in such an important way the main operation of the device, but are more typical of the system on which this particular project is based. [2]

In this sense, it can be said that the prototype on which this development is based has a higher level of non-functional requirements, since they are what make it unique, and they distance it from the conventional device, such as a video capture system. In Chapter 3 each one of them will be seen with a greater level of detail.

Once the requirements have been identified, a possible architecture for the system is studied. This results in a study of the different components that can be used to satisfy the functional requirements, without neglecting the non-functional ones, such as viability in the acquisition of the components, the lowest possible cost, and of course the compatibility between all the elements of the system.

The next step focuses on the development of the system software. This is done iteratively, based on functional requirements that, due to their simplicity, serve to test the components in a unitary manner, to later integrate the full functionality of the system.

Whenever a new component is added, a copy of the code is saved so that, in this way, it is possible to ensure the complete functioning of the system at each moment of the development, even if the required final operation has not been reached.

## **2 STATUS OF TECHNIQUE**

### **2.1 Tools**

#### **2.1.1 Raspberry**

The Project also known as Raspberry Pi, started in 2006, thanks to Eben Upton [5], Rob Mullin, Jack Lang and Allan Mycroft. This moment, is when they decided to create a small computer and simple use oriented mainly to children in educational centers, to increase the knowlege of computing from a younger age. In 2009 the charitable organization Raspberry Pi Foundation [6] was created. The exit to the market did not take place until the middle of the year 2012, when it was thrown out internationally with two models A and B.

The main difference between both models is found in their RAM (*Random Access Memory*) and the functions presented by each of them. The model A has a simpler use, with less RAM, only 256Mb (*Mega bytes*) and almost without output ports, while model B includes an increase of twice the RAM, in addition to integration of a USB port (*Universal Serial Bus*), thanks to the incorporation of a integrated HUB, and an Ethernet 10/100 Mbit (*Mega bits*) port associated to the integrated HUB already mentioned for the conection with RJ-45.

Starting the year 2014, 2 new versions of Raspberry Pi A+ and B+ are thrown out, that replace their predecessors by integrating new elements, as digital input and output ports, HDMI port (*High Definition Multimedia Interface*), as well as better audio quality or lower consumption, which allows to expand its different configurations and, therefore, functionalities.

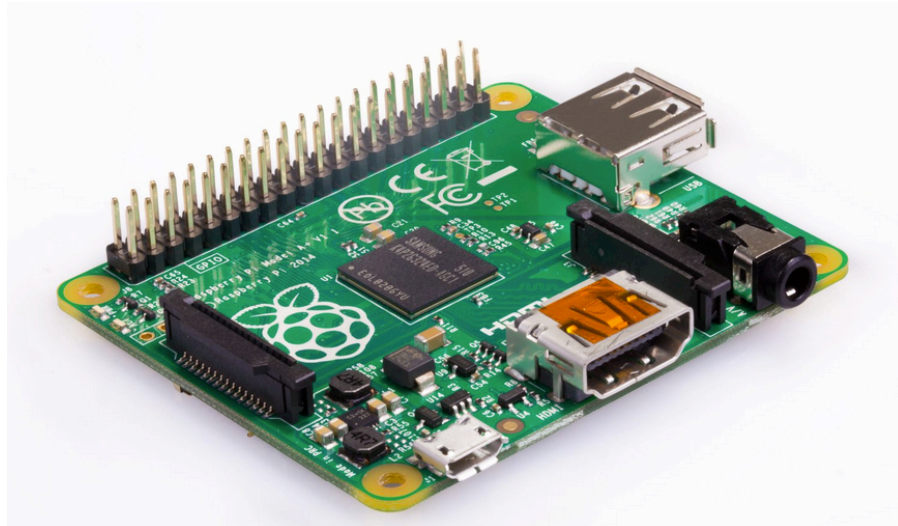


Figure 2-1 Raspberry Pi Model A+ board

In 2015, the new version of the device is presented, Raspberry Pi 2 Model B. In this version, there are only two notable differences regarding the previous ones, which is the increase of one GB (*Giga Byte*) in the RAM memory and a better microprocessor integrated with 900 MHz. In addition to the improvement of other features, as well as the extension of the number of G.P.I.O pins. (*General Purpose Input/Output*)

The great demand in the market for these devices, especially for the academic field, it is necessary to improve this last version in a remarkable way to integrate it better to the unstoppable advance of the computers and related technologies. In

this way in 2016, the Raspberry Pi foundation puts on the market its new device, Raspberry Pi 3 Model B.

### Raspberry Pi 3 Model B

This device will mean a revolution in the market of small computers, since it incorporates wireless technologies BCM43438 Wireless L.A.N (*Local Area network*) in an integrated way in addition to B.L.E (Bluetooth Low Energy), This is a great step forward, mainly due to the fact that the previous versions, which already coexisted with these technologies, sometimes required their use for the development of different projects, which meant the need to integrate additional modules, mostly BLE, making work more complicated with these devices, or further limiting their capabilities. [7]



Figure 2-2. Raspberry Pi 3 B board

In general, the new Raspberry model improves all specifications, with respect to the previous model, facilitating the use of the device by the average user, and qualitatively improving its use by more advanced users. This is the case of the improvement in the C.P.U (*Computer processing Unit*) or G.P.U (*Graphics processing Unit*). It should be remembered that the ultimate goal of Raspberry Pi is to bring the programming and management of computers to the widest possible range of users, and as a consequence make devices increasingly adapted to the

needs that these users demand. That is why, in this model, the integration of technologies best known to users is prioritized (as in the case of Bluetooth and Wireless technology, mentioned above, among others) in such a way that it becomes closer and consequently more easy to use for the consumer. In addition, it is intended to increase the power of use of the already integrated technologies, in order to improve the capabilities of the computer, and also expanding its possibilities of use.

Among the new specifications, the following stand out:

Features	Description
C.P.U.	4 cores 1.2GHz 64bit
G.P.U.	<i>Broadcom BCM2837</i>
Wireless	Chip BCM43438 Wireless L.A.N. B.L.E.
Ports	<ul style="list-style-type: none"> <li>- 4 ports USB 2.0.</li> <li>- 4 Pole stereo output and Video composition port.</li> <li>- C.S.I port (Camera Serial Interface) to connect the Raspberry Pi camera.</li> <li>- D.S.I. port (Display Serial Interface) to connect to Raspberry Pi touch screen.</li> <li>- micro SD port (Secure Digital) to introduce the external memory card where it will take place, both the storage of information as well as the load of the operating system.</li> </ul>
Input/Output	40-extended GPIO input-output pins
Power Source	Micro USB power input up to 2.5 A

Table 2-1 Improvements of Raspberry Pi 3

### ***G.P.I.O.***

Within the great number of capacities that the Raspberry possesses, special mention must be made of the enormous resource represented by the G.P.I.O pins and the possibility of using these pins as a way to interact directly with the computer

system. It is this capacity, which mainly differentiates Raspberry from other computers, such as those of basic use, whether they are laptops or desktop computers. Thanks to G.P.I.O. The system can be used in electronic projects, in a more direct way, since these general purpose pins are directly connected on the printed circuit board, and in turn the main system, allowing an instant connection between the input or output that is in any of those pins and the device. [8]

Next in Figure 2-3, you can see how the forty pins that the Raspberry Pi 3 has are positioned, as well as the function that each of them can play.

It is important to add that not all the pins can be used as input or output, since, as can be seen, some are reserved, and others have the function of ground or voltage source, which in this case have a value that it can be either 5 or 3.3 V.

These pins can be configured in different ways, which gives rise to a wide range of possibilities, through which the system can control various electronic components and devices, such as LEDs (Light-Emitting Diode), motors, loudspeakers, whose configuration it would be output or, on the other hand, read the signal coming from buttons, switches, or different sensors such as brightness, or temperature.

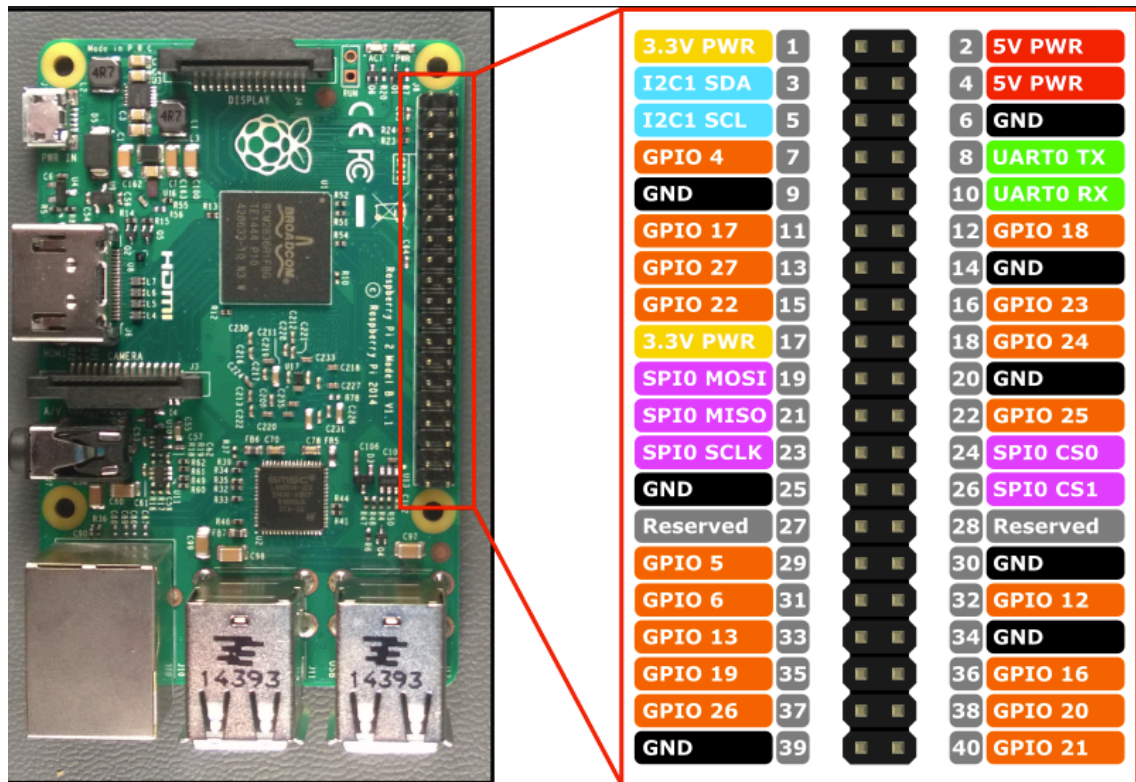


Figure 2-3. G.P.I.O pins map. 1

<sup>1</sup> *Raspberry and GPIO* <https://www.prometec.net/rpi-gpio/>

In addition, as can be seen in Figure 2 3, there are pins, specifically 3 and 5, with which it is possible to implement configurations for serial bus, as in the case of I2C (Inter-Integrated Circuit) or for a communication synchronous as is the case of SPI (Serial Peripheral Interface) in pins 19, 21, 23, 24 and 26. Something that is undoubtedly very useful for the control of different devices such as servos, or the reading of signals periodically.

At this point it should be remembered that there is no direct connection between the different GPIOs and the different analog sensors, which makes it mandatory to use analog-digital converters, which allow GPIOs to read the signals they emit. That is why the I2C and SPI pins are especially relevant when dealing with issues related to analog signals, since analogue digital converters will connect to these pins so that, through them, the system will be able to read those signals, and in this way, the user has the ability to work with them.

Once we see the different pins that make up the system, it is necessary to configure these pins, in order to read or send signals from the Raspberry Pi device. Which is ultimately the ultimate goal of the GPIO. In general there are several methods to configure the different pins of the board.

The most direct way is through the terminal. Once the system is started, either from the graphical environment, by accessing the command line, or by starting from the command terminal itself, we can know the status of all the pins by writing:

```
gpio readall
```

In this way an output is obtained in the same terminal where it shows the status of each pin, if they are configured as input or output, and in the voltage level, 1 (high) or 0 (low) in which said GPIO are as can be seen in Figure 2 4. It is important to remember that the pin number on the plate does not correspond with the name of the GPIO, therefore, it is not the same to say GPIO 2, which is on pin number 13, which pin 2 which does not correspond to a GPIO input output but to a 5V source. From this same environment it is possible to configure the different pins, as input or output, and at high or low level.

```

pi@raspberrypi:~ $ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2 | 8 | 3.3v | | | 1 | 2 | | | 5v | | |
| 3 | 9 | SDA.1 | IN | 1 | 3 | 4 | | | 5V | | |
| 4 | 7 | SCL.1 | IN | 1 | 5 | 6 | | | 0v | | |
| 4 | 7 | GPIO.7 | IN | 1 | 7 | 8 | 1 | ALT5 | TxD | 15 | 14 |
| | | 0v | | | 9 | 10 | 1 | ALT5 | RxD | 16 | 15 |
| 17 | 0 | GPIO.0 | IN | 0 | 11 | 12 | 0 | IN | GPIO.1 | 1 | 18 |
| 27 | 2 | GPIO.2 | IN | 0 | 13 | 14 | | | 0v | | |
| 22 | 3 | GPIO.3 | IN | 1 | 15 | 16 | 1 | IN | GPIO.4 | 4 | 23 |
| | | 3.3v | | | 17 | 18 | 0 | IN | GPIO.5 | 5 | 24 |
| 10 | 12 | MOSI | IN | 1 | 19 | 20 | | | 0v | | |
| 9 | 13 | MISO | IN | 1 | 21 | 22 | 0 | IN | GPIO.6 | 6 | 25 |
| 11 | 14 | SCLK | IN | 0 | 23 | 24 | 1 | IN | CE0 | 10 | 8 |
| | | 0v | | | 25 | 26 | 1 | IN | CE1 | 11 | 7 |
| 0 | 30 | SDA.0 | IN | 1 | 27 | 28 | 1 | IN | SCL.0 | 31 | 1 |
| 5 | 21 | GPIO.21 | IN | 1 | 29 | 30 | | | 0v | | |
| 6 | 22 | GPIO.22 | IN | 1 | 31 | 32 | 0 | IN | GPIO.26 | 26 | 12 |
| 13 | 23 | GPIO.23 | IN | 0 | 33 | 34 | | | 0v | | |
| 19 | 24 | GPIO.24 | IN | 0 | 35 | 36 | 0 | IN | GPIO.27 | 27 | 16 |
| 26 | 25 | GPIO.25 | IN | 0 | 37 | 38 | 1 | IN | GPIO.28 | 28 | 20 |
| | | 0v | | | 39 | 40 | 1 | IN | GPIO.29 | 29 | 21 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
pi@raspberrypi:~ $ █

```

Figure 2-4 Status of GPIO in Raspberry Pi 3

On the other side, there is a great variety of programming languages that have specific libraries to configure said G.P.I.O. in the same way as from the terminal, at the same time being able to develop a series of functions according to the interaction of different external elements with the device itself. That is, through the code it is possible to execute actions on elements with push buttons, servos, or other types of devices connected to the pins.

Among the most frequent languages Python stands out as the most used among users, although the same functionalities can be developed from other languages, since Raspberry has a great flexibility in terms of programming in different languages. Among the most used, are C or C ++, Java or Perl, although the list is much more extensive. Later on, both the motives and the language on which this project bases its development are extended.

## Raspbian

In general, Raspberry Pi is a system in which practically everything is open and configurable according to the different interests of the user, and therefore its



operating system could not be less. It is not mandatory to use a specific operating system, as long as the device is capable of supporting it, most users use Raspbian as a reference operating system for this type of device. Not for a specific reason but for several and all of them very important.

First of all, it is a free operating system that is accessible to any user, so that in a very short time, anyone with an internet connection can access it by installing it incredibly quickly in the external SD memory that will connect to the system. Do not forget this little detail that offers the possibility of having several SD cards, for example, with different operating systems, or different configurations, and change from one to another simply by changing the card.

On the other side, this operating system is specifically designed for Raspberry which makes its use on it is necessarily optimal. It is also based on the well-known distribution of Linux Debian, which allows, like any Linux system, a great capacity to configure almost everything within it.

This operating system contains more than 35,000 packages and many software structures included so that their installation is simple and fast, but that, if they are not necessary, they can remain compressed and thus not spend system resources, something that optimizes the use of this and therefore, it improves the quality of work offered to the user.

Finally, it is a system in continuous development since its creation in 2012, which means that it can be updated almost monthly, in a very simple way, and obtain appreciable improvements, either in its version with user interface, or in the single terminal command. In each update, the aim is to improve internal functionalities, such as improving the performance or updating drivers of the different interfaces, as well as faults that have been detected and reported. [9]

### **2.1.2 Open Frameworks**

The C ++ language is a good candidate for this task, since, in addition to being widely known by students of Telecommunications Engineering, it is also one of the most widely used languages in other areas of technology. Within the very large number of variants in the world of this language, there is one that specifically meets all the above mentioned characteristics quite well, this is *openFrameworks*.

Basically, it is a tool, or rather, a set of software tools based on open source C ++. It is a software that, like the philosophy of Raspberry offers the possibility to its users to participate in its continuous development, as they use it to develop their own projects. It is what is known as DIWO (Do It With Others). OpenFrameworks is distributed under a license from MIT (Massachusetts Institute of Technology), which allows any user to use it publicly or privately, with or without profit, maintaining the possibility of being open source or even without it, and, neither what to say, offering its users the possibility of contributing to development, not necessarily.

The main idea of this tool is to make everything much easier when it comes to obtaining results based on C ++ code, which is achieved by decreasing, to the greatest extent possible, the development necessary to achieve the objectives that each user proposes, so that the degree of simplicity increases. In addition, it is intended to make this an intuitive tool, so that the learning of a function, or the configuration of a function to launch it to execution, is applicable to the rest of functions, so that the user can quickly learn the management of the environment. This favors its use in academic environments, since students of programming languages can quickly reach the understanding, and thus be able to see quickly, and in graphical environment, the result of executing some lines of code that from other environments is not recommended, given the level at which students normally meet at the beginning. That is why the main theme throughout the development of this tool is none other than: "Students first". [10]

For all this, it has a large number of libraries and classes already developed and ready for use, so that when you want to implement certain functions such as a window with an image or video, execute a task from a click of the mouse, or get a graphical interface that is able to interact with the user, the developer only has to add those libraries related to these functionalities, and use the different classes they have. Classes that are already completely ready for use, of course, of a previous self-learning by the programmer about the use of these classes. For this it is highly recommended to use your website, where in the documentation section

[11] there is a very broad description of each and every one of the predefined classes within openFrameworks.

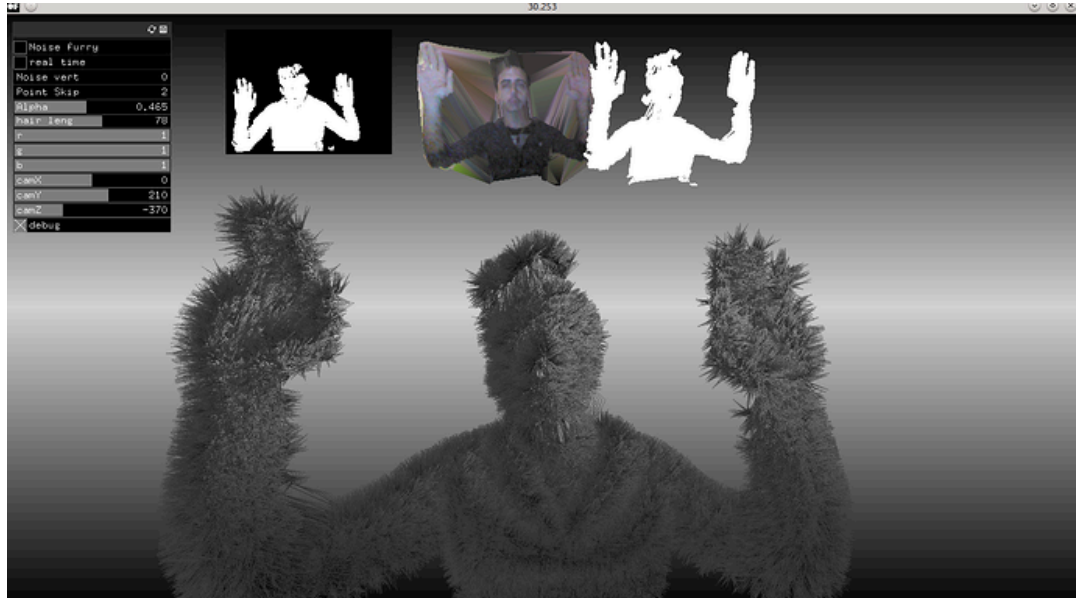


Figure 2-5 Example of window in openFrameworks about a drawing project 2

In this sense, this tool offers several libraries according to the needs that its users usually have. In general, they can be distributed in:

- OpenGL, GLEW, GLUT y libtess2 ara graphics.
- FreeType for fonts.
- Quicktime, GStreamer y videoinput for recording and videos.
- OpenCV for computer vision
- Assimp for 3D modeling
- rtAudio, PortAudio, OpenAL y Kiss FFT for audio input and output and analysis
- Poco as a miscellany of utilities.

### Add-ons

Some of these libraries are grouped in what is known as add-on. The add-ons are units of code that can be added or not to the main algorithm to increase the project's capabilities. The fact that this possibility of extending the main function is offered,

<sup>2</sup> *Small addons Furry Mesh project*: <https://forum.openframeworks.cc/t/small-addons-furry-mesh/18931>

does not only imply the advantage that the code that is being developed can be expanded quickly, only by adding the corresponding add-on and using it according to the need relevant, but also if you do not want to add, this does not have a weight in terms of code that has to be compiled, which is appreciated, when the code to compile is large. In the particular case of openFrameworks, these add-ons are grouped according to the different elements that they want to use in an additional way, as for example in the case of user interfaces, more video tools, or use of additional hardware, such as the case of this project, in which additional electronic elements are used.

For the particular case of the use of these tools on Raspberry, as explained in the previous section, the use of G.P.I.O. must be remembered. It is for this reason that a connection between the openFrameworks tools and these pins of the board is essential.

As a primary idea, the development of the code structures necessary to connect any main class of the openFrameworks environment with these inputs and / or outputs can be assumed manually, in order to have control over their configuration, as well as their reading or writing. This idea, although viable in any case, is unnecessary, since there is an accessible add-on within the same page of this software, in which the GPIO class is configured and therefore, the connection through software, with the physical pins . Something that, as on many other occasions, makes working with this tool faster and easier.

Thanks to this wide variety of utilities, add that these tools are intended to generate robust codes, which are able to maintain their operation despite adding new features, or eliminate part of them. This is achieved by having a wide distribution of functions within the main class, like offering the possibility of programming each of them independently and in this way, ensuring that a failure in any of them does not interfere in the development normal of the project, unless it is clear that they must be related to each other out of necessity.

Finally, it is important to talk about the great compatibility of openFrameworks, with a great variety of operating systems and programming environments.

It would not make sense to create such a tool and limit its use to a couple of platforms, no matter how used, since it would not fulfill the main objective described at the beginning of this section, which is to allow a multitude of users to use it. To be able to collaborate in the development of certain projects of this tool or of the tool itself. This would not be entirely viable if the user were not given the facility to work in the I.D.E. (Integrated Development Environment) that most interests it, as in the operating system you use regularly.

### **2.1.3 Analog / Digital Conversion**

In the previous point, the essential use of analog-digital converters is discussed to be able to read external signals, from other devices for instance, and also to get that already mentioned information, within the Raspberry Pi device. That is why additional hardware is needed, in order to achieve this transformation.

In the world of electronics there are countless analog-digital converters, so many that if you have to choose one in particular can involve a long and in-depth study of all the families of them, until you reach the one that best suits the characteristics that are searched. In this case, it must be taken into account, in addition to complying with the main objective, which would be to immediately transform an analog reading from an electronic element such as a potentiometer, into a digital format that can be used for interests of this project, the essential fact of compatibility with the various components mentioned above, such as openFrameworks, and Raspberry PI.

Following this last premise, the possibilities, fortunately, are greatly reduced, since the connection with Raspberry has to occur through GPIO and, therefore, these converters must be compatible either with the I2c pins of the device, or with the SPI connections.

Based on the above, there are two options:

#### **Option I<sup>2</sup>c.**

In summary, I2c is known as a standard data bus developed in 1982 by Philips Semiconductors. Although originally, it was created to interconnect several chips in televisions in a simple way, later, it is verified that it can be used with the purpose

of interconnecting integrated circuits or ICs. Although it is designed as a master-slave bus, this standard can act as an interconnection of multiple ICs in which each of them acts as a teacher, that is, each one can initiate the transfer of information, as long as it is synchronized, either automatically, or guided by specifications. Following which, it is allowed that each teacher can take turns in the use of this bus as a transmitter of information.

As we can see in Figure 2-6, the connection of elements to this bus must be accompanied by a connection that arbitrates the use of the same, for which they connect two clocks of the elements connected to the serial line or SCL (Serial Clock Line), which will be in charge of deciding who should transmit, by the SDA line (Serial Data Line), also bidirectional, in the clear case it is, of being a multi-master structure.

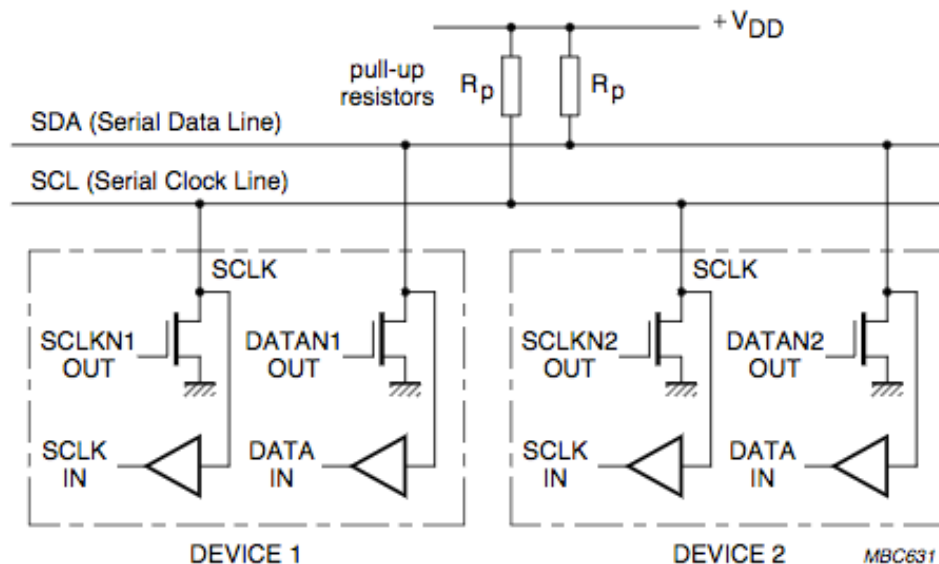


Figure 2-6 Connecting devices to the I2C bus [12]

As for its transmission speed, it will be determined by the way in which the bus is configured, or that required by the specifications. For example, in the case that the devices connected to said standard are analog / digital converters, a minimum frequency is required to ensure correct operation.

This means that, as shown in Table 2-2, there are several types of I2C connections attending to the needs that are found. And it will be the user's choice to take the configuration that best suits their interests.

Mode	Maximum Bit rate	Direction
Standard Mode (S.M.)	0,1Mbit/s	Bidirectional
Fast mode (F.M.)	0,4Mbit/s	Bidirectional
Fast Mode Plus (F.M.+)	1,0Mbit/s	Bidirectional
High Speed Mode (H.S.-mode)	3,4Mbit/s	Bidirectional
Ultra Fast Mode (U.F.M.)	5,0Mbit/s	Unidirectional

Table 2-2 I2c transmission modes [13]

In this area, and focusing on analog-digital converters, there is an element that meets the characteristics sought, taking into account, as mentioned in other sections, its compatibility with previous tools and devices. This is the case of the digital analog converter MM112 of the Velleman family.

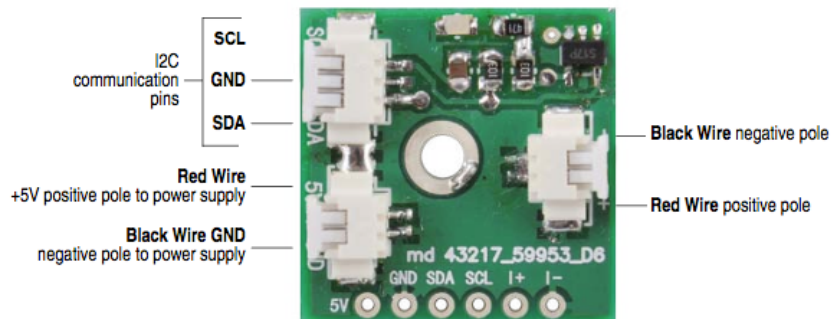


Figure 2-7 View of the MM112 board [14]

### Option SPI.

Another of the variants that the system offers us, would be the connection through the SPI pins. Something very similar to that commented previously, since it is also a serial connection. This standard of communications comes from the hand of

Motorola, and it is a synchronous protocol used to connect peripheral devices of medium and high speed. Like the previous model, SPI follows a slave master configuration, although in this case, it is the master device, which selects the slave and from that moment begins the process of transmission and reception by both devices, following a communication Full -Duplex, that is, you have a transmission line and a reception line.

This protocol makes its connections through four signals, which are:

- SCLK (Clock): It is the clock signal that marks the synchronization, with each pulse a bit of information is read or sent.
- MOSI (Master Output Slave Input): Data output of the master element and data entry of the slave element.
- MISO (Master Input Slave Output): Data output of the slave element and data entry of the master element .
- SS/Select: Signal of selection of the slave that is going to be the one that transmits or receives in that moment, can also be the one that selects the master device. In the case of the Raspberry pins, this signal is divided into the two indicated below, which determines that this device will limit the number of slave devices to two:
  - CS0 (*Chip selected 0*)
  - CS1 (*Chip Selected 1*)

With Figure 2-8 it is possible to get an idea of how these communication signals work, in a master configuration and a single slave.

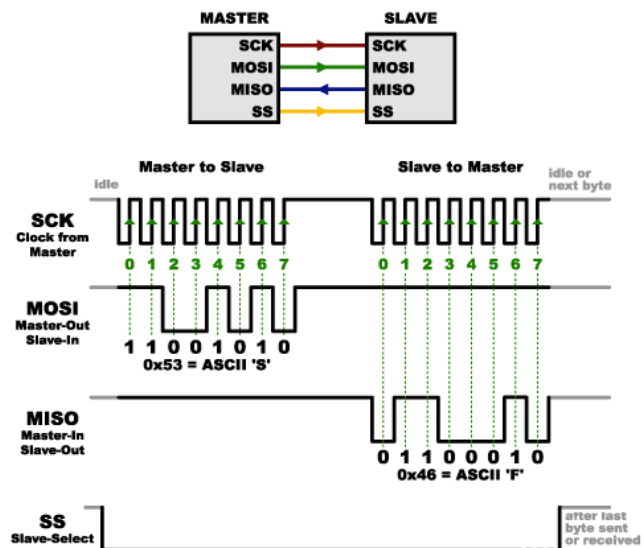


Figure 2-8 Schematic and signaling of a SPI connection [16]



As in I2c, in SPI there are also different transmission modes, depending on the states of the clock signal for polarity (CPOL) and phase (CPHA), both with two possible states, therefore, the possible combinations of them will give rise to four modes as can be seen in Table 2-3. In this case, the mode does not depend on the transmission frequency, as in the previous standard, being much higher than that allowed in SPI,[17]

SPI Operation mode		Description
Mode 0	CPOL =0	Active with rising edge. Logical state of the low clock. The information is sent when the transition is from low to high.
	CPHA =0	
Mode 1	CPOL =0	Active with falling edge. Logical state of the low clock. The information is sent when the transition is from high to low.
	CPHA =1	
Mode 2	CPOL =1	Active with rising edge. Logical state of the low clock. The information is sent when the transition is from low to high.
	CPHA =0	
Mode 3	CPOL =1	Active with falling edge. Logical state of the low clock. The information is sent when the transition is from high to low.
	CPHA =1	

Table 2-3 SPI operating modes

Once this standard has been analyzed, the dilemma of using an appropriate converter, based on it and which in turn is compatible with the openFrameworks and Raspberry tools, is reached. In this case, the most commonly used option is the Digital Analog Converter MCP3008 [18].

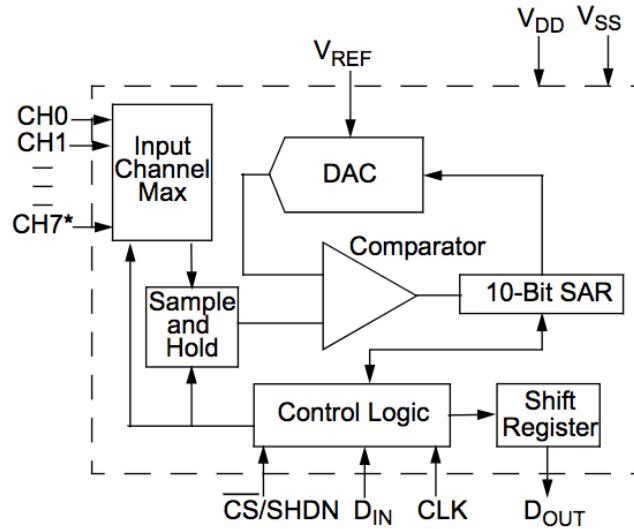


Figure 2-9 Functional block diagram for MSP3008 [18, p. 7]

This IC contains a digital analog converter by successive 8-bit encapsulated approximations, with eight pins configured as analog voltage inputs. The power range is from 2.7 to 5.5V which is perfectly compatible with the levels offered by the Raspberry, which can be either 3.3V or 5V. This model is taken as there is the 6-bit version of this family, with this would be two pins without functionality, since the same way as in the 8-bit, the eight side on the right side would be configured for connections SPI, therefore, taking into account the possible applications of the model, and the same price and specifications of both, will take the 8-bit for the development.

In addition, it has a simple serial interface, which allows connecting with other devices through the S.P.I. protocol, having the corresponding pins associated with this type of configuration, in Figure 2-10 it can be seen how the different pins of the package are distributed.

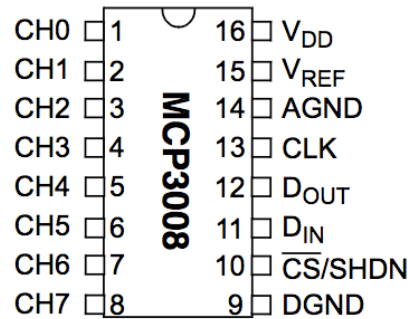


Figure 2-10 Diagram of MCP3008[18]

By specifying only one element from which to obtain analog information, as is the case of the potentiometer, only one of the inputs, such as CH0, could be used, and as for the configuration, it could be a Mode 0, since it is very usual work with rising edge.

However, being an encapsulated chip, its connection to the computer itself would require additional added circuitry, since the connection between Raspberry terminals and those of the encapsulation must be supported by a perfectly connected board and cabling, in order to ensure an adequate connection between pins.

In addition, the SPI connection must be enabled within the computer system in the same way as defined in the previous section for I2c, but instead of selecting that option in the menu, the SPI option of the same Menu is selected.

As far as this connection is enabled, they are assigned to the Raspberry GPIOs, defined above, for SPI connections, taking into account that the Master would be the computer system, and the slave the MCP3008.

Finally, it is important to add that, in this type of projects, a previous study of the variants that can be used for their development, contributes a great amount of knowledge, that, in spite of discarding one or several of the options that are shuffled, It allows valuing the advantages offered by the chosen line, and further enhance its virtues. In addition, it is important to have a broad view of the market, as this will help future projects have less work prior to choosing the technology.

### **3 REQUIREMENTS**

#### **3.1 What is really a requirement**

One of the main problems we find when facing the creation of a new system is to answer the question: What do you want to build? Even though its apparent answer seems simple, nothing could be farther from the truth.

##### **3.1.1 Definition and attributes of requirements**

Regarding the afore mentioned question, there is a plethora of answers, within those would be defined that we want to develop from a high level point of view, without considering any sort of viability or versatility of what is required, until a vast and detailed theory containing the why and the how of each and every concept that we intend to accomplish. This last perspective is known as Requisite Engineering.

Therefore, we can say that the answer to to this question should be done following a series of guidelines and parameters to be considered, that together, make what is desired to the system. These guidelines to be followed are what is defined as requisites.

“The most difficult past when building a software system is deciding what to build. No part of this task invalidates the system in such a great way, if it is done incorrectly. Nothing is harder than fixing it later.” (Fred Brooks)

A requisite, is nothing more than a quality or characteristic that that is required from the user of the system that we intend to build. However, this user, that sometimes can be the project creator, initially doesn't have a clear idea of how there requisites may or may not be accomplished, or simply doesn't have the technology or software knowledge used in the system and therefore cannot define such requisites in an adequate fashion. In this case, a certain engineering is necessary at the time of managing such requirements. It is something similar in great fashion to the commonly known Software Engineering.

This means that, a good requisite scheme helps that the project always aims at the right direction, knowing what the user, or in many cases, the client really wants or expects, and therefore will successfully respond to their needs. It is, without question, a very robust way of creating projects, starting from a stable foundation and, without a doubt will act as a guide throughout the whole process. [20]

To determine said requisites, there are several procedures, through which we can reach a final result that fits perfectly to the user's defined needs. One of these procedures would be [21]:

- Collection of Requisites . Here, we write down all the information that the client or user brings. This information can be of a very diverse nature, and, as previously mentioned, the client sometimes may not cater to the project's technical needs, as much as his personal needs towards the system.
- Definition of Requisites. Once obtained a full list of all the necessary system's requirements, the next step would be to define with exactitude, at a technical level, the system's requirements. This task, as all the ones before has several available options, but usually can be defined using natural language, or on the other hand, a more formal language, from use cases, although this last one is less used due to its ambiguity. On the other hand, following several scenarios, it can also vary depending on the author, but in any case, a good analysis of the different possible scenarios can presuppose a great benefit regarding the functional needs of the system.

Lastly, one can follow patterns or pre-established sheets, where data is more structured and are described in a natural language. These would be tables with different fields where the user keeps adding information. This eliminates the possible ambiguities since the information is more structured, but we must not exceed with the level of detail of such tables, because at the time of filling in the information, we want the task to be as painless as possible to the end-user.

- Requisite Validation. In this last step, we want to demonstrate that the previously defined requisites, comply with the user or client's needs . This step is crucial to be executed, considering that this is initial reason why the user has trusted the system developer to comply to his needs.

Of all different available methods for the validation of user's requirements, most of them involve checking directly with the user, and in this way, detect possible errors, where in a lower level point of view, would not be possible to detect.

Lastly, it is critical for a requisite list, that all of them are *verifiable*, so that whether a person, client or user, or a machine, for example different testing systems, can test and check each and every system requirement. Furthermore, there requisites should be verifiable, there should be a way to check if the requisite is fulfilled. For example, a requisite would not be valid if it used infinite measures, where we have no way of checking. Therefore, it is crucial for this characteristic, that we comply with an additional one, which is, that this list is *unambiguous*, and that means, that each requisite has one single unique interpretation.

### 3.1.2 Requirements groups: functionals and no functionals.

In requisite Engineering, there is a dimension called Characteristic that defines [22] It focuses on a way of classifying the requisites considering the distinct natures of the desired system's specifications and defined in certain requisites. Taking this in consideration, we can split it in two main currents: functional and non-functional.

- **Functionals.** They define the functions that the system should do, this is, that the system should execute. In the existing, different systems, its different parts execute a specific function, and it is the group of all functions that originates the final result which is the system itself. The functional requirements are those that specify which tasks should comply each of its elements, that interact, direct or indirectly, with the user, as are all the system's functionalities, at a global level.
- **Non Functionals.** Now that we have defined the functional requirements, the developer may be very tempted to group all other functions as *undefined*, since they would be the rest of the requirements that are not part of the previous ones. This action would lead to a non-homogeneous set, within which different types of possibilities would fit. Which is not entirely uncertain. Although they do not stop being important, in fact, they can be very useful and make big differences between projects that, in terms of technology, have a huge similarity in terms of functional requirements, but which, nevertheless, at the time of compare the non-functional, we can see that they are totally different systems.

Among this type of requirements, We would define for example those related to appearance, comfort in terms of use, ease, how the system works within the required parameters, and also the quality of the different elements with which the user has contact.

### 3.2 System requirements

Id	Name	Functional	Description	Justification	Satisfy by	Verification
1	Image	Yes	The device should capture a moving image	A mirror captures the image in front of it	Webcam/Integrated Camera	We should be able to capture the image
1.1	Image Storage	Yes	The system should be able to store the captured image	To process this image, the image should be able to be stored in the sytem	Built-In processor	The captured image by the camera should be able to send to the built-in processor
1.2	Temporary Storage	No	The system should save the video before it is shown	The saved video should be processed, and this way, insert the delay	Built-in processor	If the shown image includes delay, this implies that it should have been stored previously
1.3	Alignment	No	The device that capures the image should be aligned with the surface where it is shown	When an image is shown in a mirror, the image is aligned	Webcam/Integrated camera and the surface that shows the image	The image shown should match with in the surface where it is shown

1.4	Image Processing	Yes	The system will process the image so that it can execute its functionality	Add a delay and then show the image requires processing	Built-in processor	We can work from the processing system with the image that has been captured by the camera
1.5	Inversion	Yes	The image should show inverted in display surface	A mirror show an inverted image	Screen/Projector	The shown image would be displayed with an horizontal inversion
1.6	Mirror function	No	The system should look, in the best possible way, to a mirror	To fulfill its main purpose and integrate in a natural way, in a dance class	Projecting surface	The system should be integrated as just another mirror in the room
2	Delay	Yes	The shown image should be displayed with a delay of a few seconds	So it does not interrupt the work flow of the final user.	Built-in processor Screen/Projector	The shown image is displayed with a delay
2.1	Variable delay	Yes	Image delay should be variable. Controlled by the user	For a better work flow, the delay should be adjustable, adaptable to each user	User interface	Ability to change the delay seconds



2.2	Maximum Delay	Yes	The delay should have a maximum of 20s	The storage buffer has a finite capacity	Processor	The maximum selectable delay should be 20s
2.3	Minimum delay	Yes	The minimum delay should be 0s	The system should be able to work as a simple mirror	Processor	The minimum selectable delay should be 0s
3	Visualization	No	The system should have certain adjustable visual characteristics	The system should not difficult the user activity due to low output quality	Processor, Projector and Camera	The user should have a fluent feed at all times
3.1	Continuosity	No	The system will show the video in a continuous way, without cuts of any kind	When maintaining work flow, it is very important that the video does not block at any moment	Processor	The system will proect at all times, a moving image, while it is in use.
3.2	Resolution	No	The shown image should have a certain quality	The system should show an image where we can see certain details in positions or movements	Projector	In naked eye, image quality should not be pixelated

4	User interface	No	The system should be accessible by the user in a simple way	For the user interaction with the system, a physical interface must exist	Box	Controls should exist, that allow the change of the main parameters in the system
4.1	Simplicity	No	The interface should be intuitive	This way, we can increase the use of this device to more users	Box	The controls should have the most simple functions
4.2	Speed	No	The parameters control should be fast	Interference with the use of the device should be as least as possible	Box	Each part of the interface should modify in a fast way, the parameters of the system
4.3	System protection	No	The system should be secured in the best possible way	Considering it is a group of electronic components, they should be secured and isolated in the best possible way	Box	All possible electronic components should be kept hidden and inaccessible from the user

Table 3-1 Table of Requirements

### 3.3 Architecture

The system in general would be composed of three parts in particular. The inputs are the system itself, and the outputs, as shown in Figure 3-2.

First, there would be the camera module, considered as one of the entrances, through which you can capture environment images in which the device is located and thus be able to introduce them to the system to which it will be connected by one of its ports. Another entry of the system will be that of which the user is part. This, through its actions on the user interface will be able to control certain parameters at the same time of functionalities, thus modifying the procedure to be followed by the system. These actions can be determined by pressing a button, changing the position of a switch, or varying the position of the delay through the subsystem designated for it.

Secondly, there is the system which will be responsible for processing all the information and / or implementing the actions required by the user, in such a way that the functions for which the system is designed, such as delay variation, video visualization, or simply the performance of the delayed mirror function, showing the image with a delay selected by the user or imposed from the start.

Finally, the output of the set is given by the Display device, which does not have to adjust to a screen, but can be any element capable of connecting to the system, and display an image with the requirements described in the previous section. This display shows the result of the whole process, giving the user the output that it requires from the beginning, so it can be said that it is a closed system by the user, since he has the ability to decide what to see, see it, and change at that moment the way to receive this information.

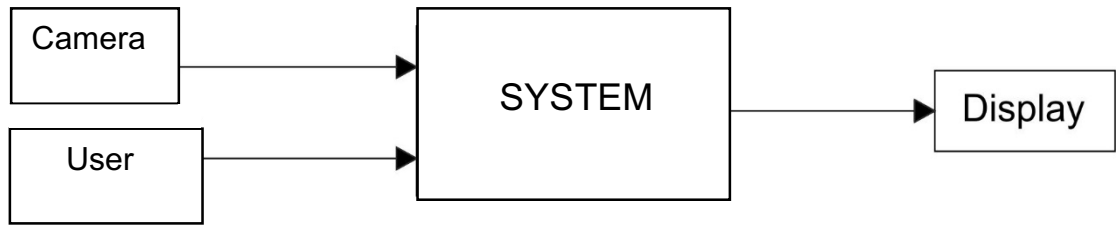


Figure 3-1 Block diagram of the system

## 4 HARDWARE DEVELOPMENT

### 4.1 Physical architecture

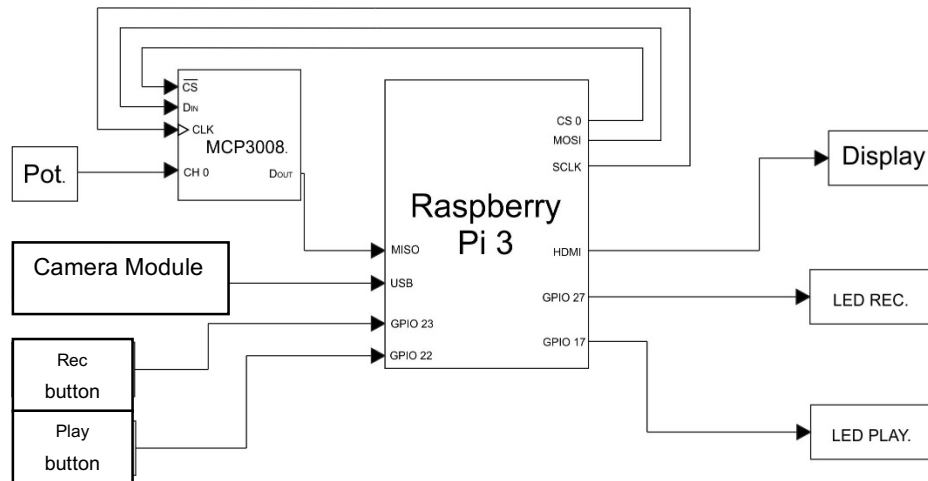
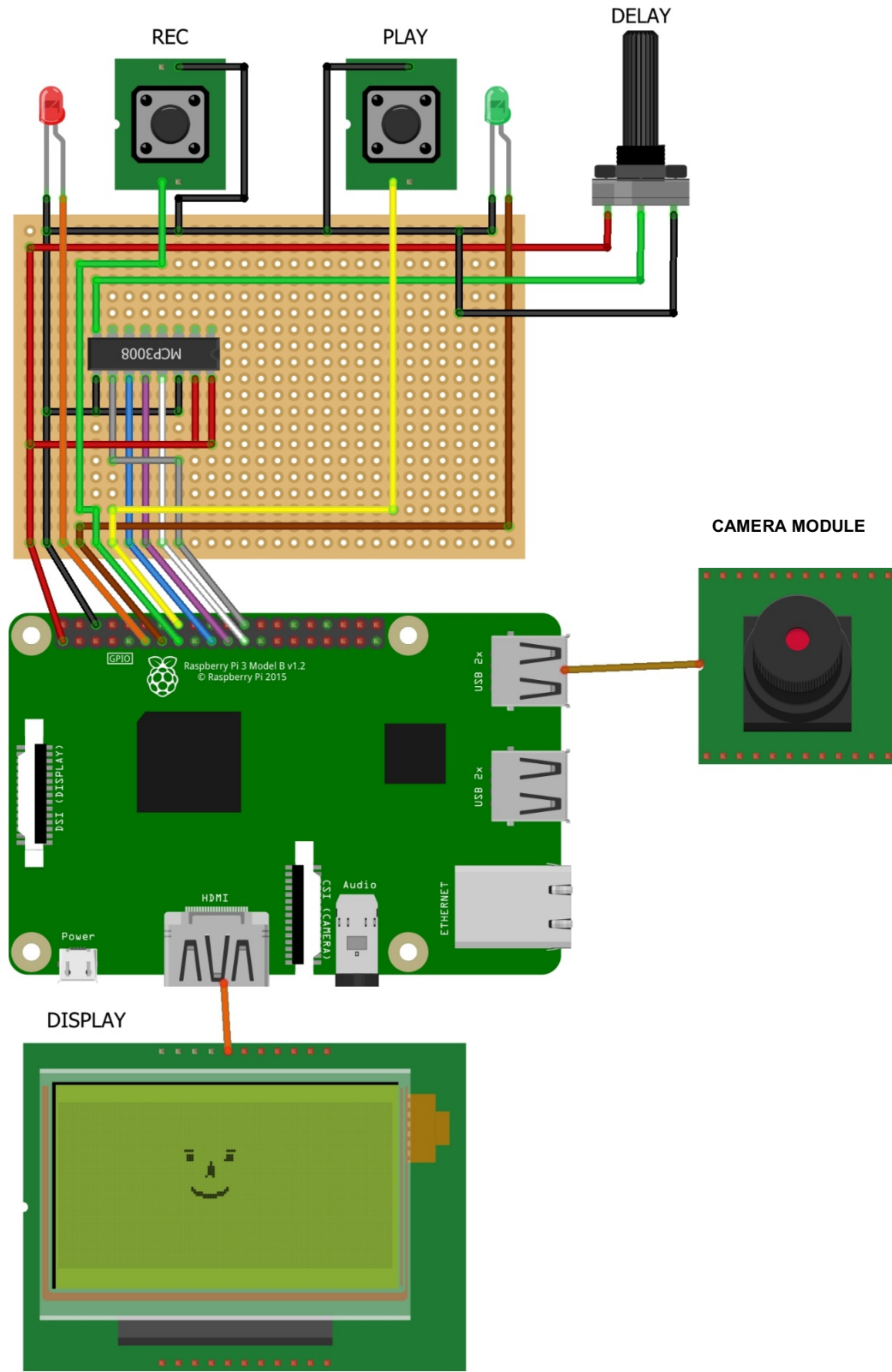


Figure 4-1 Block Diagram of Hardware configuration

In Figure 4-1 the architecture of the entire system can be observed. It is important to note that there are two parts clearly differentiated by the plates on which they are located. On the one hand, the Raspberry Pi device board, which is in the lower part of the set, is connected to the other board only through different wired lines. The video input device, which in this case is a WebCam, would be connected to a USB input port of the previous recessed system. Then, the device that displays the image would be connected to the HDMI output port, as in case of a screen, or a projector either short or long shot.



fritzing

Figure 4-2 System architecture designed on *Fritzing*<sup>3</sup>

<sup>3</sup> Fritzing is an open source software specially designed for electronics circuits <http://fritzing.org/home/>

On the other hand, the circuit board on which the rest of the components are welded would be on top of the model, that is on the previous plate, in order to reduce, as much as possible, the size of the assembly, in Figure 4-2 it is shown in a different plane for a better visualization. In this part both the push buttons and the LEDs are connected, as well as the analog-digital converter. The connection with Raspberry is done as explained above, reserving on this board an opening for each cable. In this way, the potentiometer remains to be connected, which would be through wired tracks from its pins to the different pins of the digital analog converter. Next, the different parts that make up the model are explained in a more detailed way.

#### 4.1.1 Camera subsystem

It is formed by the image capture system, whose input would be the actual image to be captured, which is the input signal of the subsystem, and the Raspberry USB ports, which are responsible for sending the image, already transformed to digital format, to the processor subsystem. This digital image is considered the entry of the subsystem.

It should be remembered that in this model you can connect up to four different devices, as many as the number of ports, although it will only be possible to use one of them, which will be selected automatically. As expected, the port used is completely indifferent, since the system will recognize the device wherever it is connected. This subsystem is responsible for entering the image files to the device.

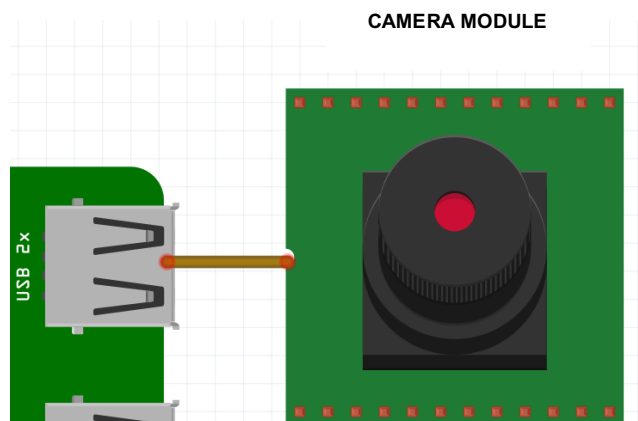


Figure 4-3 Camera subsystem

### 4.1.2 User interface subsystem

This part would be composed of the different interconnected components with which the user must interact, such as the push buttons, which act as ports of entry, like the potentiometer, and the LEDs that would be responsible for sending the output light signal, all of this is shown in Figure 4-4. The latter, although not directly, as in the case of the ports of entry, they would interact with the user of the system, as they provide information on the state in which it is located.

In addition, in turn this subsystem can be divided into two, which includes the elements related to the functionalities of Play and Rec, ie the one that compose LEDs and pushbuttons, and the subsystem that composes the delay control functionality, which would be only the potentiometer.

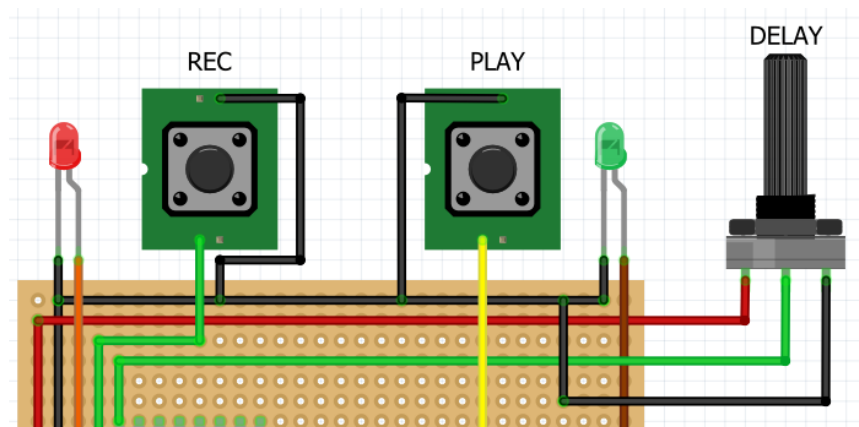


Figure 4-4 User interface subsystem

As additional information, regarding the connections in order to that both pushbuttons and LEDs connected directly to the Raspberry GPIOs. This is possible without the need to add pull-up or pull-down resistance, since these Raspberry pins would include integrated and configured elements. Thus, maintains connected to high level, 3.3V, the input pins while the associated pushbuttons are not pressed, which could automatically connect its voltage to ground.

In the same way with LEDs, their connections do not require additional resistors, since the voltage between their terminals, once a high voltage level is established, up to 3.3V, in the associated output GPIO, the maximum current that can be give

will be around 16mA, which does not exceed the maximum allowed through the LED, which saves the use of additional elements, and ultimately saving space in the implementation of the circuit

### 4.1.3 SPI Subsystem

Formed by the elements and connections that would form the SPI communication of the system. Their inputs are those coming from the output of the voltage divider that forms the potentiometer, the clock signal from the processor system, the MOSI signal from the processor as well as the CS signal, its outputs from the other side are the signal that it would send the processor system transformed in digital format, and the MISO signal. Basically it would be composed of the MCP3008 converter, connected to the Raspberry pins reserved for this type of connections.

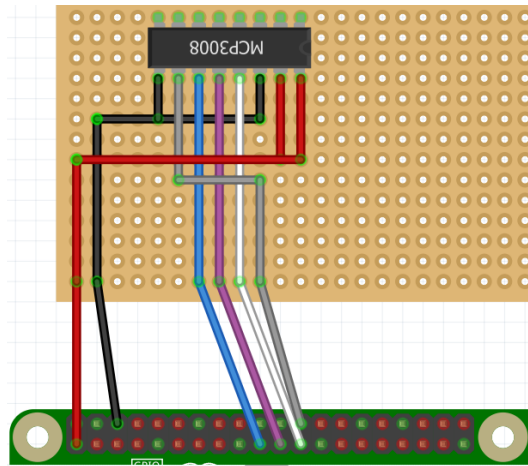


Figure 4-5 SPI Subsystem



#### 4.1.4 Processor subsystem

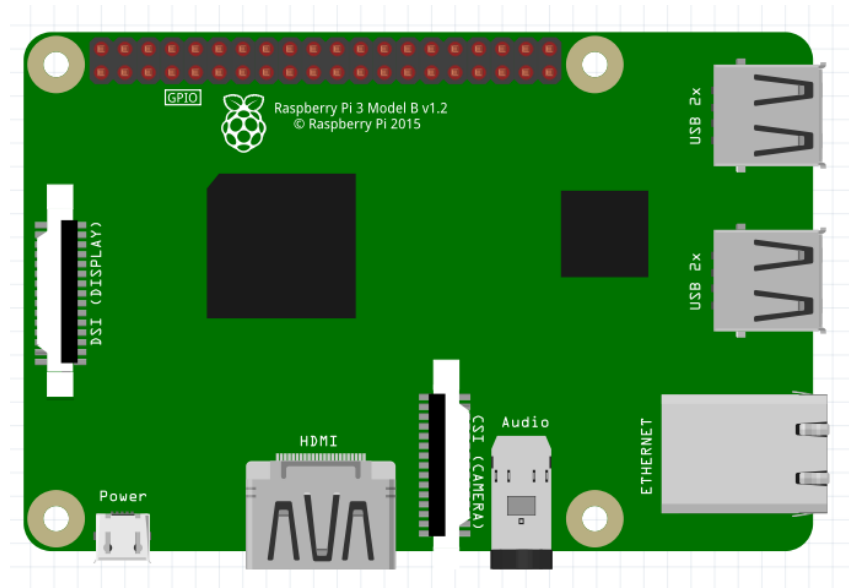


Figure 4-6 Processor subsystem

The processor system would be composed by the Raspberry PI embedded system itself. without the pins, USB or HDMI ports, since these would be part of the rest of subsystems. Its input ports are those that would be connected to the output ports of the other subsystems mentioned above (pushbutton GPIOs, MISO, CS0), as well as the connection system to the USB ports, and their output ports, they would be those that would likewise connect to the inputs of the previous subsystems (MOSI, SCLK, CS0, GPIOs of the LEDs), as well as the connection system to the HDMI port.

The subsystem will be in charge of acting as processor and model memory, interpreting inputs and returning the appropriate outputs based on them.

### 4.1.5 Display output System

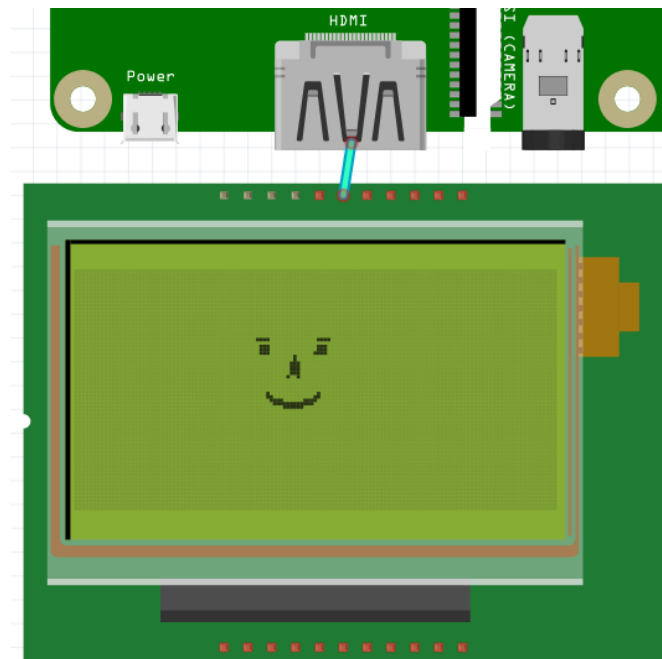


Figure 4-7 Display subsystem

Mainly formed by the Raspberry Pi HDMI output port and the device by which the image is displayed. This will be the part of the model that can be considered as the output of the system, which will be the result of all the processing from the input of the image, through the response to the entries by the user, and followed by the corresponding processing of the information. Once the image is shown by the output device, it can be said that the process has finished.

## 4.2 Implementation

For the implementation of the circuit, different electronic components existing in the market are used. As it is possible to see in the previous section, some of them are quite basic components. This is the case of the pushbuttons or LEDs.

The MCP3008 converter can be considered something more complex indeed. Not only in terms of use, concretely within this kind of development environments, but also in their integration into the circuitry of this prototype.

On the other hand, the processing system, which would correspond to an element such as the Raspberry, could not be considered in any way simple, but quite the opposite, although it is an element that is found quickly in these times, since either through the internet or various specialized stores.

Add also that the price is something symbolic in all of them, except, as is obvious, the Raspberry device, but also, it is still affordable for any user who wants to implement the system.

As for the elements of camera or screen, you can use all those that allow to meet the requirements mentioned in 3.2. Although this fact should not be a problem in terms of obtaining it.

On the other hand, the system needs to be "encapsulated" in some type of structure in order to improve its transport, increase the protection of the different components that make it up, and make it visually more attractive and manageable for the user. That is why, as part of the implementation of this system, a structure designed specifically for him is added to meet the above needs.



Figure 4-8 System implementation from the top of the open user interface

Next, the final implementation of the model is shown in Figure 4-8, in which all the components are interconnected so that you can see how the connections are from the internal level. In addition, you can see both the converter and the back of the potentiometer with the connections established in its pins. This view corresponds to the one that shows the two plates interconnected in the same plane, as would be shown in section 4.1 above. The rear part of the perforated plate in which the elements are located as LEDs and pushbuttons, which can not be seen from this perspective, can be seen in Figure 4-9.

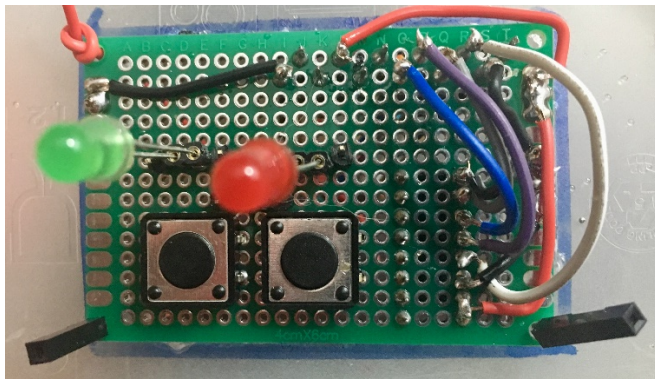


Figure 4-9 Part of subsystem welded on the board

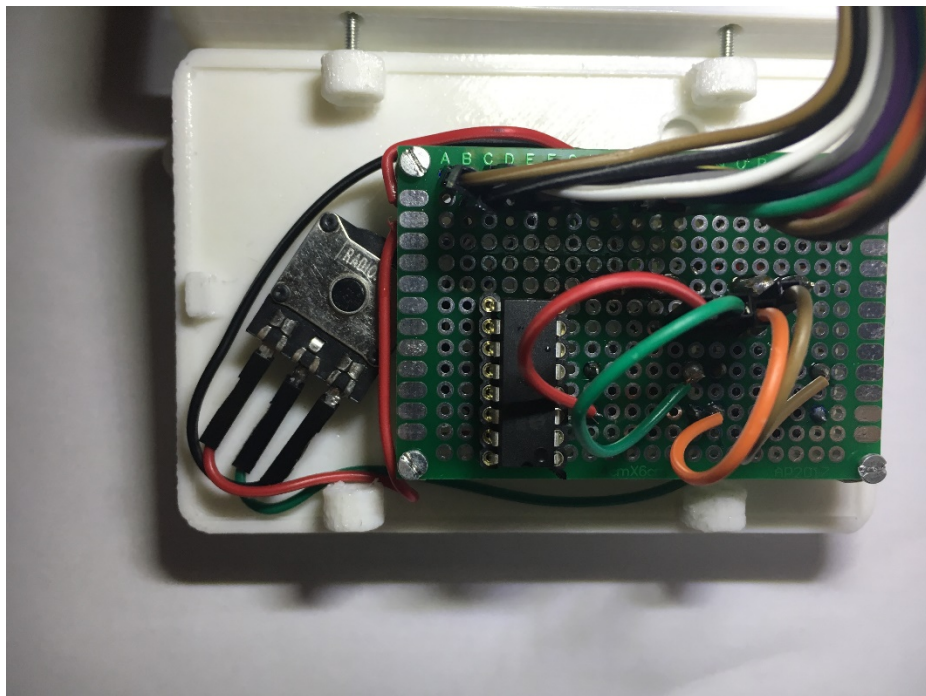


Figure 4-10 Back sight of the system with potentiometer and MCP3008 converter

## 5 SOFTWARE DEVELOPMENT

### 5.1 Introduction

In the first chapter the learning objective was to deepen the work environment related to the C ++ language, very well known at the technological level, and at the same time so useful when working with computer-based systems of small size as is the case of *Raspberry Pi*.

Following this line, it was also intended to use environments and tools, based on this language that would allow the development of different projects on the embedded system, for which *openFrameworks* is chosen, as an *object-oriented* C ++ based work environment, and its large variety of libraries to be able to implement these functionalities in the system, specifically, the use of the G.P.I.O. library, to directly connect the pins of the *Raspberry* board with the developed code, and thereby control, from the code itself, each of them. This allows both to capture the actions required by the user, which will be considered as entry orders, and to show the results of the process by the output peripheral that is used, which will be the output signals.

Therefore, this chapter aims to describe how it is possible to achieve the function of a delayed mirror, implemented on a C ++ code, on a *Raspberry* system, using libraries of the *openFrameworks* tools. All this is done on a specific IDE known as the integrated *Geany Programmer Editor* based on the *Raspberry Pi* device, although, as previously mentioned, it can be any other even a blog of notes and compile the code directly on the terminal command, method which in turn is often used for users of this type of embedded devices. In this way, it is intended that the entire software part of this project be described.

### 5.2 Program Structure

The structure of any program based on *OpenFrameworks* is always the same, since this is one of the main characteristics of this type of tools. As discussed in

previous sections, this implies a great ease both at the time of understanding, and in the development of applications, since all will be based on the same sequence. First, and since every structure in C++ executes the main program or *main()*, within *main.cpp*, the function of *Setup()* in which they are created, variables, classes and related functions that will be used during all the code.

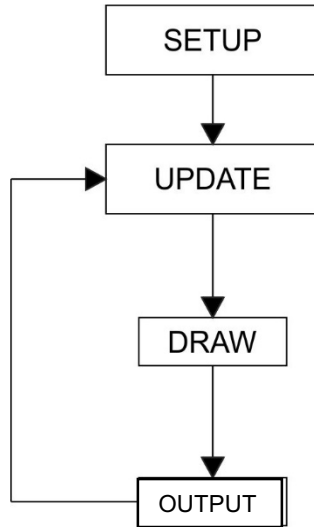


Figure 5-1 Flowchart of an openFrameworks program

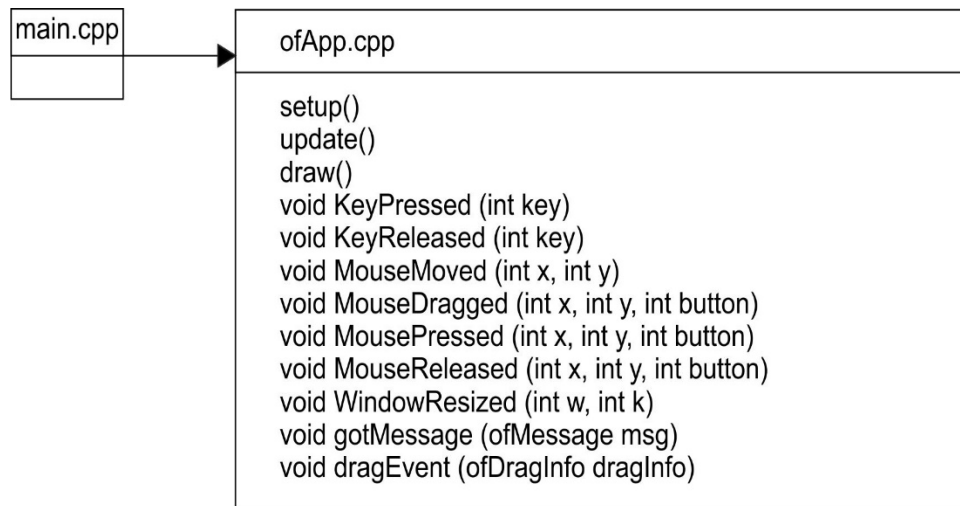


Figure 5-2 openFrameworks generic class diagram

### 5.3 Implementation

The development of this project, at software level, can be divided into three clearly differentiated parts. One of them could be considered as the part of the code where it interacts directly with the GPIO pins of the embedded system, which could be

termed as input functions. On the other hand, the part of internal processing of the image can be considered where it deals with the information collected by the input devices. And finally there would be the part related to the exit of the system, where the result of this whole procedure would be shown.

Figure 5-3 shows the flow diagram of the code developed for the system, showing the different phases of the same. Similarly, Figure 5-4 shows the class diagram specific to the system, including the different libraries and classes used. All of them are developed in a more precise way in the following sections with special emphasis on the most relevant classes and libraries for this development.

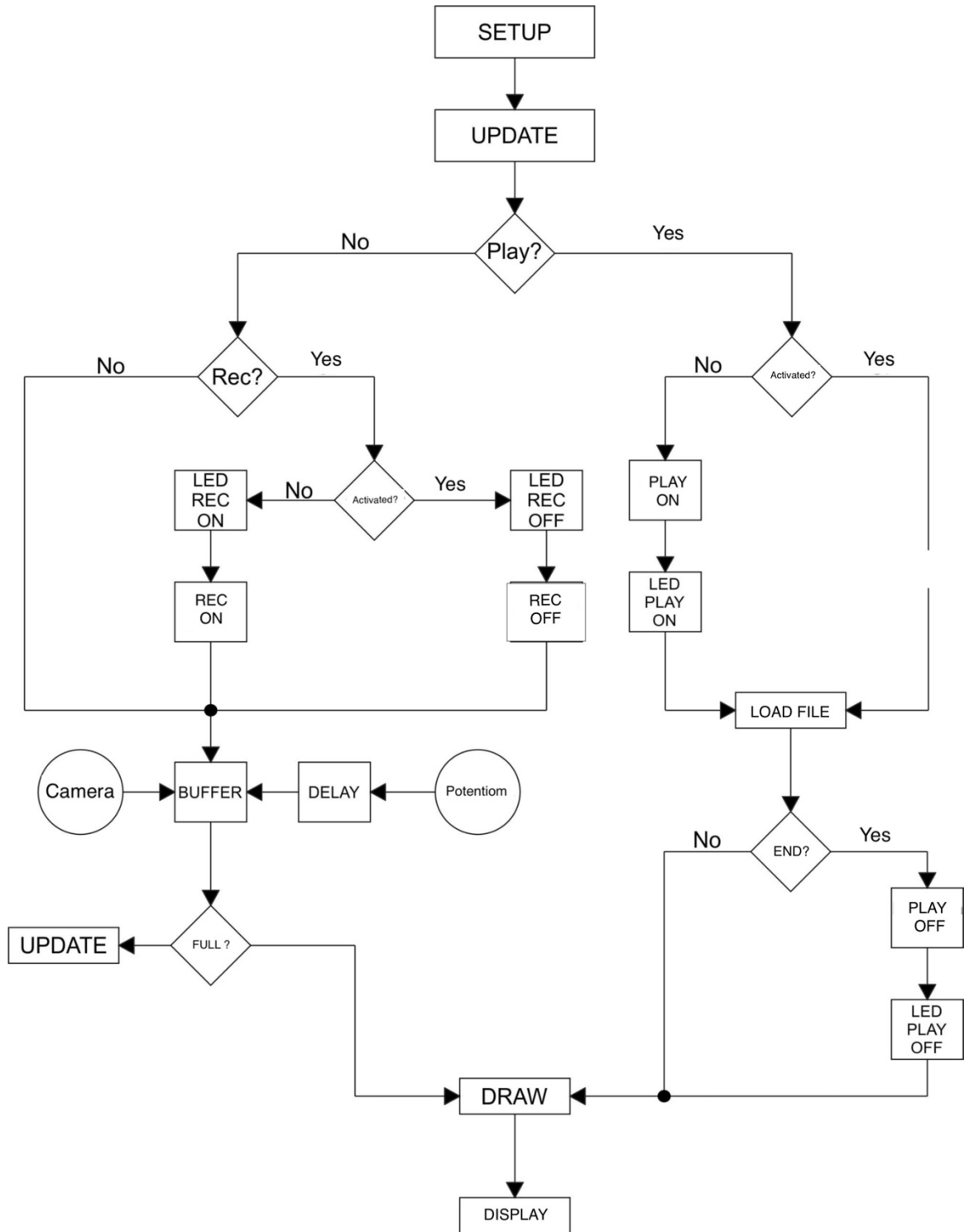


Figure 5-3 Flowchart of Delayed Mirror



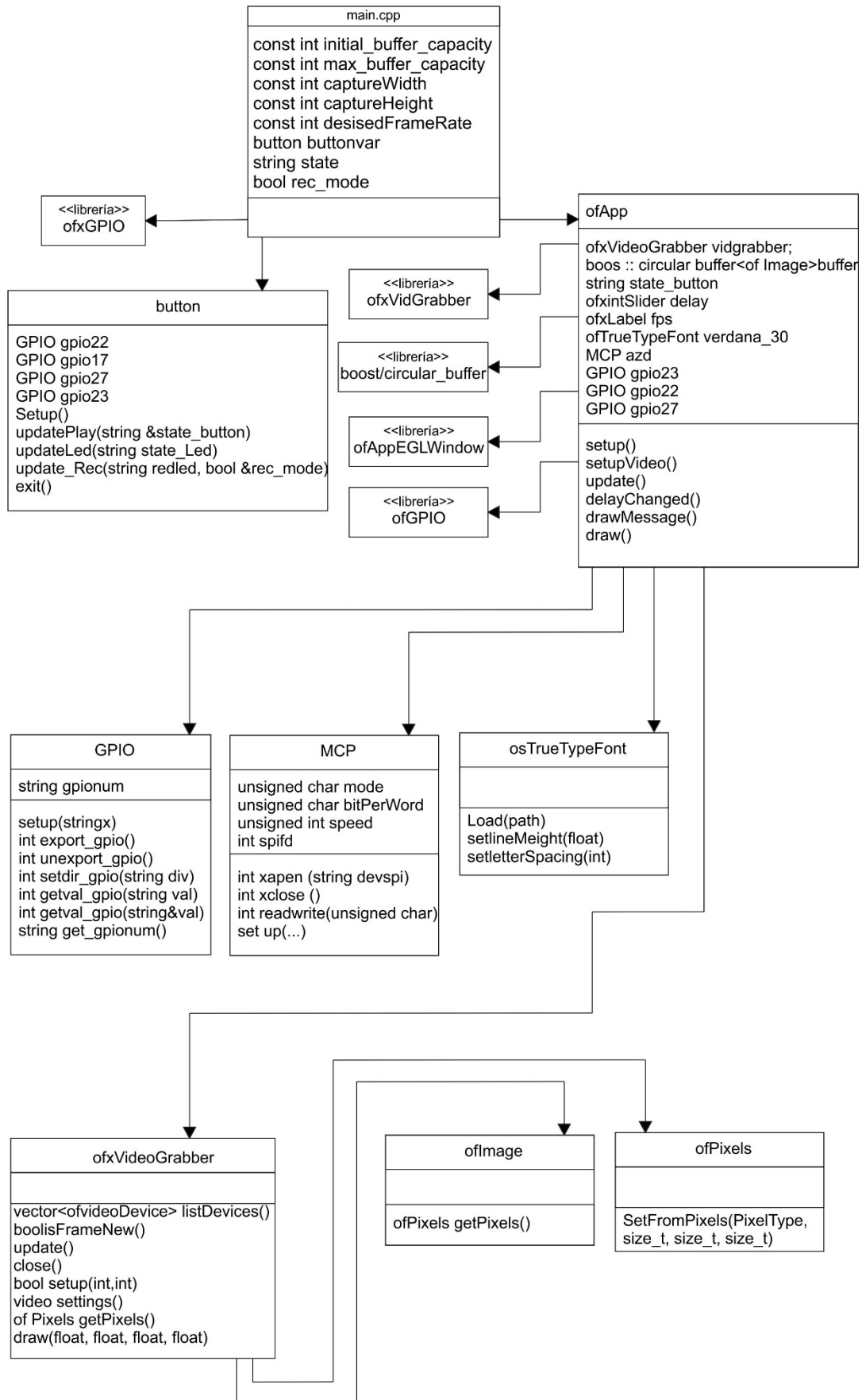


Figure 5-4 Classes diagram, specific for this model

### 5.3.1 Input Functions

The functions that can be considered as input, are those that take data from the outside, and translate them in an appropriate way to the way they should be treated throughout the process.

First, the development that allows taking images from an external device, such as a *webcam*, and process them is studied. In the case of this project, it is desired to take the information coming from said device and store it in an image buffer.

In that sense, the *openFrameworks* libraries offer a wide variety of functions that allow working with this type of information. First, you must define the main class, which, by default in this type of environment, is known as *ofApp.cpp*, which will have an associated constructor *ofApp.h*, where all the functions and global variables of the system are defined, all this is clear without counting the *main.cpp* class, from where the previous main function will be called, because we must not forget that despite everything we are still in a C ++ environment.

Throughout the development, and to avoid confusion, it will be referred to as the main class or function depending on what it refers to at that moment, while the *main()* function will be mentioned by name.

This main class will be defined by its main function *ofApp()*, and it will have associated a series of input and output parameters as shown below.

```
ofApp(int initial_buffer_capacity, int max_buffer_capacity, int _camWidth, int  
_camHeight, int _desiredFrameRate, GPIO & gpio_rec, GPIO & gpio_led_rec, GPIO &  
gpio_play );
```

At the moment, only part of these entries are explained, and the rest of the most relevant are developed throughout this section.

Starting from the inputs of type integer related to the *buffer*, it has to be said that they will be responsible for describing the main parameters of the operation of this device, since they define the maximum capacity that the *buffer* can have where they will be stored. captured images, as well as the initial capacity, which later will be translated as maximum reachable delay and delay in the moment of starting the device, all this will be predefined in the code at the beginning, inside the *main* function, although the initial capacity, or variable delay, will be modified within the function itself throughout its use.

As for this *buffer*, it will be a circular *buffer* in which images are saved as they are captured by the camera, but which, depending on the chosen delay, will have a greater or lesser capacity, and therefore the number of images retained will be variable, depending on the delay that is selected. The fact that it is circular, is mainly because there is no infinite memory, and so, as the images are shown by the output device, they are coming out of the last positions of said *buffer*, and those that were in Previous positions then occupy the last positions to be shown successively. This buffer is defined within the header file as shown below where it can be seen that it is defined as a kind of *array* of images.

```
boost::circular_buffer<ofImage>          buffer;
```

On the other hand, parameters related to the camera are introduced, such as width or height, which will define the size of the image to be processed, which allows controlling the size of the information to be processed from the beginning. .

Within this main class, in addition, a series of associated functions are already predefined, although not developed, such that, as described in previous chapters, programming is easier, since the class has these functions associated with them. other subclasses that allow thus to use certain functionalities in a friendlier way in terms of their programming. One of these functions is the one that configures, and establishes the video input parameters.

***void ofApp::setupVideo()***

Said function would be called within the function *setup()*, generic of the class, which will be called automatically and will configure all the elements that are in it. Therefore, for this particular function it is not necessary to import any additional library, since the video variables are fully integrated into the main class, in addition to being predefined within what has been previously commented, so it would have to be developed directly according to the classes defined by *openFrameworks*.

Thus, in this function a search would be made of the available devices, peripherals connected at that moment to the device, which are capable of recording video, and once found, configure the first of them, according to the specifications given as input. This would already be configured the external element that would capture the user's work environment, as well as itself.

***ofxGPIO.h***

Continuing with the input elements, it is observed that entries related to GPIOs are added. Well, in this case, it must be added that it would be necessary to add an additional library in the form of add-ons, mentioned in previous sections, in order to be able to connect said code with the board.

These libraries are easily found within the *openFrameworks* page itself, which undoubtedly facilitates part of the development work. These libraries are defined in *ofxGPIO.h* and, once added in the appropriate directories, they can be used simply by adding the *#include* line of them. In this way a button class is used, the GPIO add-on has already been defined, and the *buttonvar* object is defined on it, in which all the necessary GPIOs are defined internally.

As with the image parameters, the input and output elements of the device related to these pins are also defined in the *main()* class, so that, in this way, they can be used later within the main class *ofApp()*. That is, the button class is defined and the necessary elements would be defined within it.

```

GPIO gpio22, gpio5, gpio17, gpio27, gpio23;

void setup() {

gpio22.setup("22"); // PLAY Button
gpio22.export_gpio();
gpio22.setdir_gpio("in");
gpio23.setup("23"); // REC Button
gpio23.export_gpio();
gpio23.setdir_gpio("in");
gpio17.setup("17"); // GREEN LED for PLAY
gpio17.export_gpio();
gpio17.setdir_gpio("out");
gpio27.setup("27"); // RED LED for REC
gpio27.export_gpio();
gpio27.setdir_gpio("out");
}

```

In this way, and as defined in the previous chapter, GPIO 22 and 23 are configured, which, as a reminder, do not correspond to the numbering of the pins on the board, since these would be 15 and 16 respectively, as inputs coming from the push buttons. The GPIO 17 and 27 are configured as outputs, which will be connected to the *Play* and *Rec* LEDs defined in the previous chapter, and thus control their switching on and / or off.

In addition, there are defined within the *buttonvar* constructor functions to update the state of the LEDs, so that they initialize said elements as off.

Once the input parameters of the main function have been defined, already within it, in its header file *ofApp.h*, the input parameters related to the SPI connection of the device are defined, which will be, as seen above, the that provides the connection with the terminals of the potentiometer. The reason why this configuration is added within the main class is none other than the fact that it

significantly affects the main function of the device, which is none other than the variation of the delay, which in turn, as its own definition indicates, it must be a variable that can change at any time, affecting different parameters within this main class, which means that it can therefore be considered a better strategy to define it within the main class.

Therefore, within the header file *ofxApp.h*, the SPI parameters are defined as shown below:

```
MCP a2d;

int a2dVal = 0;

int a2dChannel = 0;

unsigned char data[3];
```

The MCP class is predefined in the GPIO addon, when created, it will automatically generate in its constructor a configuration of the Raspberry SPI pins, which by default will accept 8 bits per word, will be configured in the SPI 0 mode, and the speed of transmission would be 1Mbit per second.

Although, in the case of wanting to take other parameters, this possibility is also offered, since, in the same constructor, the parameterized function is established, as can be seen below.

```
MCP() {

    this->mode = SPI_MODE_0 ;
    this->bitsPerWord = 8;
    this->speed = 1000000;
    this->spifd = -1;
    this->xopen(std::string("/dev/spidev0.0"));
}

MCP(std::string devspi, unsigned char spiMode, unsigned int
spiSpeed, unsigned char spibitsPerWord){
```

```

    this->mode = spiMode ;

    this->bitsPerWord = spibitsPerWord;

    this->speed = spiSpeed;

    this->spifd = -1;

    this->xopen(devspi);

}

```

The *devspi* parameter is a string of characters that indicates, in the case that there is more than one SPI device connected, which must be configured, containing in its value the path in the computer directories, where to find that element. In the case of this project, since it is always a single system connected by SPI, the *spdev0.0* will be taken, since, as can be seen in the configuration defined in the previous section with respect to the SPI pins, it is the GPIO that is taken of the possible.

The object associated to this class, for this development will be *a2d*, whose functions allow to read the information related to the output channel of the converter, and that later is transformed into an integer, *a2dVal*, which will be the value read from the output of the potentiometer, transformed our code. In this development, although it is predefined in the constructor, as previously mentioned, but the said object is defined in the generic function *ofApp::setup()* of *openFrameworks*, so that it loads when launching the code and thus achieve the necessary objective .

Before continuing, when mentioning the value that is taken at the output of the potentiometer, it has to be said that in this case the voltage value measured as output of the voltage divider generated by the potentiometer is not relevant, since it is directly connected to the *Raspberry* feed, the maximum value does not exceed this voltage, and therefore cannot damage any component, and the minimum will be mass, which produces an even less harmful effect, so it is not necessary to have Note that there may be peak voltage for example. All the values are transformed so that they adapt to the maximum delay imposed, this being imposed from the beginning.

Once the variables are configured, it is defined below how they will be used, within the *updateSpi()* function.

***void ofApp::updateSpi()***

The main purpose of this function is to constantly update the information received by the SPI pins, since they are defined as global variables that will be modified, it will not be necessary to take them as a parameter, but once this function is called its value will remain just as it comes out of it. They are described, therefore:

```
data[0] = 1;
data[1] = 0b10000000 | ( ((a2dChannel & 7) << 4) );
data[2] = 0;
a2d.readWrite(data);
a2dVal = 0;
a2dVal = (data[1]<< 8) & 0b1100000000;
a2dVal |= (data[2] & 0xff);
```

The array *data[]* will store in position 1 the information related to the chosen input channel, which will be collected in *a2dChannel*. In the case of this configuration, the channel chosen in MCP3008 is 0, hence its value, while positions 0 and 2 will start with value '1' and '0' respectively. Once these values are established, the value read from the MCP is taken, with the function *readWrite()* and this information is saved in the array, the value of *a2dVal* is initialized to zero in case no chip value is collected, and subsequently take the two most important bits of position one of said array, displaced eight positions to the left which would imply multiplying its value by 256 in integer, and resetting the rest of values. The latter is done to have more accurate values. And finally the value of the content of the position two of the *array* is added, in this way a range of values contained between 0 and 1023 is always obtained, regardless of the converter that is used in the family.



After this process, an integer is obtained, momentarily, since its value will have to be continuously updated to take any variation in the potentiometer. This value is the integral equivalent of that taken by the SPI terminals, and it will be the one used to define the delay, a process that is seen in the following section.

In this way, all the elements of the code related to the entries are defined. The development continues now with the part related to the processing of the information.

### 5.3.2 Processing functions

Basically, the functions of processing the data obtained from the inputs are summarized in one. The function *ofApp::update()*.

#### ***void ofApp::update()***

As previously described with the *setup()* function, this function is predefined in the *openFrameworks* libraries. When code is added to this function that code will be executed iteratively, until the end of the process, either because it is interrupted, by any type of external element, or because the process ends, which makes this one of the most important functions when it comes to doing any type of development with this type of tools.

In this case, the first thing that is done in this function is to update the value of the variable *a2dVal* naming the function defined for it, defined in the previous section. Once this is done, its value is translated into a content between 0 and the maximum delay value, which will be defined from the beginning as the maximum capacity of the buffer. *This is done as follows:*

```
New_Delay = (a2dVal*MaxDelay)/1023;
```

Subsequently, the state of the buttons is evaluated, connected through the different pins of the board.

In the event that the external play button is pressed, the system will start playing a video that has been stored, thus stopping all other functions, and will update the status "on" of the LED associated with said button.

If, on the other hand, this button of the physical interface is not pressed, the Rec button is evaluated, in which case, the fact that it has been pressed implies that the system will begin to record what is coming out of the buffer. And, as in the previous case, the status of the LED related to this function is updated, which will remain on until the recording is stopped, due to the fact that the button has been pressed again.

Another important task within this function, would be the part where the delay shown by the screen is updated, through the delay object of the ofxIntSlider class, which is nothing else than a bar in the upper left corner of the screen in which it shows the delay in seconds that you have at that moment, which will indicate to the user at all times what the delay is, it can be controlled more accurately, not only when turning the potentiometer, but also through the image shown by the screen.

Once, and as the flow chart shows, it has been checked if a button has been pressed or not, the required action has been taken, and the delay has been read and updated, the next step is to record the buffer image captured from the camera, defined from the class ofImage, in said buffer, placing it in the first place available so that when it is filled it is the first one to leave and therefore be displayed. This process will be done every time the update() function is executed, which it does sequentially before draw(), in an infinite loop until an event interrupts it.

### 5.3.3 Output functions

This section is summarized in a single function, the **draw()** function. It is a function that is part, together with setup() and update(), of the three main functions that any application based on openFrameworks has, it is executed once the process of the

update() function of the previous section has been completed. , as well defined by openFrameworks.

In this function it simply aims to show the series of images that are stored in the buffer once it is full, that is, once it has been fulfilled that the number of images or frames stored in said buffer is equal to its size, which is translated to a high level in which they will be displayed once the delay imposed by the potentiometer has been met. In the period during which this buffer is being filled, the screen shows the phrase "I am capturing", thus indicating to the user the state in which the system is located, once it is full, the recorded image appears directly, with the corresponding delay . Indicate only this function, for the easy understanding of the reader, part of the conditional structure that corresponds to this function:

Internally, and thus implemented as a base in *openFrameworks*, this function looks for a means by which the image can be transmitted, which in this case is the *Raspberry* system's HDMI port, since this device does not have another image output port, then check that there is a device connected to this port, and finally, launch this image of the *buffer* to be displayed on said device.

```
if (buffer.size() < buffer.capacity()) {
    string msg = "Wait, it is loading...";
}
else {

    buffer.front().draw(0,0,videoWidth,videoHeight);
}
```

## 6 TESTING

This chapter describes the different tests that the user must do to ensure that the different functional requirements of the system described in section 3 are met. In this way, the commitment acquired with the user is achieved, which in some cases can be a client.

Test number: 1	Capture image	Requirement: 1
<p>The system must be able to take a picture of the environment. For this, once the system is connected, at some point the image that has been captured by the webcam must be able to be shown on the screen.</p> <div data-bbox="485 457 1302 871" data-label="Image"> </div>		

Table 6-1 Test 1. Requirement 1.

Test number: 2	Image storage	Requirement: 1.1
<p>It must be possible to save an image on the system that has been captured by the camera at a certain time prior to verification.</p>		

Table 6-2 Test 2. Requirement 1.1.

Test number: 3	Image processing	Requirement: 1.4
<p>The images stored in the system must be able to be modified so that they are displayed on the screen, so that one or more of their characteristics will differ from the original ones.</p>		

Table 6-3 Test 3. Requirement 1.4


Test number: 4	Image inversion	Requirement: 1.5
<p>The image shown on the screen must have the same characteristics in terms of layout that would be reflected in a mirror.</p> 		

Table 6-4 Test 4. Requirement 1.5

Test number: 5	Delay of the image	Requirement: 2
<p>The system must be able to show on screen the image that the camera records with a time delay that the user must be able to modify by adjusting the driver designed for it.</p>		

Table 6-5 Test 5. Requirement 2


Test number: 6	Delay variation	Requirement: 2.1
<p>The user must be able to control the time it takes for the image captured by the camera to be displayed on the screen.</p> 		

Table 6-6 Test 6. Requirement 2.1.


Test number: 7	Maximum delay	Requirement: 2.2
<p>The maximum time an image that is captured by the camera takes to be displayed by the screen will in no case exceed 25 seconds.</p> 		

Table 6-7 Test 7. Requirement 2.2.


Test number: 8	Minimum delay	Requirement: 2.3
<p>The minimum time that an image that is captured by the camera takes to be displayed by the screen will in no case be less than 0 milliseconds.</p> 		

Table 6-8 Test 8. Requirement 2.3

## 7 CONCLUSIONS AND FUTURE LINES.

For ending with this thesis, this chapter presents itself which it is intended to see a series of reflexions which are reached after the study, development and project implementation. Also the advantages of being elaborated on devices based in ARM processors, specific in Raspberry PI 3 B. Finally, how good could be on environments that until now have an use pretty limited of the technology.

In the same way, it shows possible lines of evolution and association with another technologies which can be reached started from this prototype.

### 7.1 Conclusions

Reaching the final developing of the code about the tool openFrameworks, once implemented the necessary circuitry, and once checked the other functions of the system, the following final conclusions can be determined.

- The processors ARM, and the systems based in them, contribute a great progress regarding the development of the tool at user level. Mainly

because contribute to the power that suppose a microprocessor, a RAM or a CPU enough for the implementation of multitude of applications.

- The portability is another of his characteristics most important, as it allows a greater ease of use by the user, making it possible the use of this type of applications in diverse environments, without the limitation that the size implies.
- The low consumption of these devices allows that you can eliminate the connection to the electric grid, replacing it with an external battery, like those that could be used as external load source for the mobile phones. Which, at the same time increase the grade of portability of this kind of systems.
- The simplicity that supposes the integration of electric components with Raspberry devices, suppose a great approach of electronics to a greater number of users. This is achieved by making these developments, more friendly procedures, which motivates learning this kind of technology by users, that do not have to be so involved in this kind of environments.
- The projects based in this kind of systems, also have a low economic cost. Raspberry is of course the component whose price is bigger. But the fact of being a system designed and developed mainly for a use by students, implies, that its price has to be quite reachable. So that any student is able to acquire it.
- The tools OpenFrameworks suppose a great advantage in terms of developments based on C++, object oriented, since they allow create projects in a simple way, getting very good results, especially as far as the visual part is concerned.
- Thanks to the large number of libraries, that increases every day due to its open code software nature, the multitude of projects that can be developed is immense.
- Despite this, there is still a certain *hole* in terms of the combination of Raspberry and OpenFrameworks. Although this does not imply the impossibility of the implementation of certain developments, it is actually the opposite, can ever motivate the developer by discovering for himself how it would be possible to implement certain features still to be discovered.
- Nowadays, the microcomputer systems, are still small, for the development of some applications, especially in those related to videos, due to the overload what works means with this kind of information. This is the case, for example, of the limitation in the size of the buffer used in the development. Although this is not a limitation, since this type of systems follows a constant evolution, where one of its main objectives is to improve the main characteristics of the microprocessor.



- Thanks to these projects, you can observe the certain technological emptiness that exists in certain environments, such as artistic installations, which can be a motivation for people of entrepreneurial nature for the development of different ways of reaching this field of users, getting better and thanks to the technology, their most routine activities, as is the case of *Delayed Mirror*.
- The great boom supposed due to put on the market the 3D printers, suppose a great advantage for this kind of development, giving the opportunity to create more personalized interfaces and adapted to the requirements of the users, which suppose another additional advantage, that facilitate, even more, the portability of the system.
- Otherwise, it is allowed to present the device a more suitable format for the final user of the system, abstracting it even more of the technology, that in some cases can be a difficulty for certain users, and improving its use in terms of user-system interaction.

## 7.2 Future lines

During the development phases and subsequent realization of this project, it has been possible to observe the variety of possibilities that are opened starting from this prototype. Attending mainly to the large number of elements compatible with the motherboard of Raspberry Pi 3 B, the following possible extensions of the system can be rated:

- As mentioned in the previous section, Raspberry will evolve improving some of the characteristics as it has been doing in recent years. This brings a great advantage in terms of this prototype as they will be able to continue expanding their capabilities, for example and with special relevance, the one related to the maximum buffer size.
- Thanks to the incorporation of a Bluetooth module over the motherboard, you can study its possible connection in such a way that it is possible to control said system from another remote system connected through that wireless technology.
- A specific case would be one in which the remote system was the Smartphone, where you could even consider the possibility of designing an application simulating the user interface of the prototype from which you could control completely from a certain distance.
- At the same way, the use of the Wifi module of the device could be evaluated, both to develop the previous function, and to take advantage of the Internet connection in such a way that everything shown on the display could be retransmitted. Something that would be very useful in areas such as distance education.

## REFERENCES

- [1] V. T. Soccol, A. Pandey, y R. R. Resende, Current Developments in Biotechnology and Bioengineering: Human and Animal Health Applications. Elsevier, 2016.
- [2] L. Molina-Tanco, C. García-Berdónés, y A. Reyes-Lecuona, «El Espejo Retardado: una innovación tecnológica para el aula de danza», presented at V International Congress of: Danza, Investigación y Educación, 2016.
- [3] Y. Liu, «Electronic and Information Technology Applications in Sports», J. Residuals Sci. Technol., vol. 13, n.o 8, 2016.
- [4] S. C. Timón Ramón Méndez, Pedro Martín, Víctor Sánchez y David, «Desarrollo tecnológico en el deporte | La Huella Digital». .
- [5] «Eben Upton», Wikipedia, la enciclopedia libre. 30-oct-2015.
- [6] «Raspberry Pi Foundation - About Us». [Online]. Available at: <https://www.raspberrypi.org/about/>.
- [7] «Products - Raspberry Pi». [Online]. Available at: <https://www.raspberrypi.org/products/>.
- [8] M. Richardson y S. Wallace, Getting Started with Raspberry Pi. O'Reilly Media, Inc., 2012.
- [9] A. Pajankar, Raspberry Pi Image Processing Programming: Develop Real-Life Examples with Python, Pillow, and SciPy. Apress, 2017.
- [10] «about | openFrameworks». [Online]. Available at: <http://openframeworks.cc/about/>. [Last access: 26-oct-2017].
- [11] «documentation | openFrameworks». [Online]. Available at: <http://openframeworks.cc/documentation/>. [Last access: 28-oct-2017].
- [12] «i2c--201471211348805.pdf». [Online]. Available at: [http://www.lcis.com.tw/paper\\_store/paper\\_store/i2c--201471211348805.pdf](http://www.lcis.com.tw/paper_store/paper_store/i2c--201471211348805.pdf). [Last access: 31-oct-2017].
- [13] «I2C», Wikipedia, la enciclopedia libre. 25-ago-2017

- [14] «usermanual\_mm112\_en.pdf». [Online]. Available at: [https://www.velleman.eu/downloads/0/user/usermanual\\_mm112\\_en.pdf](https://www.velleman.eu/downloads/0/user/usermanual_mm112_en.pdf). [Last access: 01-nov-2017].
- [15] «MCP3221 Low-Power 12-Bit A/D Converter with I2C Interface Data Sheet - 20001732E.pdf». [Online]. Available at: <http://ww1.microchip.com/downloads/en/DeviceDoc/20001732E.pdf>. [Last access: 01-nov-2017]
- [16] «Serial Peripheral Interface (SPI) - learn.sparkfun.com». [Online]. Available at: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>. [Last access: 01-nov-2017].
- [17] A. Y. R. Hernández y R. C. Sánchez, «CONVERSIÓN A/D CON PROTOCOLO SPI PARA AUDIOFRECUENCIAS», JÓVENES EN Cienc., vol. 1, n.o 2, pp. 1470-1475, dec. 2015.
- [18] «2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface - MCP3008.pdf». [Online]. Available at: <https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf>. [Last access: 29-oct-2017].
- [19] I. Sommerville, Ingeniería del software. Pearson Educación, 2005.
- [20] R. González P., «Ingeniería de requisitos», Ingeniería De Software, 13-oct-2015. .
- [21] M. J. Escalona y N. Koch, «Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo», dec-2002. [Online]. Available at: <https://www.lsi.us.es/docs/informes/LSI-2002-4.pdf>. [Last access: 02-nov-2017].
- [22] «Un entorno metodológico de ingeniería de requisitos para sistemas de información - Fondos Digitalizados de la Universidad de Sevilla». [Online]. Available at: <http://fondosdigitales.us.es/tesis/tesis/30/un-entorno-metodologico-de-ingenieria-de-requisitos-para-sistemas-de-informacion/>. [Last access: 02-nov-2017].
- [23] «D\_L56BID\_0\_L56BID\_DIODO\_LED\_DOCUMENTACION.pdf». .
- [24] «Potenciómetro eje lineal 10K - Ray Online». [Online]. Available at: <http://tienda.ray-ie.com/potenciometros/40-potenciometro-eje-lineal-10k.html>. [Last access: 08-nov-2017]
- [25] «Pulsador Tact Switch de 12x12mm». [Online]. Available at: <http://www.shoptronica.com/pulsadores-smd/454-pulsador-tact-switch-tht-de-12x12mm.html>. [Last access: 08-nov-2017].