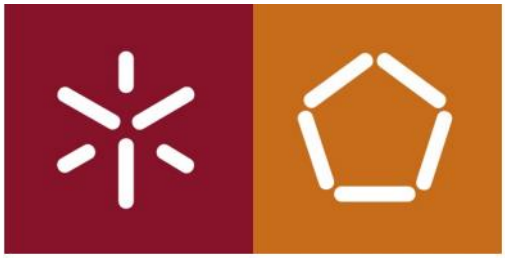


Universidade do Minho
Mestrado Integrado Engenharia Eletrónica Industrial e Computadores
(MIEEIC)

Embedded Systems

Face Detection & Facial Recognition
On a Following Camera

-Project Plan-

Authors:
João Sousa nº82273
Pedro Sousa nº82041

Professor:
Adriano Tavares

Contents

List of Figures

List of Tables

List of Code

1. Introduction

1.1 Problem Statement

1.1.1. Problem Statement

Face detection, as well as facial recognition, has popularized in the days of today due to the rising evolution of technology, of advanced algorithms and of better computational power. Each day even more consumers seek new devices that automatize their everyday ordinary tasks, providing a simpler and a more efficient personal and professional life.

However, the practical introduction of image processing for face detection or facial recognition still predicts a costly and complex implementation. So, we propose to develop a face tracking camera that makes its implementation simpler, without the need for superior understanding about the subject.

A future application of our project be for a smart following camera, such as the ones at the airports. By detecting a face or choosing one with facial recognition, the application will center the camera on the person's face (following the person).

1.1.2. Problem Statement Analysis

The problem statement above describes the needs to be deconstructed and analyzed in more detail. It is mandatory to define the Subsystems and the Entities, as well as their functions and correlations.

There are two subsystems, the Framework and the Application, and both will run on the Device system. The Framework provides to the Application the resources to execute face detection or facial recognition

On the Framework level there are 2 Entities:

- The "Device System" - process the image through facial detecting or facial recognition.
- The "Camera" - for video capturing;

On the Application level there are 4 Entities:

- The "Motor" - moves the camera angle to point to the right position.
- The "Interface" - mainly shows the captured video, the outcome of the framework (image passed through face detection or facial recognition), sent by "Device System".
- The "User" - interacts with "Interface", defining the system parameters, the application mode and the content to be put under survey;
- The "Device System" - operates on the "Motors" and sends the processed video to the "Interface". Through the outcome of the framework, it applies the dedicated control on the motors of the camera to follow and center the person

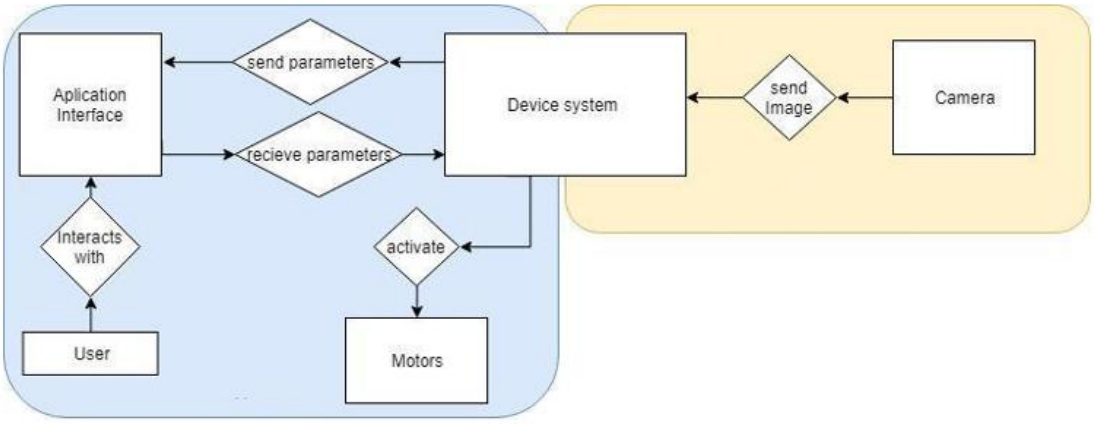


Figure 1- Problem Statement Analysis

2. Market Research

Nowadays, the practical introduction of the new technological advances still predicts a costly and complex implementation. Facial Recognition is an example of that demanding the hand of high-level professionals and some expensive hardware.

The increasing demand for surveillance systems to enhance safety and security are increasing the adoption of facial recognition systems, giving our application an huge potential.

1.1. Target Market

Our application aims for the necessity of a simple implementation of Facial Recognition, without the need for a superior understanding and programming.

External devices and homemade projects could easily add Facial Recognition, were the only procedures are: input video/image and collect the processed data.

1.2. Competitive Analysis

- **Tend Insights Lynx Indoor Camera**

This camera provides a smart video monitoring solution that pairs with any existing Wi-Fi network. It has facial recognition to detect familiar faces and to activate alarm if not. This product represents an end-purpose of our project.

It cost around 50$ and needs paid subscription to video cloud storage.

Figure 2- Tend Insights Lynx Indoor Camera

Our goal isn't to develop a surveillance camera, but to provide a Facial Recognition application. As so, there are specific advanced companies dedicated in the use of Facial Recognition, offering dedicated services:

- **Panasonic Corporation: FacePRO**

Panasonic with the WV-ASF950, offers a facial recognition engine that features real-time facial recognition and face search. FacePRO also has high accuracy to recognize within common video surveillance technology.

Figure 3- FacePro

There isn't a fixated price due to the its flexibility. The company provides a dedicated solution system to the specific problem, making each case a different one.

- **Animetrics: FaceR API**

The company Animetrics presents a variety of services on algorithms for facial recognition. In particular, Face Recognition API can be used to find human faces, detect feature points, correct for off angle photographs, and ultimately perform facial recognition.

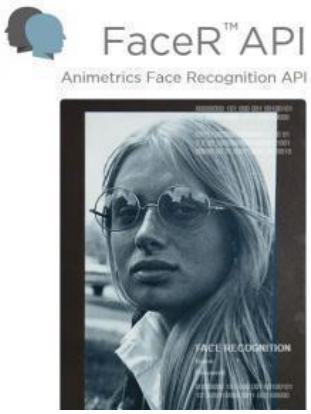


Figure 4- FaceR API

3. Analysis Phase

3.1. System

3.1.1. System Overview

With system overview we make a high-level description preview of the application where our framework is going to operate.
The System is composed by 3 components:

- The Camera - the component that transmit the video for the system;
- The Raspberry Pi – the main station where our application will run, using the help of OpenCV library for Facial Recognition and reading of angle servo-motors;
- The Servo Motors - responsible for the camera movement;

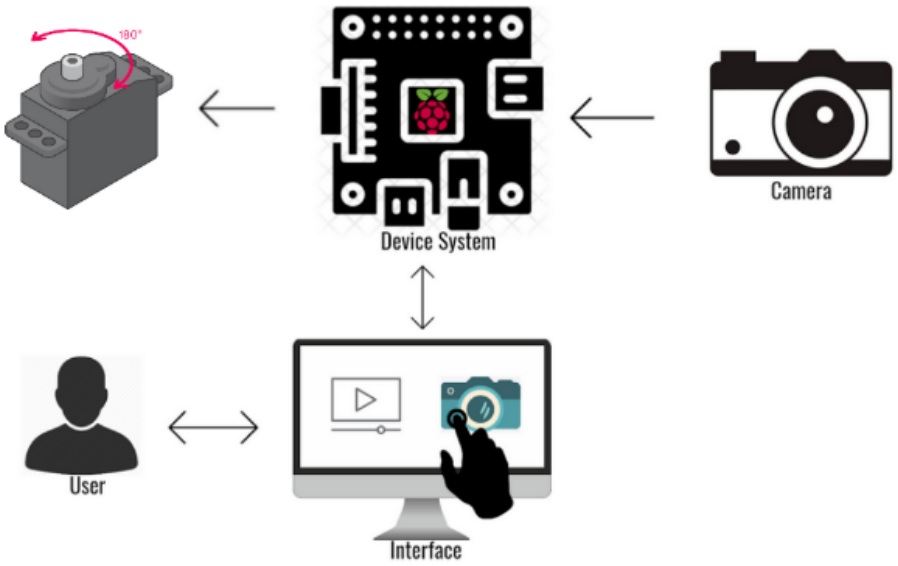


Figure 5- System Overview Diagram

The Device System after producing the result data of the Detection (or Recognition) it centralizes the camera on the person's face. As so, there is an external platform running the Interface where the video is shown, and the mode defined.

3.2. Requirements and Constraints

In order to the better predict a good performance of the final product, its mandatory to establish all the requirements (functional and non-functional) and all the constraints (technical and non-technical) implied on the project.

3.2.1. Non-Technical Constraints

Represents all the limitations that exist apart from the practical development of the project. In this case, the project group must:

- Not surpass the final deadline;
- Maximum of two students per group;
- Be budget friendly duo to money limitations;

3.2.2. Technical Constraints

Represents all the established limiting factors that affects the execution of a project. In this project, its implementation requires:

- Raspberry Pi 3b+;
- Embedded Linux and Buildroot;
- C/C++ Object Oriented Programming;
- Implement own device driver;
- Pthreads;

3.2.3. Functional Requirements

It defines all the specific functionality that the system will accomplish.
As so, the framework will:

- Detect faces;
- Recognize faces;
- Retrieve and send data according to the needs of the Application;

The Application will:

- Adjust the angle of camera to center the face;
- Use the framework for face detection or facial recognition;
- Display the video in a real time;
- Display a mode selection menu;
- Display controls for the motors;

3.2.4. Non-Functional Requirements

It defines all the quality attributes we pretend our system to have. As so, we design it to be:

- Friendly user and coherent interface;
- High flexibility;
- Fast Real-Time System;
- Low cost;

3.3. System Architecture

As any other good project, it starts on a good fundamental well-constructed Architecture. Defining the link between the subsystems and making the connection between the hardware and the software, is the essential key to achieve the required state of quality.

As defined previously, the subsystems of our project are the Framework and the Application. Our framework is flexible and not limited to the limited to the Applications needs. Their communication is based on the Application request of the Framework utilities, and the Framework provides the result to the Application.

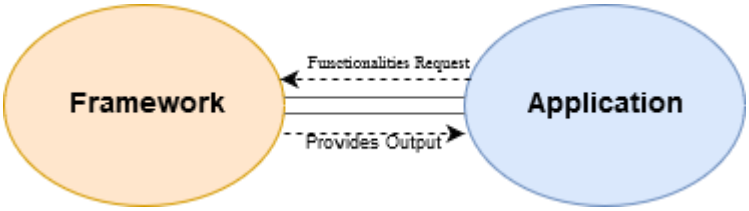


Figure 6- System Architeture Diagram

3.3.1. Hardware Architecture

This example of application´s hardware consists in using a Raspberry PI as the main processor and it will receive the input image from the camera and the angle of servo-motors. Also consists in a External Interface to show real video capturing.

The diagram in figure 7 shows a simple representation of the Hardware.

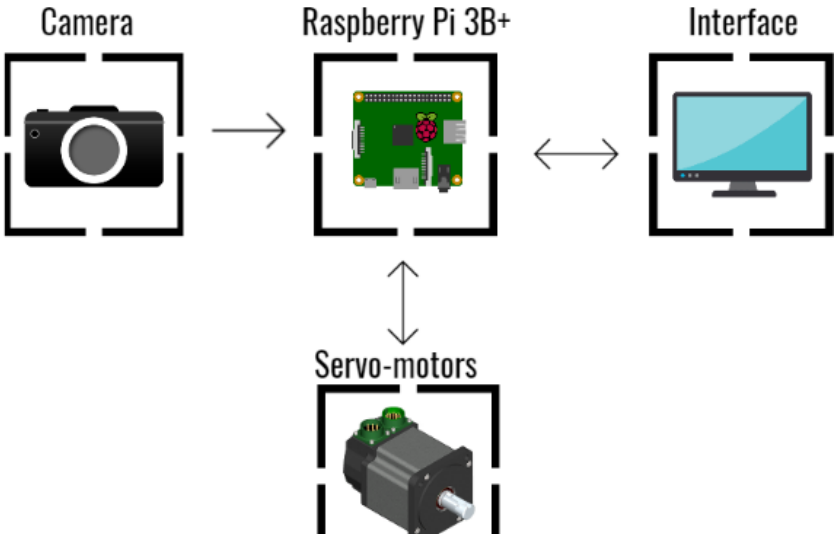


Figure 7- Hardware Architecture Diagram

3.3.2. Hardware Specification

With hardware specification we have a more detailed prescription of all of elements that are included on our application. As so, the components are:

- Raspberry Pi 3 Model B+ - Responsible to receive data from camera and processing the image.
  - Broadcom BCM2837B0, Cortex-A53 64-bit @ 1.4GHz;
  - 1GB LPDDR2 SDRAM;
  - 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2, BLE;
  - Extended 40-pin GPIO header;
  - Full size HDMI port;
  - CSI camera port to connect a Raspberry Pi camera;
  - 4 USB 2.0 ports;
- Raspberry Pi Camera – Responsible to transmission the image
  - IMX219 8-megapixel sensor;
  - attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi

- Servo Motors – Responsible for the management of the camera movement and angles adjustment.

3.3.3. Software Architecture

In terms of the software, our system requires some specific drivers such as the camera driver and servo-motors driver. The middleware covers the acquisition, the camera angle adjustment and processing of the data though libraries like OpenCv. As application there is the Graphic Interface that shows the result of the Face Detecting.

The diagram in figure 8 shows a simple representation of the Software.

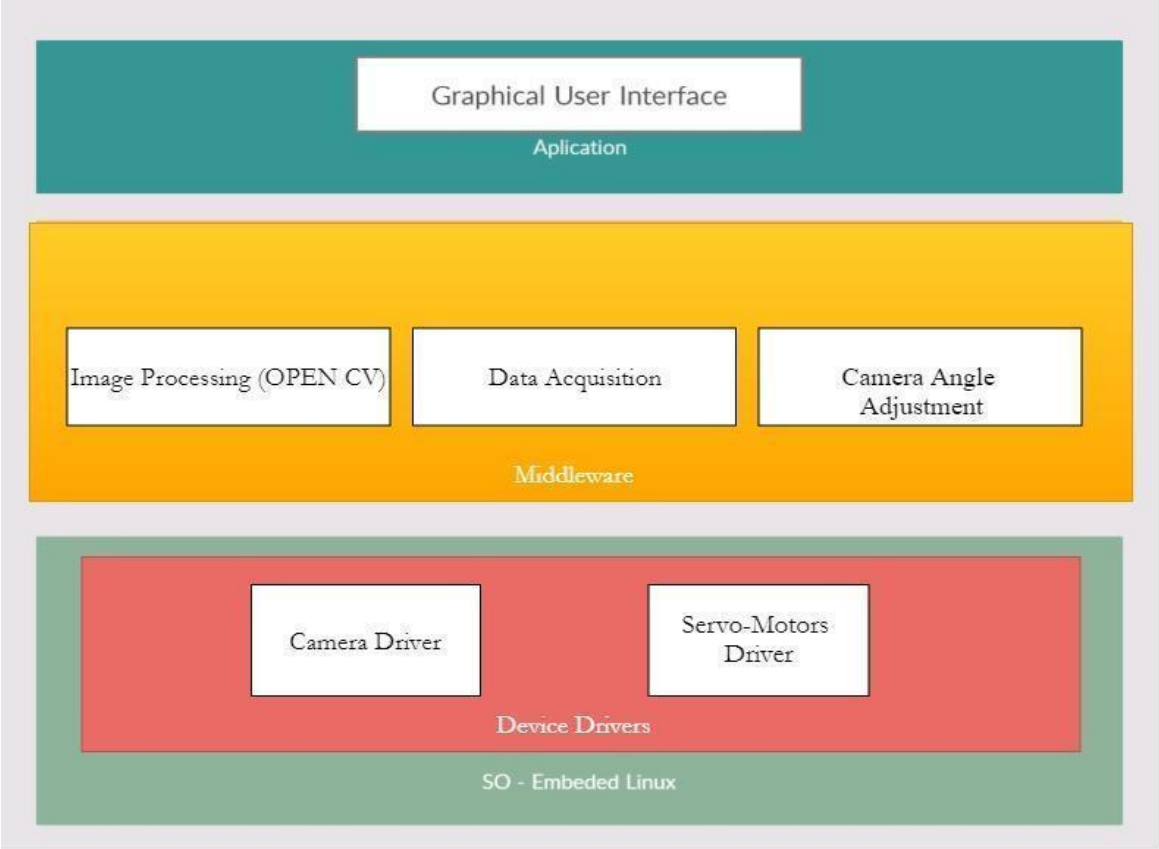


Figure 8- Software Architecture Diagram

It's represented the three layers of the application where our framework will be used. It's also represented the Embedded Linux System, where are located the device drivers, the middleware and the application.

3.3.4. Software Specification

The software specification represents all of tools that we will need to build to have a functional System. As so, our software elements are:

>Buildroot - for cross platform development;
>OpenCV Library- for the computer vision and apply it in the facial recognition;
>Pthreads API - for multitasking;
>Qt creator - for build the interface;
>Embedded Linux - Operating System;
>C/C++;

3.4. Local System

3.4.1. System Events

The board below presents all the events that can be triggered in our framework, and are divided by name, description, source and type.

The table below shows how many synchronous events (that occur in a sequential manner or that are predictable) and asynchronous events (that that are not predictable and that may or may not occur) we have.

The events that need to be synchronous are the "Refreshing Streaming", the "Processing of Image" and the "Video/image sample trigger". These events are responsive to acquire the image/video (from camera or pc storage) and send it to interface and/or the framework.

The "User provides Image/Video" is asynchronous as is happens only when the User decides to do it so.

Event Name	System Response	Source	Type
User provides image/video	Send image/video for processing	User	Asynchronous
Video/image sample trigger	Acquire and process image for recognition	Local System	Synchronous
Process image trigger	Analyze the image and detects face.	Local System	Synchronous
Refresh streaming	Rendering the streaming of image/video to interface.	Timer	Synchronous

Table 1- System Events Table

In the case of our future application, we present the following table below to represent its events.

Event Name	System Response	Source	Type
Control Camera Movement	Rotate the camera based on the User choices.	User	Asynchronous
Sensor Data Gathering	Collects samples of servo-motors angle	Time	Synchronous
Data Analysis	Process the data collected for adjusting of image.	Local System	Synchronous
Adjustment of Camera	Moves motors based on position of the detected face	Local System	Synchronous

Table 2- Application Events Table

As we can see with the table above the only event asynchronous in the application is the "Control Camera Movement" that will be controlled through the external interface. The rest of events will be synchronous such like the reading of the "Servo-Motors Angle, the "Data Analyses" and the "Adjustment of Camera.

3.4.2. Use Cases Diagram

In our system the primary actor is the "Application" that will use our framework. It is going to be responsible to achieve the goal of our system.

As we see bellow on diagram (Figure 9), when the application interacts with the framework, the system can search for an image on storage, or capture an image from camera. When one of these events occurs it then proceeds to analyze the faces on the image. If the system successfully recognizes or detects a face, the system sends the processed image to the interface.

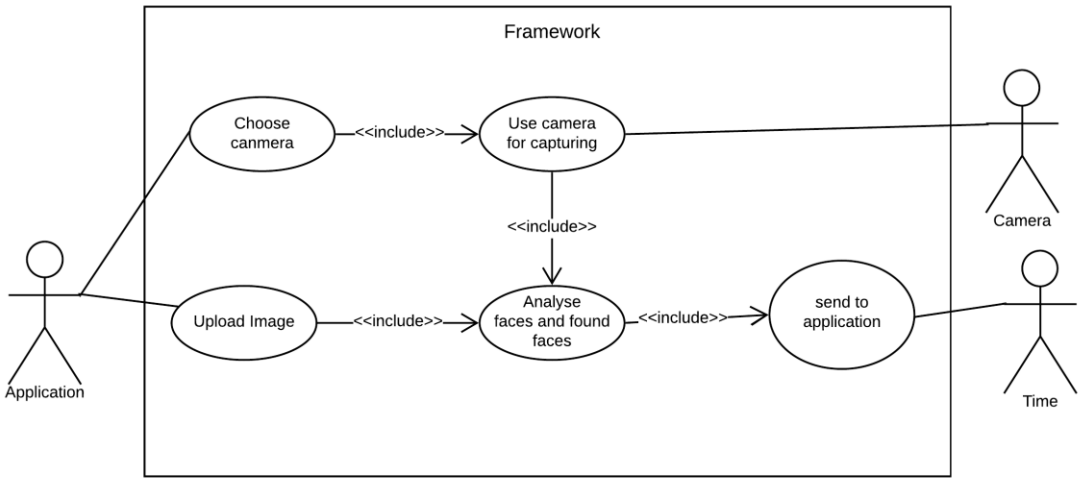


Figure 9- Use cases diagram for Framework

A user can handle the application in two different modes:
-Automatic mode:

The user through an interface, decides the way the systems will operate, just use face detection and follow a selected face, or use facial recognition and follow the person if recognized.

With the diagram below we can see better how application will work using our framework. We can see that after the user's decision, the data is analyzed, and it's applied the correction of the camera angle. At the same time the interface is receiving the processed data.

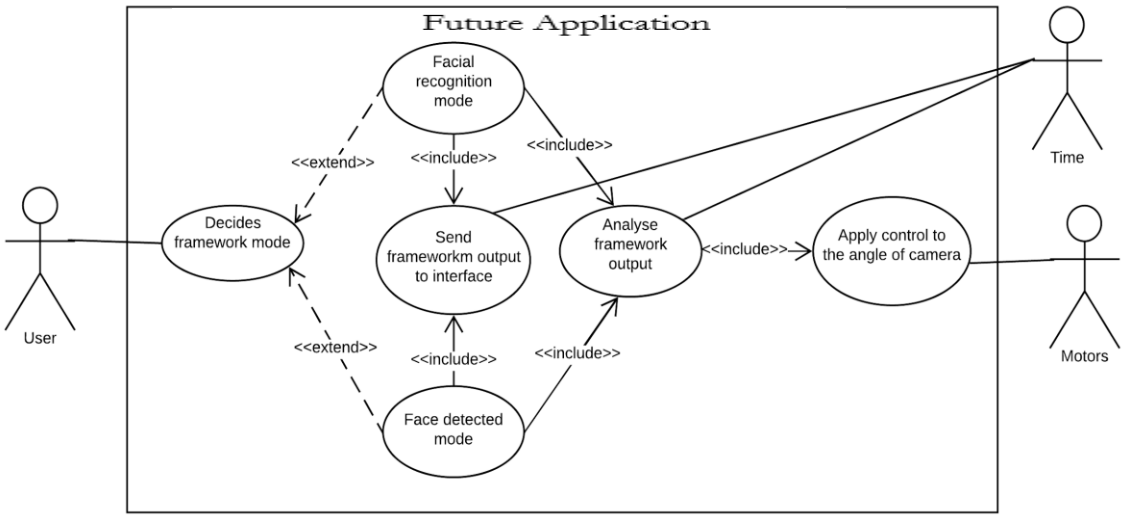


Figure 10- Use cases of the future Application (in automatic mode)

-Manual mode:

The user remotely controls the direction the camera. In this mode the User is allowed to decide which mode of framework he desires to operate.

At the end the external interface will receive the data that was processed.

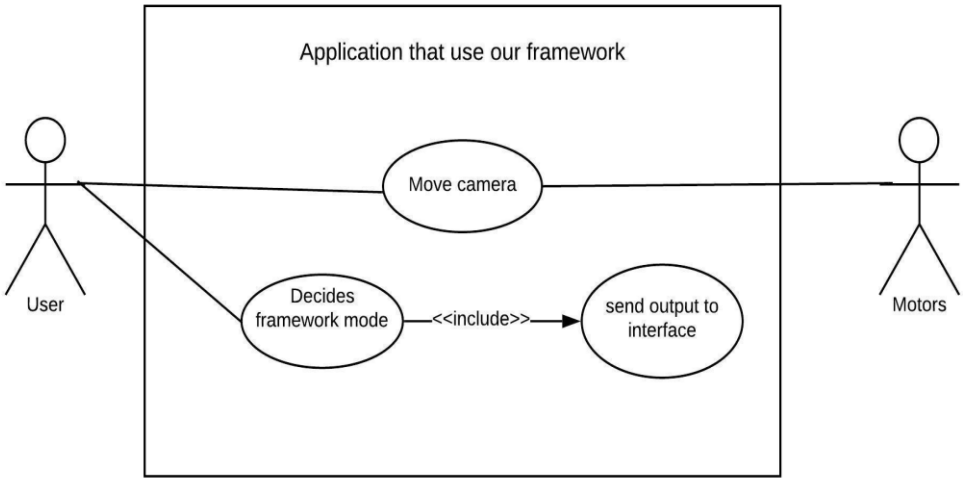


Figure 11-Use cases of the future Application (in manual mode)

3.4.3. State Chart Diagrams

On the state chart diagram bellow (figure 12), we seek to show the general states on the framework level.

At first the framework is on Idle (inactive or not working at a job). When the application uses our framework, putting the parameters, the system verifies them. Our framework must be capable to make two different types of processing: face detection (to search and detect a human face) and face recognition (recognize a specific person, already saved in memory). They are applied (separately) when the image is capture by the camera. Then, after this process our framework will provide the application to use the image that was processed.

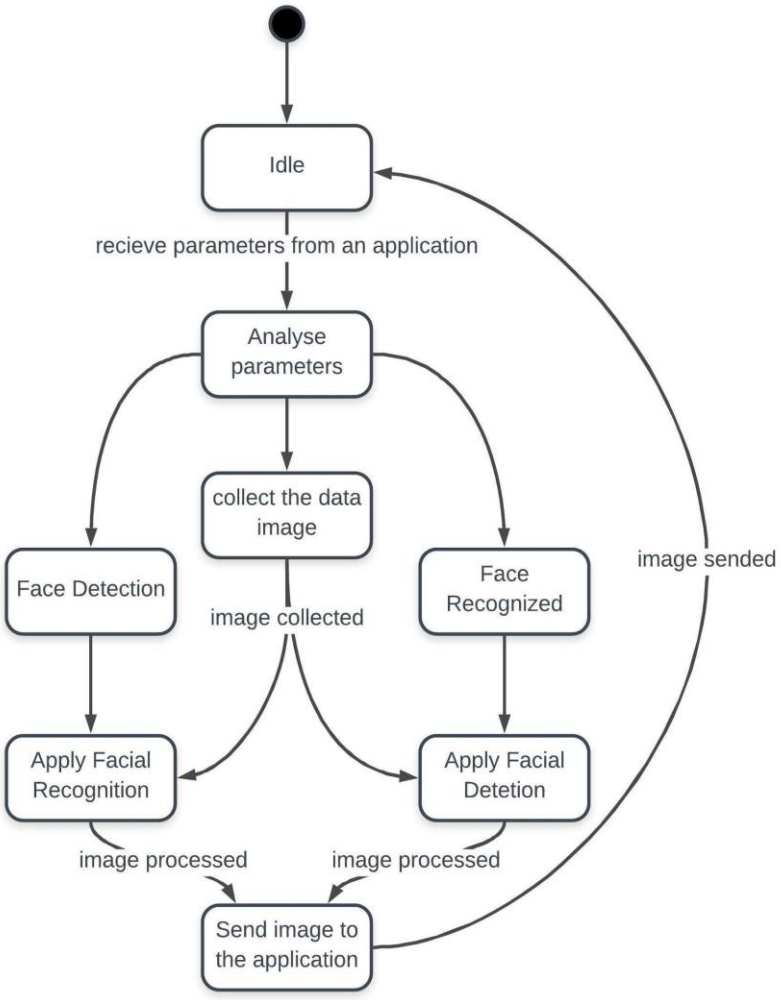


Figure 12- State chart diagram of framework

In the diagram below (figure 13) is illustrated the states chart for the application interface. The first procedure of our programs is the idle, where no interaction occurs. In comparison, this diagram got some different states from the above one, in which the interface:

- Waits for the user choices. First our interface must be capable to display to the user an initial menu that allows him to choose two different operation modes.

- Must be capable to show the processed image when received from the framework.

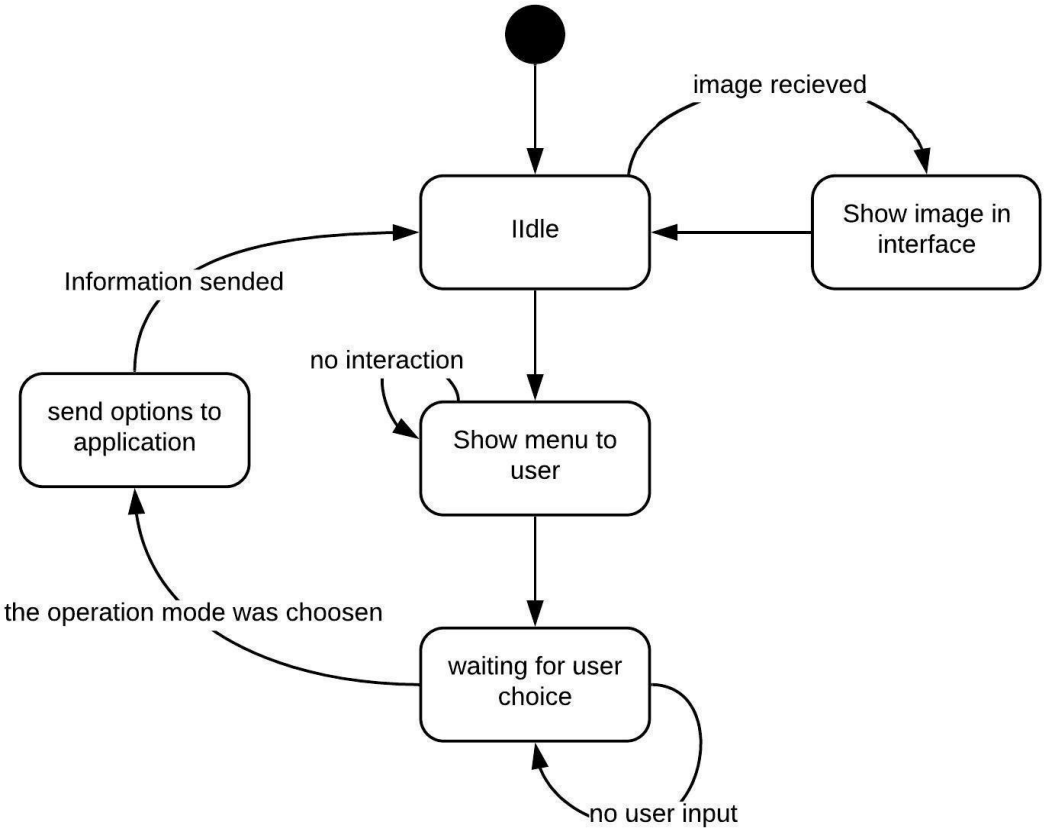


Figure 13- State chart Diagram of the application Interface

At last we represent the application state diagram. The application will follow people faces, centering the face on the middle of the image.

The application will activate on the servo-motors, moving the camera in relation to the analyzed position of the person on the image. So, the application takes over our framework to get all of information about the image. At the end it sends the analyzed position data to the interface.

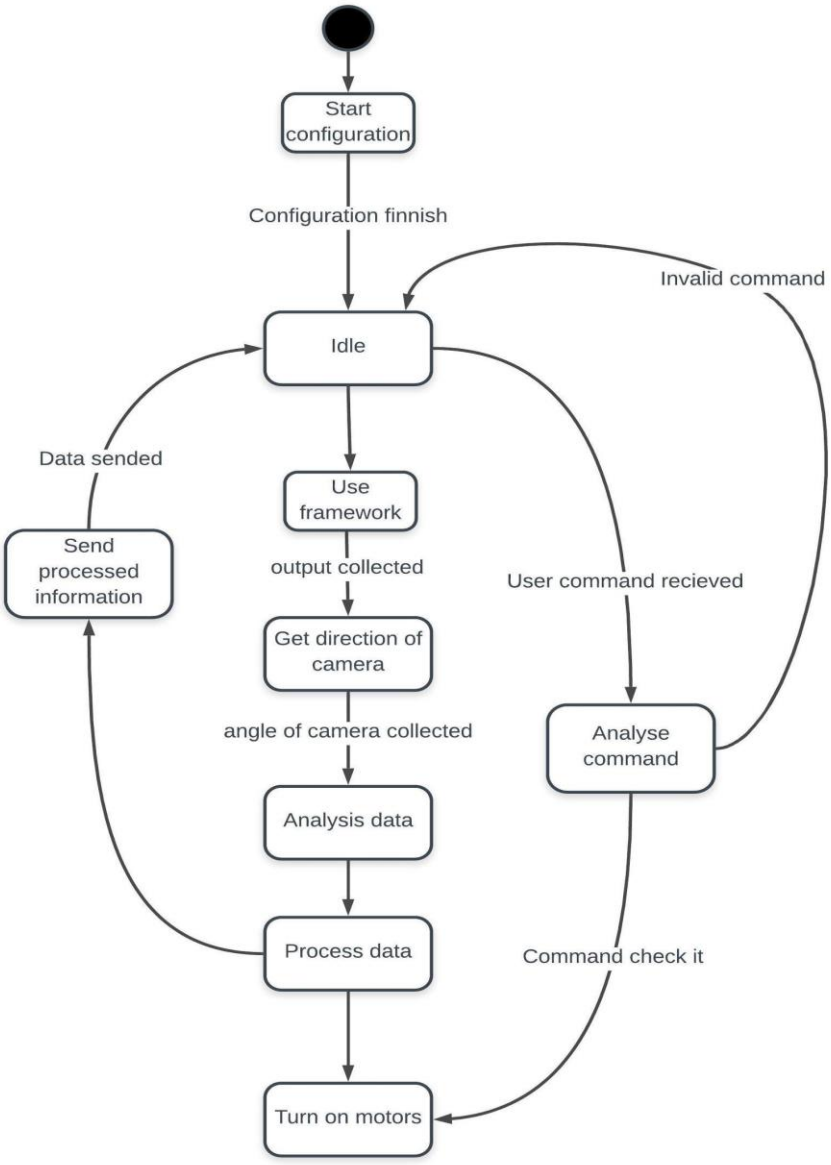


Figure 14- State Chart diagram of the future application

3.4.4. Sequence Diagram

The sequence diagram mentions the order of events in the systems. The diagram bellow is based on the framework sequence of events.

The first event (asynchronous), happens when the user inputs the operation mode of application. As mentioned before, the user is allowed to choose how the application will be working, (manual or automatic mode). If the event is automatic the application calls our framework to collect and analyze an image. In the other way, if it is manual the application will wait for an input from user to control the camera.

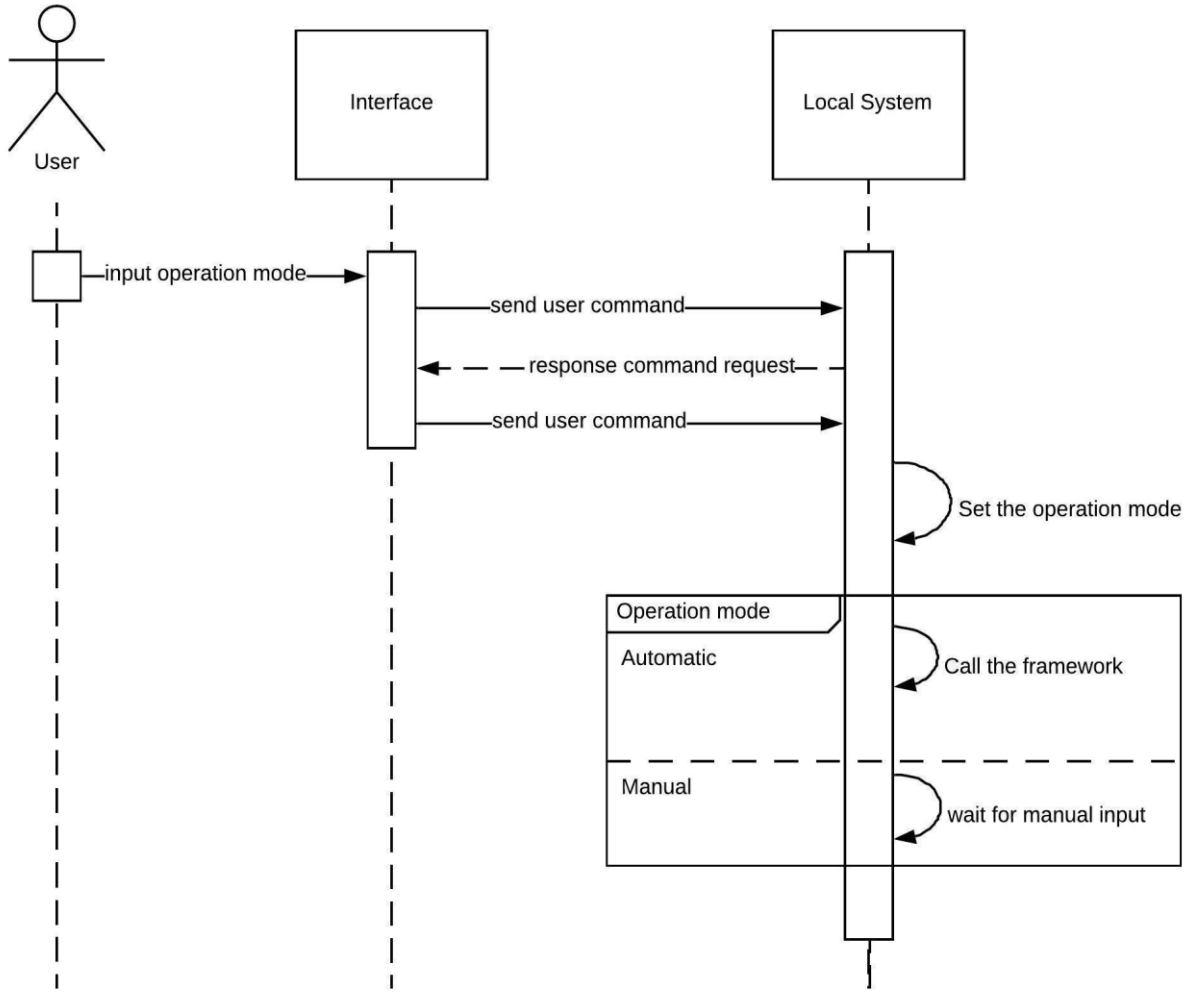


Figure 15- Sequence diagram of input operation mode event

So, if the user wants to control the direction of camera, he just needs to interact with the interface on manual mode.

He inputs a command and the application set the angles of the Motors.

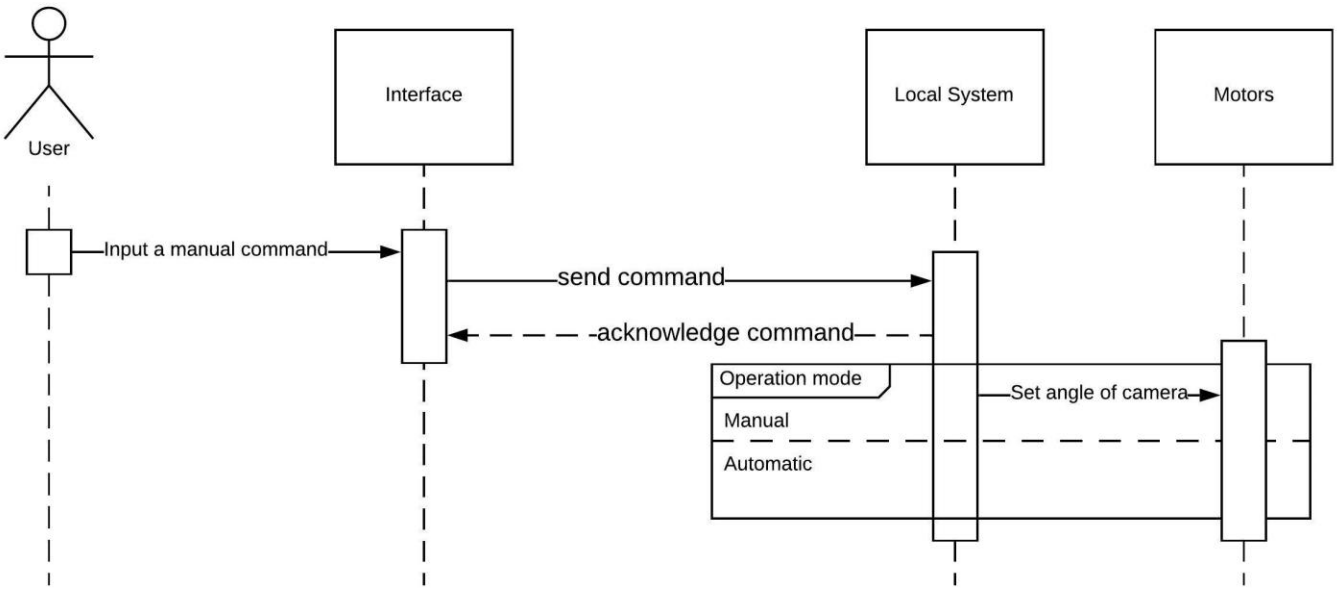


Figure 16- Sequence diagram of input manual command event

In case the User defines the operation mode as automatic, the actor running our framework is the "Application".

Our framework must be capable to access the camera and collect some environment images to analyze, and additionally it must be capable to prepare the image for any application that intends use them.

After our framework is called, it will check the application's parameters and an image will be captured from camera.

Next, our framework will run according to the User choice, through "face detection" or "face recognition".

On "face detection", the system will try to find a face, and if detected our framework will collect all of information about the image and its position and prepare to send the output. However, if was not detected, the framework will save the information that a face was not detected and does not alter the image.

On "face recognition", the process is the same, but the framework is going to apply the facial detection too, making our program with a more advanced feature.

Finally, the framework will process the image.

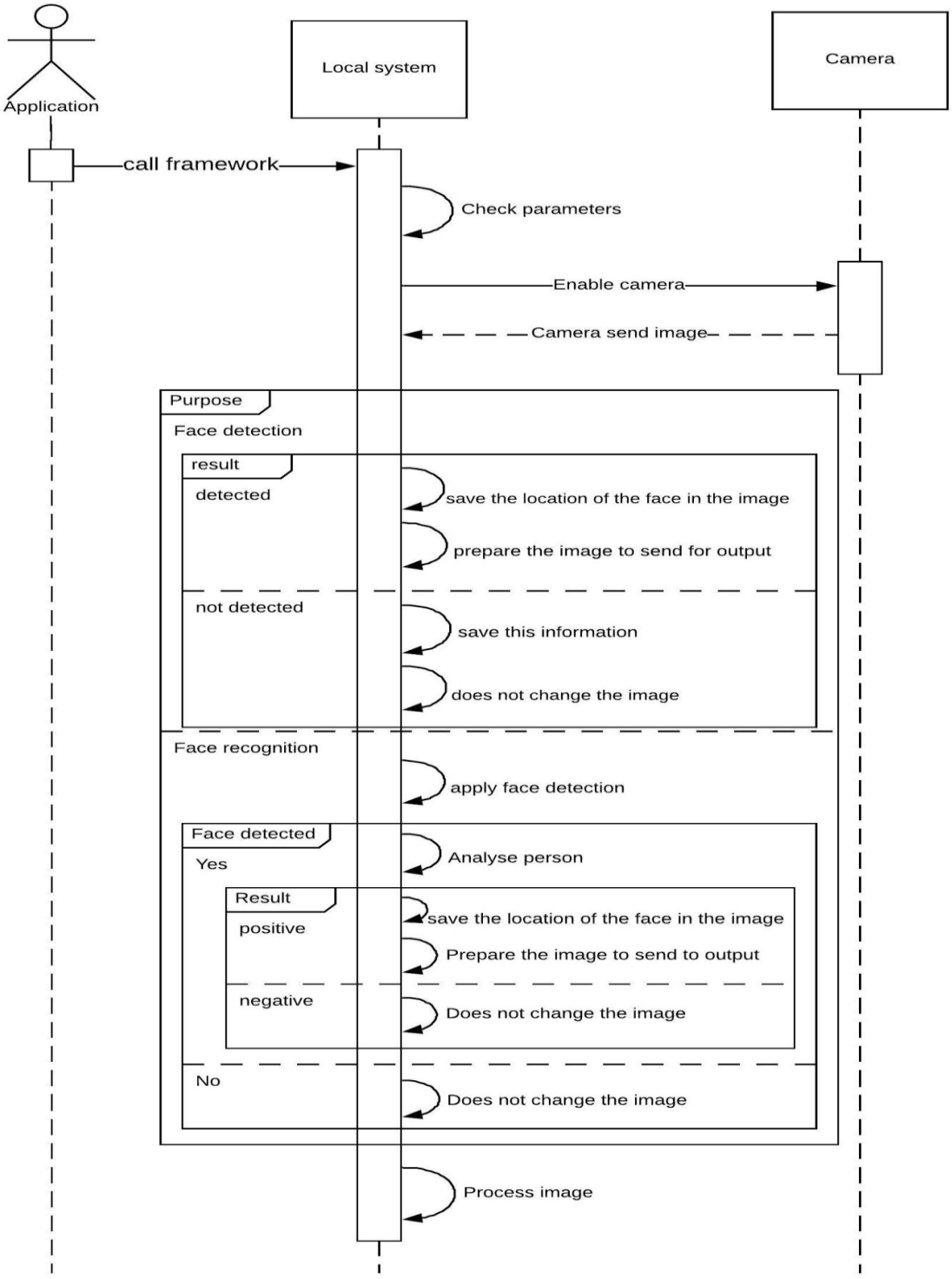


Figure 17- Sequence diagram of framework call event

To end the sequence diagrams its explained bellow the automatic mode.

When the time triggers a sample period and there is a processed image, the application will collect the output of the framework, establish communication with interface, analyze the data and send all necessary data to the interface, adjusting the angle of camera with the servo-motors.

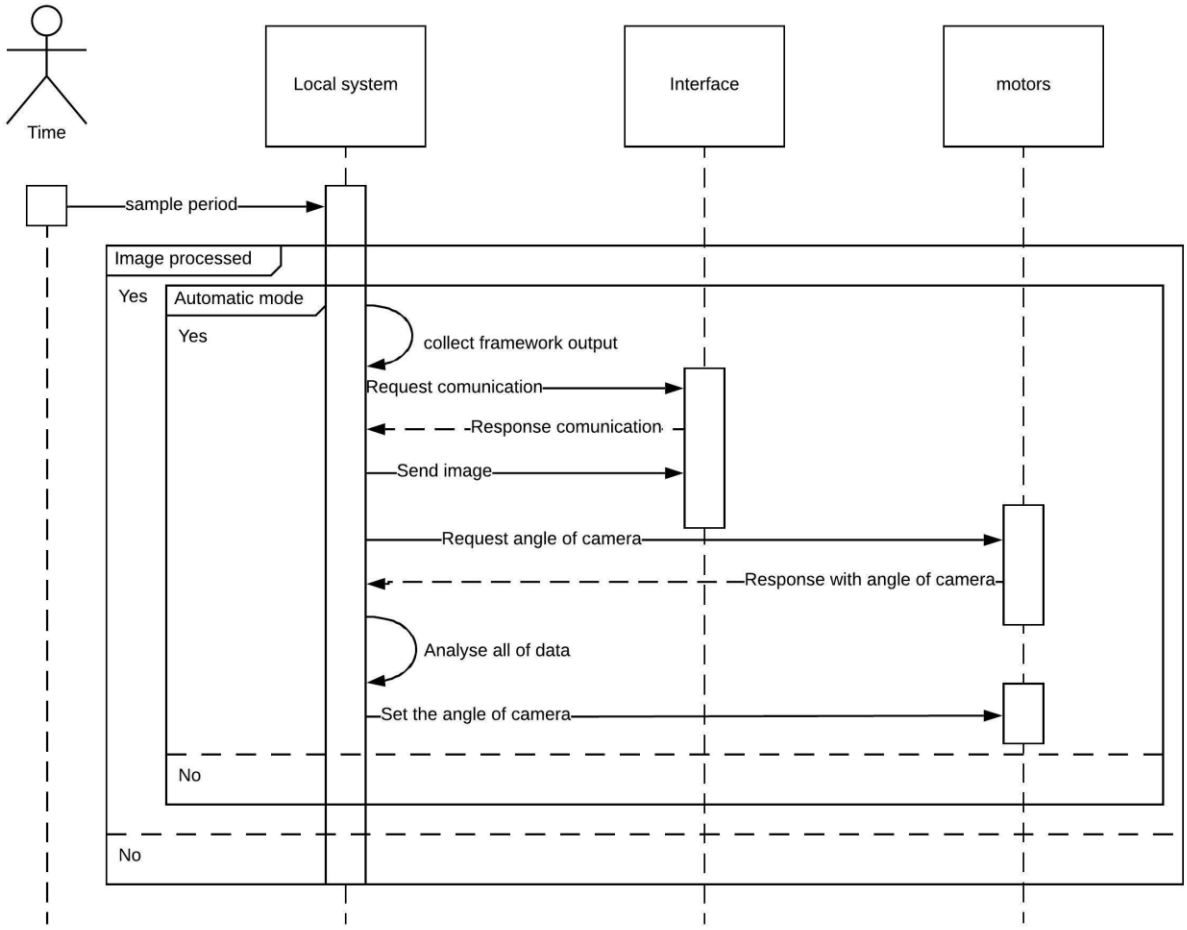


Figure 18- Sequence diagram of sample period event

4. Design Phase

On this phase our project we decrypt the fundamentals that was described on last phase, Analysis phase. We intend project the product in detail.

At first, we going to introduce a several base on theoretical concepts that are important to understand some contents covered on this project.

With the learned subject of class, we intend to meticulously cover all of software and hardware topics and respect its features.

4.1. Theoretical Concepts

4.1.1. Device Drivers

A device driver is a set of function and data that operates or control a particular type of device. A driver provides a software interface to hardware devices enabling operating systems and other computer programs to access hardware functions without needing to know the precise details of the hardware being used.

To interaction with hardware the user is provided by a several of functions. These functions will be used to allow data transfer between kernel space and user space.

4.1.2. Processes and threads

Events	User Functions	Kernel Functions
Load a module	insmod	module_init( )
Open a device	fopen ( )	file_operations: open
Read from a device	fread( )	file_operations: read
Write to a device	fwrite( )	file_operations: write
Close a device	fclose( )	file_operations: release

Table 3- Procedures and threads functions

In terms of definition, a process is an executing program. A set of process that are being executed is called by multiprocessing or multitasking. They can be executed at the same time or not, also noun as "parallel" execution.

There are three models to describe multitasking:

- Round-robin model: when a task is executed to the end, and after that another task is initiated;
- Round-robin with time-slice: each task got a period to run;
- Preemptive model: each task has its priority and are executed by their priority order;

With those models, the tasks have different types of states:

- Run -The task is currently being executed;
- Wait – Task in blocked state waiting for specific resource or event;
- Stop – Task in blocked, usually used for debugging;
- Zombie – Task in blocked and no longer needed;

Some multitasking OS also provide threads (lightweight processes) as an additional, alternative means for encapsulating an instance of program.
The task can own one or more threads.
A thread is a sequential execution stream within its task. Unlike tasks, which have their own independent memory spaces that are inaccessible to other tasks.

A process distinguishes from thread in these topics:

- As we refer before, the tasks have their own independent memory spaces that are inaccessible to other tasks;
- Threads can live entirely in user space, so that no kernel mode switch needs to be made to create a new thread;
- Threads have less overhead than processes;

4.1.3 Signals

Signals are indicators to a task than an asynchronous event has been generated by some external event or some internal event (problems with instructions being executed among others).

When a task receives a signal, it suspends executing the current instruction stream and context switch to a sign handler. The signal handler is typical executed within the task´s context.

The purpose of signals is typically being used for interrupting in an OS, because of their synchronous nature. However other purpose of signals is the intertrack communication mechanisms like shared memory, message queues and so on.

4.1.4 Detecting faces

Face detection is a computer technology that determines the locations and sizes of human faces in images ignoring the rest of the image. There are many techniques to detect human faces. To implement our framework, we will use the OpenCV.

To detect faces we need to fulfill the following steps:

- Import Image: to import the image we need to know the location of the image;
- RGB to Grayscale: puts the image in monochromatic. It's an important step because it facilitates the detection of faces;
- Image manipulation: this step includes resizing, cropping, blurring and sharpening of the images;
- Image Segmentation: this step is to found multiple faces on a single image;
- Haar-Like features algorithm: this step informs where the faces (or other things that belong to faces) are located on image;
- Surround face: this step makes a rectangle box in the picture to show the region of interest in the image.

With all of these steps we can detect faces in a image.

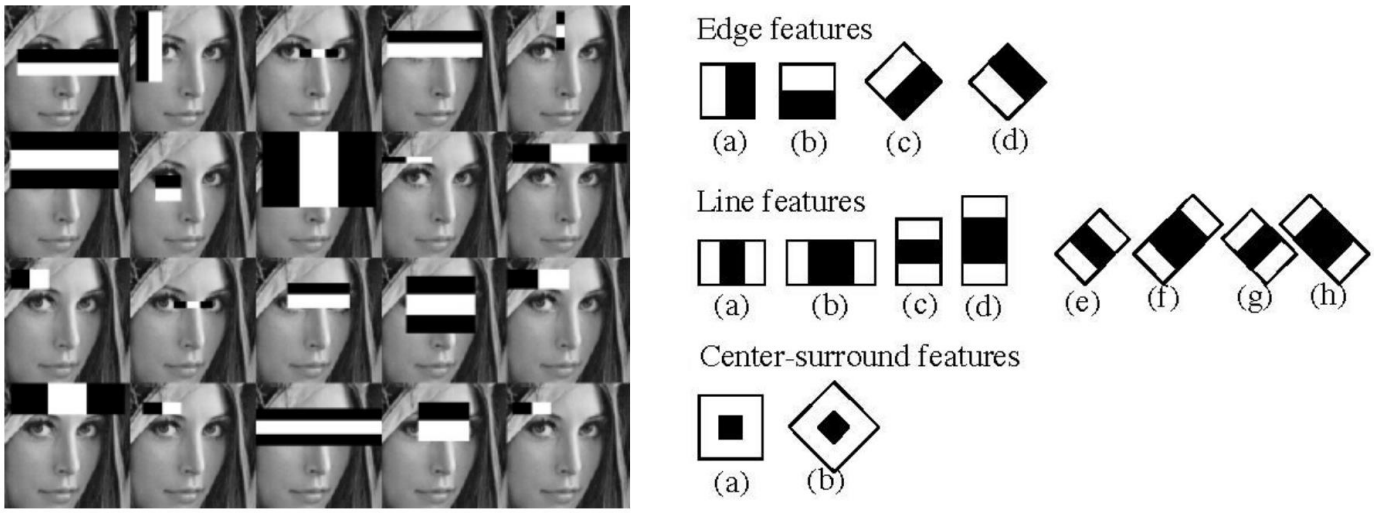


Figure 19- Features of haar-Like algorithm

The "haar-like" algorithm is described by the edge features, line features and center-surround features, through them it identifies all the face´s parts.

The algorithm selects the black and white image and converts the shadows part to black and converts bright spots to white. For instance, when the algorithm localizes a nose, it localizes a line feature through the brightness and the shadows (shadow represent bit=1, brightness represent bit=0).

Figure 20- Example of line feature in a nose.

4.1.5 Recognizes faces

While face detection only detects the presence of a face, A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source.

Automatic face recognition is all about extracting those meaningful features from an image, putting them into a useful representation and performing some kind of classification on them. Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. Features like marker points (position of eyes, ears, nose, ...) were used to build a feature vector (distance between the points, angle between them, ...).

The OpenCV library allows us to create our own Classifier which can be used to detect any other object in an Image by Training your Cascade Classifier.

Here the currently available algorithms are and their OpenCV calls:

- **EigenFaces** – cv2.face.createEigenFaceRecognizer()
- **FisherFaces** – cv2.face.createFisherFaceRecognizer()
- **Local Binary Patterns Histograms (LBPH)** – cv2.face.createLBPHFaceRecognizer()

Eigenfaces face recognizer:

This algorithm considers the fact that not all parts of a face are equally important or useful for face recognition. Indeed, when you look at someone, you recognize that person by his distinct features, like the eyes, nose, cheeks or forehead; and how they vary respect to each other.

This is how EigenFaces recognizer works, it catch the maximum variation among faces and help us to differentiate one face from other. It looks at all the training images of all the people as a whole and tries to extract the components which are **relevant** and useful and discards the rest.

Below is an image showing the variance extracted from a list of faces. Thus, whenever you introduce a new image to the algorithm, it repeats the same process as follows:

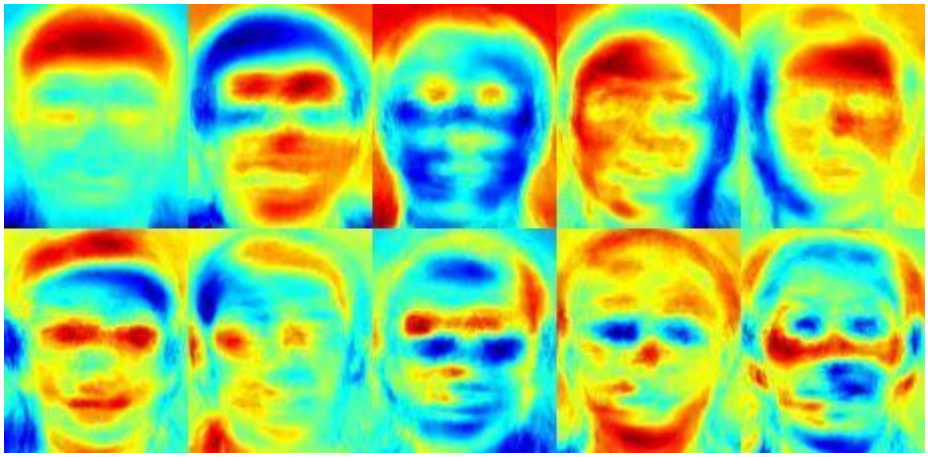


Figure 21- Eigenfaces algorithm example

1. Extract the principal components from the new picture.
2. Compare those features with the list of elements stored during training.
3. Find the ones with the best match.
4. Return the 'person' label associated with that best match component.

In simple words, it's a game of matching.

However, one thing to note in above image is that EigenFaces algorithm also considers illumination as an important feature. In consequence, lights and shadows are picked up by EigenFaces, which classifies them as representing a 'face.'

Fisherfaces face recognizer:

This algorithm is an improved version of the last one. Since EigenFaces also finds illumination as a useful component, it will find this variation very relevant for face recognition and may discard the features of the other people's faces, considering them less useful. In the end, the variance that EigenFaces has extracted represents just one individual's facial features.

By tunning EigenFaces so that it extracts useful features from the faces of each person separately instead of extracting them from all the faces combined. In this way, even if one person has high illumination changes, it will not affect the other people's features extraction process.

Precisely, **FisherFaces** face recognizer algorithm extracts principal components that differentiate one person from the others. In that sense, an individual's components do not dominate (become more useful) over the others.

Below is an image of principal components using FisherFaces algorithm.

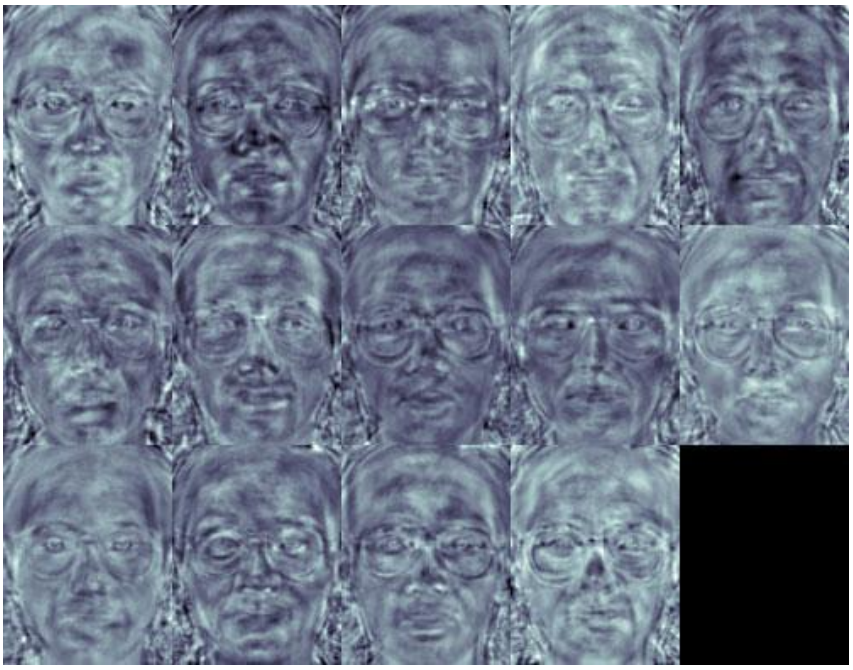


Figure 22- Fisherfaces algorithm example.

One thing to note here is that FisherFaces only prevents features of one person from becoming dominant, but it still considers illumination changes as a useful feature. We know that light variation is not a useful feature to extract as it is not part of the actual face.

Local binary patterns histograms (LBPH) Face Recognizer:

We know that Eigenfaces and Fisherfaces are both affected by light and, in real life, we can't guarantee perfect light conditions. *LBPH face recognizer is an improvement to overcome this drawback*. The idea with **LBPH** is not to look at the image as a whole, but instead, try to find its local structure by comparing each pixel to the neighboring pixels.

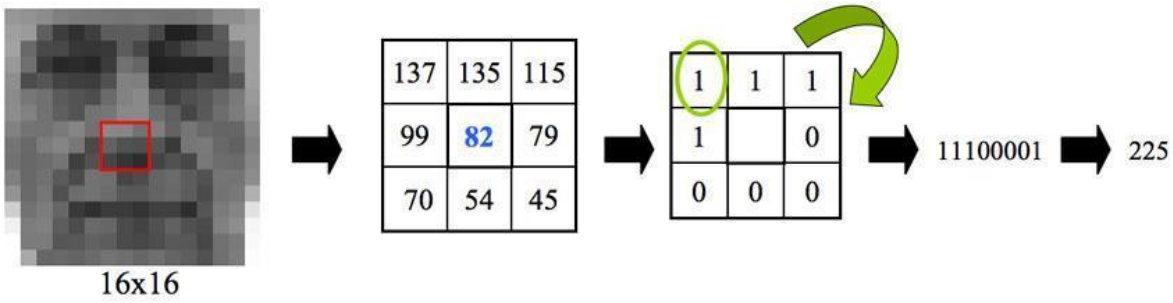


Figure 23- LBPH illustration

Now, after you get a list of local binary patterns, you convert each one into a decimal number using binary to decimal conversion (as shown in above image) and then you make a histogram of all of those decimal values. A sample histogram looks like this:

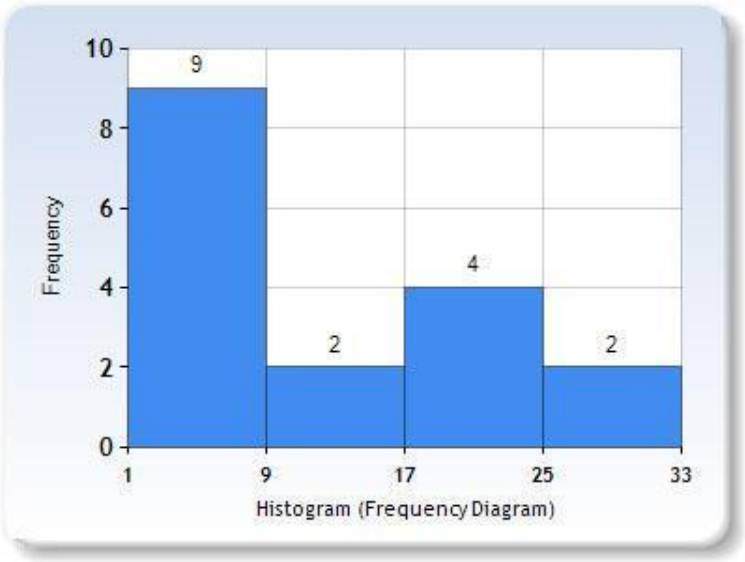


Figure 24- Graphic of probably LBPH method

Later during recognition, the process is as follows:

1. Feed a new image to the recognizer for face recognition.
2. The recognizer generates a histogram for that new picture.
3. It then compares that histogram with the histograms it already has.
4. Finally, it finds the best match and returns the person label associated with that best match.

Below is a group of faces and their respective local binary patterns images. You can see that the LBP faces are not affected by variation of light conditions:

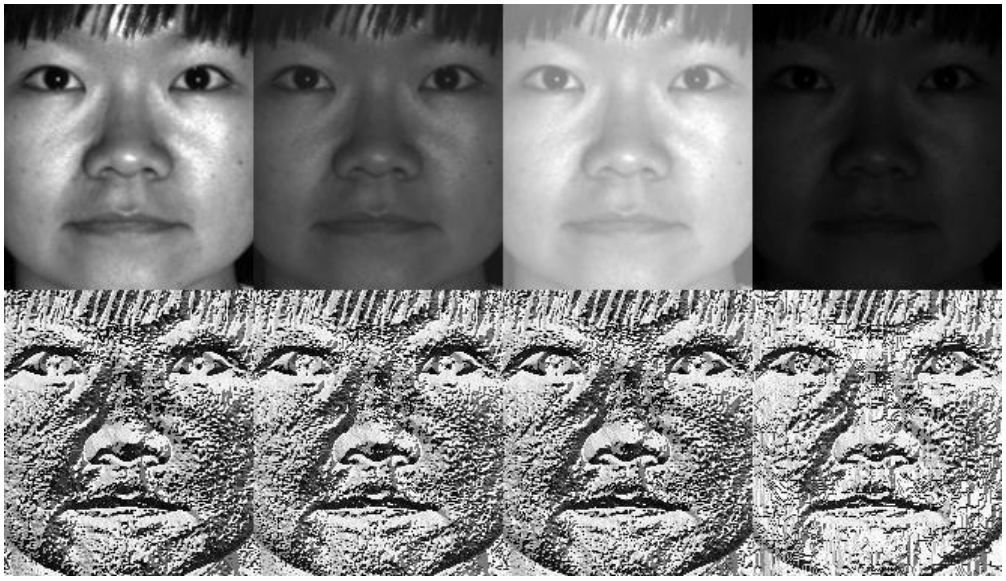


Figure 25- Result of LBPH algorithm in a image

4.2. Hardware Specification

4.2.1 Raspberry Pi 3 B+

The use of Raspberry Pi is a constraint for our project, therefore our board will be Raspberry Pi 3B+. This board will make our program more flexible, since can be added more functionalities to the application build.

The Raspberry Pi 3 Model B+ is the latest product in the Raspberry Pi 3.

Raspberry Pi3 Model B+ Specifications:

- Extend 40 Pin Gpio header
- Full size HDMI
- 3 USB 2.0 ports
- 2x SPI interface
- 2x I2C interface
- CSI camera port for connection a raspberry pi camera

- DSI display port for connection a raspberry pi touchscreen display
- Micro SD port for loading the operating system and storing data
- 5V/2.5A DC power input

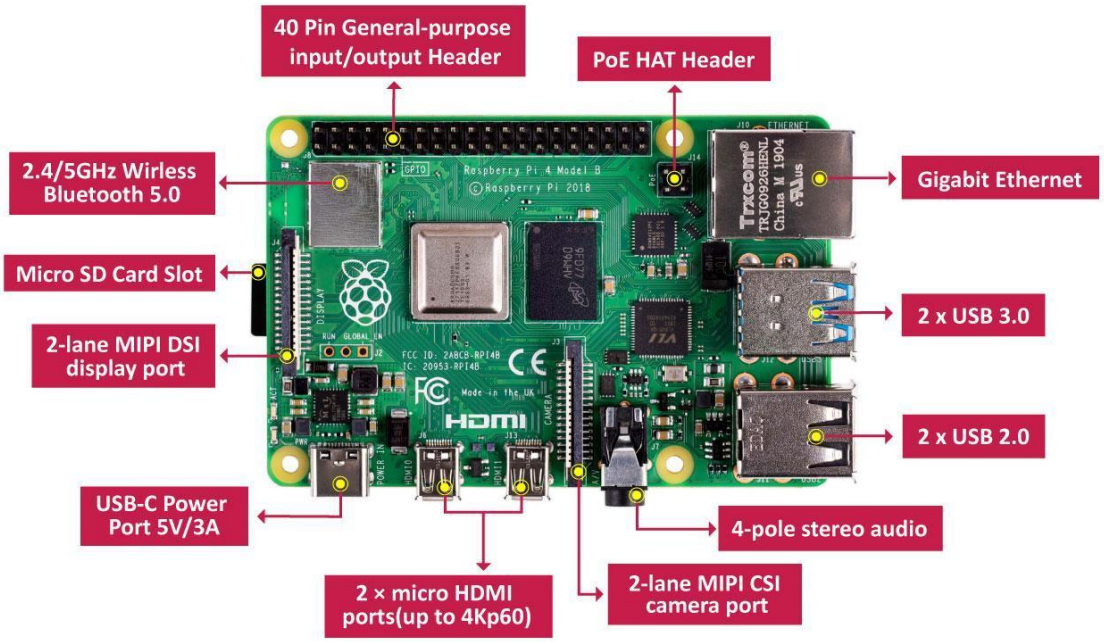


Figure 26 - Illustration and description of Raspberry Pi 3B+

4.2.2 Camera

In order to acquire images, a camera must be used and connected to the raspberry board. The Raspberry Pi Camera Board plugs directly into the CSI connector on the Raspberry Pi.

The module attaches to Raspberry Pi, by way of a 15 Pin Ribbon Cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI), which was designed especially for interfacing to cameras.

In our project the camera will be responsive for capturing images, and as we will apply the facial detecting or facial recognition. The capture must be high quality to improve the detection/recognition. However, the system must have a good performance and high-speed CPU. So, to do that balanced the parameters that we will focus are luminosity and angle of camera.

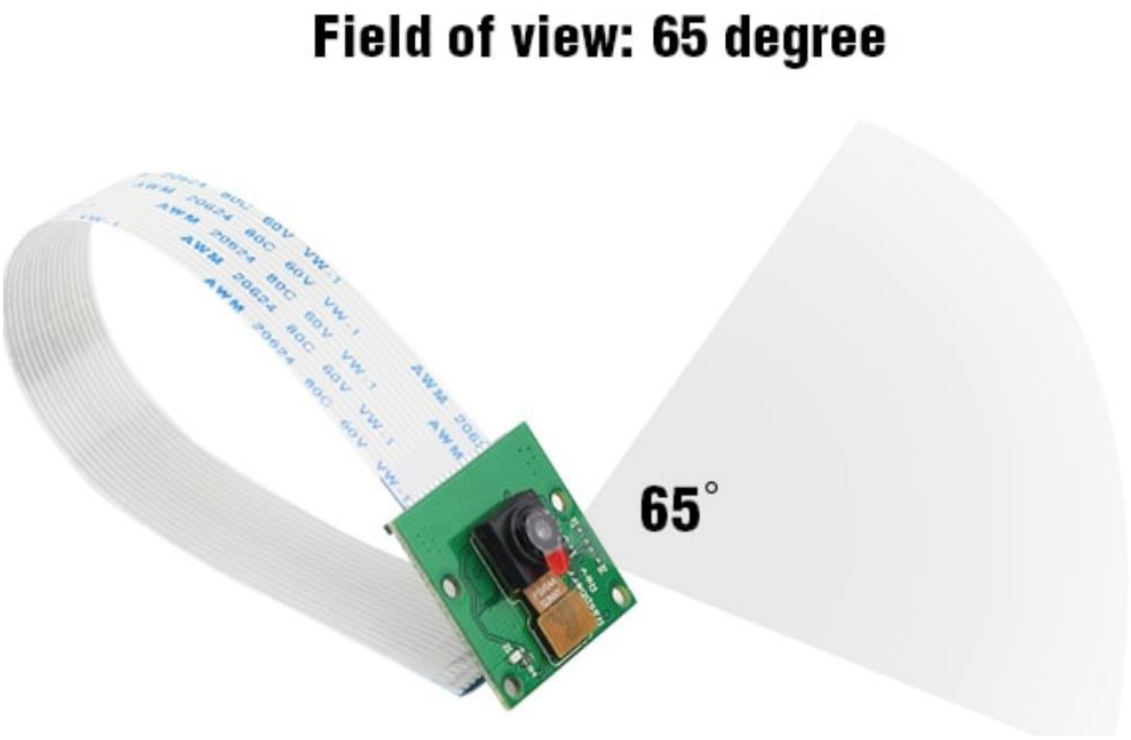


Figure 27- Pi Camera features.

Product Size (L x W x H): 17.00 x 2.50 x 0.20 cm
Product weight: 0.0050 kg

Test cases:
To have a good use of the camera it is necessary to be able to perform some tests. Such as, take a picture to see the resolution and other features of the image, capturing an image checking if was stored and test a video for streaming.

Test Cases	Expected Output	Real Output
Take a picture	An image is shown on screen;	
Capturing a image	Stored pictured that was captured;	
Test Video	Streaming a video;	

Table 4- Test Cases of Camera Pi

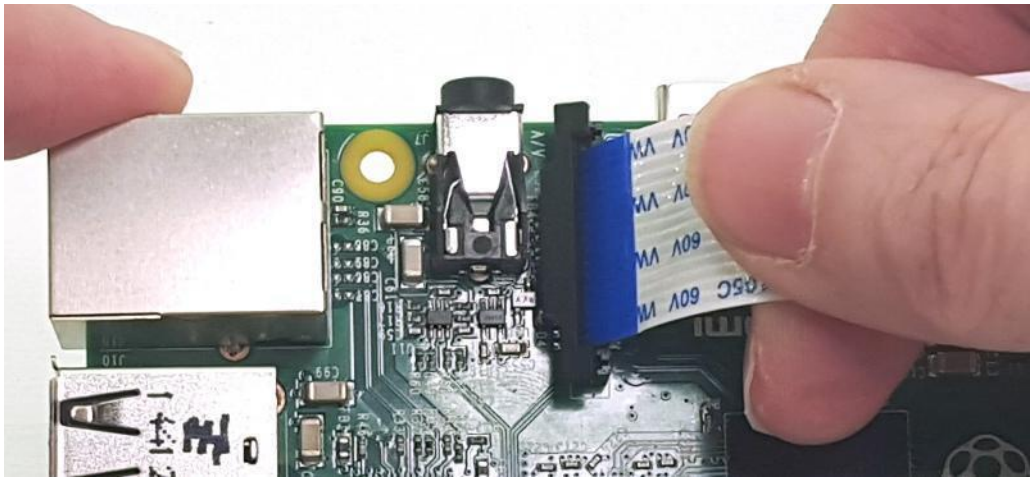


Figure 28- socket of pi camera on Raspberry Pi 3B+

4.2.3 Servo Motors

The 2 servo motors allow the angle adjustment of the camera in order to follow a person, rotating in the two axis (horizontal and vertical axis). The SONICMODELL Servo Motor is a tiny and lightweight motor with high output power, and has metal gears for added strength and durability.

The servo has three wires. Two are for power and the third wire is for signals to position the servo. The signal wire expects input from a pulse width modulator.

The period should be 20 milliseconds long and the duty cycle encodes the position of the motor. If the duty cycle is 1.5 millisecond, the servo is positioned at 0 degrees. If the duty cycle is 2 milliseconds, is all the way to the right (90 degrees). If the duty cycle is 1 millisecond, the servo is all the way to the left (-90 degrees).

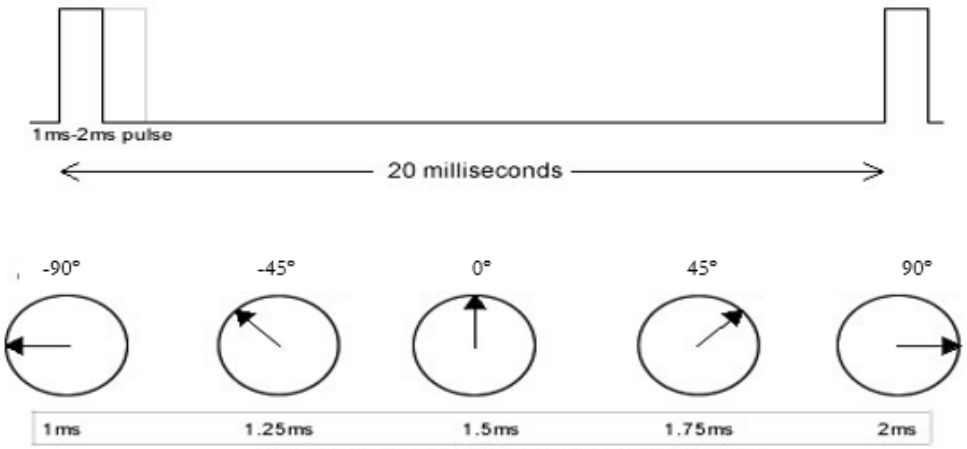


Figure 29- Example of how to control Servo Motors with PWM

Specifications:

- Weight: 18 g
- Dimension: 6x9x3 cm
- Stall torque: 1.8 kgF·cm (4.8V), 2.2 kgf·cm (6 V)
- Operating speed: 0.1 s/60 degree(4.8V), 0.08s/60 degree(6 V)
- Operating voltage: 4.8 V – 6.0V
- Dead band width: 5 µs

Figure 30- Servo Motors

Test Cases:
To have a good performance of the motors it is necessary to be able to perform some tests. Such as, initialization **PWM** to see the output waves, **driving motors to several directions** checking which duty-cycle percentage correspond to each angle.

Test Cases	Expected Output	Real Output
Initialization PWM	visualizing the wave;	
drive motors to several directions	input different duty-cycles of PWM	

Table 5- Test Cases of Servo Motors

4.2.4 AC/DC converter

The AC-DC converter will power the motors and the raspberry pi. This is a simple AC-DC converter that has an alternate current (AC)input tension of 230 V from the power grid and a direct current (DC) output tension of 5V. The AC-DC that we will use has the following characteristics:

- Input voltage: 100-240Vac and between 50/60Hz;
- Output voltage: 5Vdc voltage and capacity up to 2500mAh;
- Protection against overload and over voltage.

Figure 31- Example of Power Supply

4.2.5 Structure

The structure of the application we based on a Pan-Tilt model, where the two servo-motors will be coupled. It will enable the orientation of camera in 2 axis, necessary for the face centering of a person (aiming the camera in the person direction)

Figure 32- Tilt and Pan Mechanism

Test Cases:

Test Cases	Expected Output	Real Output
Test mechanism	visualize the movements	
Test Tilt	input PWM in tilt servo-motor	
Test Pan	input PWM in pan servo-motor	

Table 6- Test cases for structure

Pin Specification:

The communication can be performed by i2c. However, it's possible to use the individual pins to control the motors individually.

Pinout	Definition
Gnd	Ground, in the context of electronics, is the reference point for all signals or a common path in an electrical circuit where all of the voltages can be measured from.
5V	Power voltage
S1	S1, is the pin that control the angle of the servos through the PWM input.

Table 7- Pin Specification

Figure 33- Pinout of Pan-Tilt Hat

4.3. Hardware Peripherals and Communication Protocols

4.3.1 PWM Protocol

Pulse Width Modulation is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. PWM is be used to control the servo motors, due to the need of a duty cycle.

By controlling the time that signal spend in high state over a consistent interval, the percentage of time ON relative to the percentage of time to off, is called by duty-cycle.

The signal can only in the range of 5V(high) or ground(low).

4.3.2 CSI Protocol

The Camera Serial Interface (CSI) protocol is a standard interface between a camera and a host processor. The use of this protocol presents some advantages:

- The CSI protocol requires far less computing power on the host side than other interfaces (such as USB), due to the reason that the camera is directly connected to CPU, and the video is routed automatically to an onboard coprocessor leaving the CPU available for vision processing application tasks.
- Better overall system performance;
- Better vision results;

4.3.3 Prototype Peripherals and respective Protocols

These protocols are responsible for the interaction and the stability of the communication between the hardware and our board (Raspberry Pi). For different hardware, in our case, we have different protocols, for instance:

-PWM for control the angle of camera

-CSI for connection between camera Pi and the raspberry.

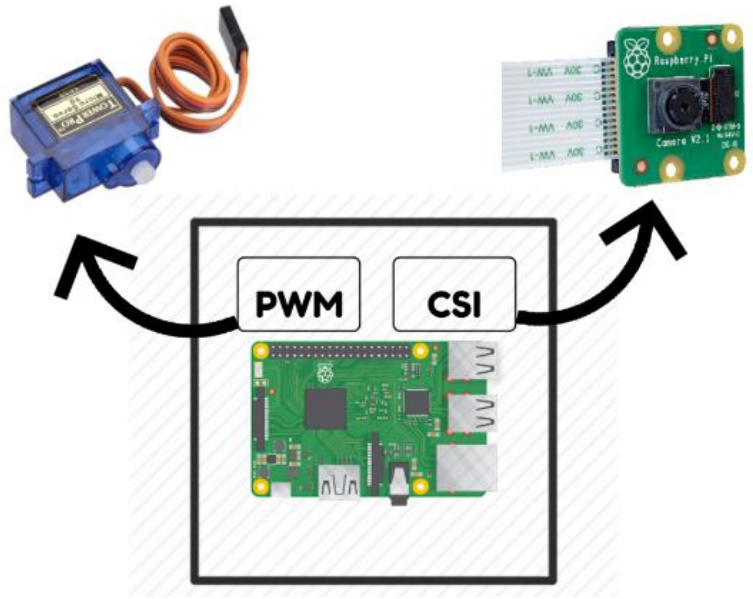


Figure 34- Prototype Peripherals and respective Protocols

4.4. Software Specification

4.4.1. Buildroot

Buildroot is a is a simple tool, a set of Makefiles and patches, used to generate embedded Linux systems through cross-compilation. Buildroot can generate any or all of a cross-compilation toolchain, a root filesystem, a kernel image and a bootloader image.
We will use Buildroot because it is able to generate a cross-compilation toolchain. It will provide a root filesystem, a Linux kernel image and a bootloader for our target which in our case is Raspberry PI 3B+.

A Cross Compiler is a compiler capable of creating executable code for a platform other than the one which the compiler is running. In our case, it is needed to compile code in C/C++ Language, located on our host (computer), to run in our Raspberry Pi 3B+.

Figure 35- buildroot icon

4.4.2. Qt Creator

Qt Creator is an integrated development environment (IDE) that provides the tools to design and develop applications with the Qt application framework. Qt is designed for developing applications and user interfaces once and deploying them to several desktop, embedded, and mobile operating systems. Qt Creator provides you with tools for accomplishing your tasks throughout the whole application development lifecycle, from creating a project to deploying the application to the target platforms.

Qt is written in C++ such as the interface, using Qt's Widgets module. Interaction with the interface is made by events. Qt also offers many classes and functions for working with threads. QThread class is the foundation of all thread control in Qt. Each QThread instance represents and controls one thread. QThread are

an abstraction layer from the POSIX Threads, meaning that thread in Qt can be implemented using native Pthreads.

Figure 36- Qt Creator icon

4.4.3. C/C++

"C is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations. By design, C provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, including operating systems, as well as various application software for computers ranging from supercomputers to embedded systems."

"C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation."

Wikipedia

4.4.4. Open CV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

Computer vision is the essential part of our project, so OpenCV library is indispensable. We use it to detect face and to recognize face using the algorithms that it provides.

Figure 37- Open Cv icon

4.4.6. Pthreads

Threads are a mechanism that permits an application to perform multiple tasks concurrently. A single process can contain multiple threads, executing the same program, and they all share the same global memory, including the initialized data, uninitialized data, and heap segments.

The threads in a process can execute concurrently. On a multiprocessor system, multiple threads can execute simultaneously, also known as parallel execution. Pthreads are defined as a set of C language programming types and procedure calls, implemented with a pthread header and a thread library.

Thread procedures can be categorized into four groups:

- Thread Management
- Mutexes
- Condition Variables
- Synchronization

Table 8- Pthreads icon

4.5.Task Overview

To achieve a proper functioning of our system we need to establish the threads communication and how they reach synchronization. Therefore, this diagram includes the types of synchronization between the threads and which are the Mutex, Condition Variable and Semaphore and the crossed information through the different processes (daemons and main process) which is Queue.

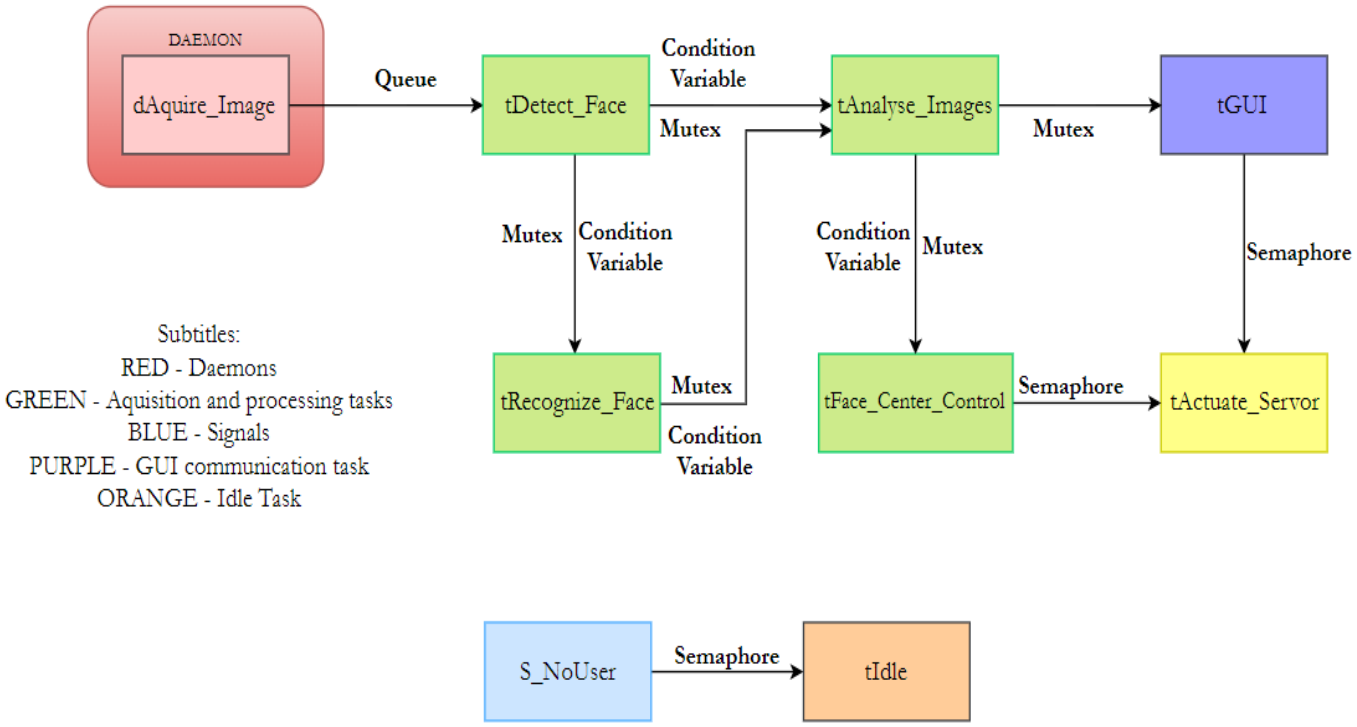


Figure 38- Task Overview Diagram

4.5.1. Processes:

As described before, on concepts a process is an executing program and in our project the instances of computer that will be executed are:

Main Process: is responsible for the initialization, for the image processing, for the management the direction of camera and for the shutdown.

Daemon Process: which is *dCapture_image*, that will be the responsible for the acquisition of camera image.

4.5.2. Daemons:

dCapture_image: this background process will always acquire images from PiCamera, providing more efficiency in our application;

4.5.3. Threads:

tFace_Detection: Will be the first interaction with image. Will detect if a face is in the image or not.

tFacial_Recognition: The *User* has a Facial Recognition option to choose included in the program. It is a thread that will be responsible to apply the algorithm to recognize a face in a image (The algorithm will be chosen in application phase. The algorithms are described in *Theoretical Concepts* chapter).

tAnalyse_Images: Will be responsible for the gathered information from the OpenCV mechanism´s, must be capable to determine the location of the person's face on the images, prepare the images to send to interface and the coordinates (and other data) to the *tFace_center_Control task*.

tFace_Center_Control: With the coordinates of the person face will apply the implemented control on the servo motors.

tActuate_Servo: Responsible for the actuation of motors. Will actuate on the motors based on the user interact with interface (manual operation mode) or on the calculated control from *tFaceCenterControl* (automatic operation mode).

tIDLE: Executed when the S_NoUser is triggered, putting the system in a low power idle state.

tGUI: Will provide the execution, management and update of the graphic interface.

4.5.4. SIGNALS:

S_NoUser: This signal will be trigger when no user was detected in long space of time.

4.5.5. Condition Variables

cod_Face_Detection: This condition variable will be the responsive to warn the *tFacial_Recognition* that a face was detected.

cod_Face_Processed: This condition variable will warn *tAnalyse_Images* that an algorithm of OpenCv was already applied on the image.

cod_Image_Analyzed: This condition variable will warn *tFace_Center_control* that a image was detected and is ready for centering control.

4.5.6 Task Synchronization:

Sem_ActivateServo: This semaphore unlocks the *t_Actuate_Servo,* allowing the actuation of servo-motors.

Sem_Disable: This semaphore unlocks the *t_idle,* putting the system on low power consumption idle state.

Mutex_Face_Detection: This mutex is responsible for image data protection, that is a shared resource between the *tFace_Detection* and *tFacial_Recognition.*

Mutex_Face_Processed: This mutex is responsible for image data protection, that is a shared resource between the *tFace_Detection* and *tAnalyse_Images* and through *tFacial_Recognition* and *tAnalyse_Images.*

Mutex_Face_Location: This mutex is responsible to protect the location of the human/humans faces on each image. This mutex will protect these resources between the *tAnalyse_Images* and *tFace_Center_Control.*

Mutex_UpdateInterface: This mutex provides blocking mechanism when Reading and writing to the GUI update object.

4.6. Task priority:

In a system with a great number of tasks, each with its purpose, it's necessary to prioritize them in order to achieve our goal.

Not every task is equally important, some more than others. It is necessary to identify what is the most important task at any moment and give those tasks more attention, energy, and time.

For our system we design these priority task pyramid diagram below.

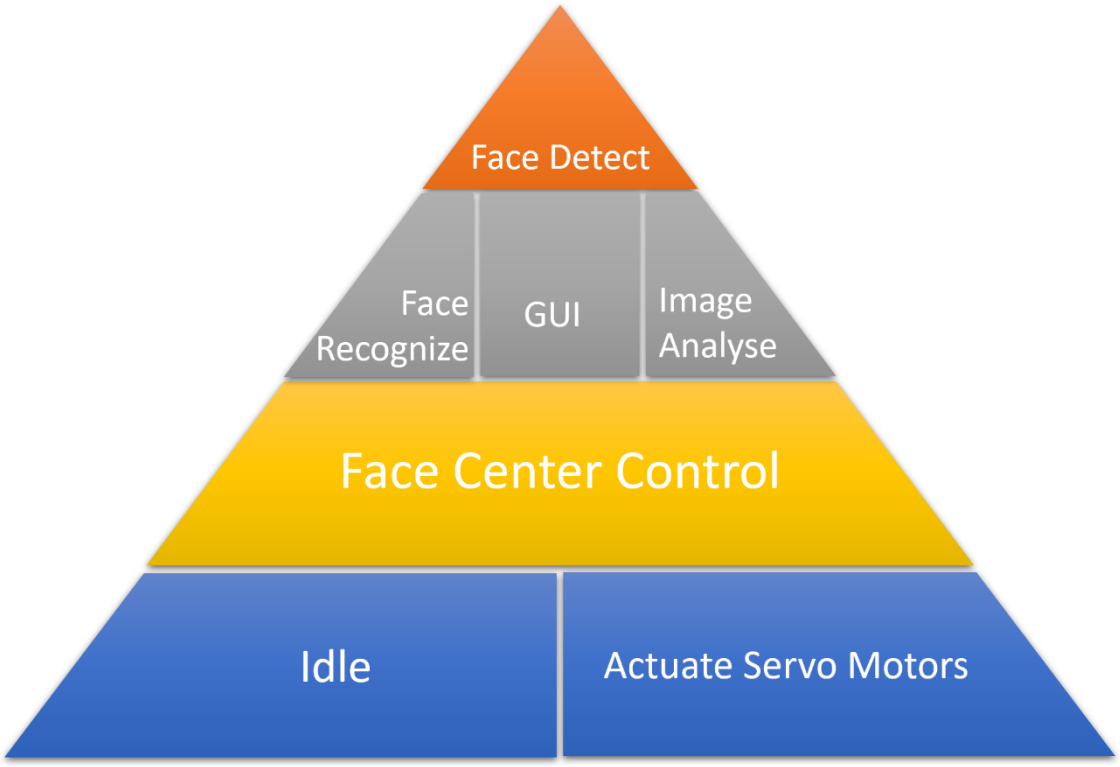


Figure 39- Task Priority Pyramid

In addition to task priority, we also have a process priority (the concept is identical). The processes don´t have equal importance, so, we must study our system and detect which is more important process and put it at top of pyramid diagram.

In the pyramid diagram below "image acquisition" was chosen as the most important process because without acquisition of image, our system doesn't have his main resource (images) and won't accomplish the purpose of our application. The lower layer is presented by Main process that will be responsive for the initialization, for the image processing, for the management the direction of camera and for the shutdown.

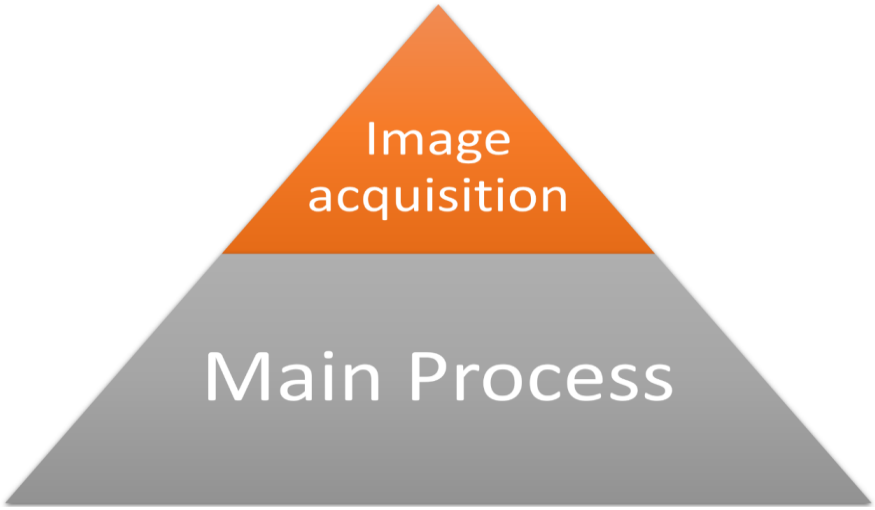


Figure 40- Processes Priority Pyramid

4.7. Flowcharts

4.7.1 System Initialization

The system starts initializing the two process: the Daemon and the Main process. On the daemon part, it first creates the daemon (with his standart initialization functions) and starts the daemon process, which is to capture the image from camera, as said before (and more detailed in the daemon flowchart section). The main process runs and inserts the device drivers, creates the task of the system and inits the scheduler.

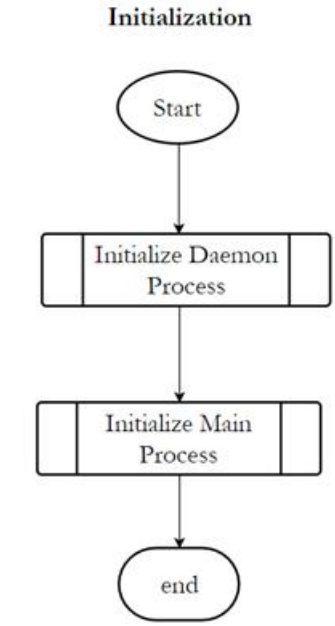


Figure 41- Flowchart of initializa

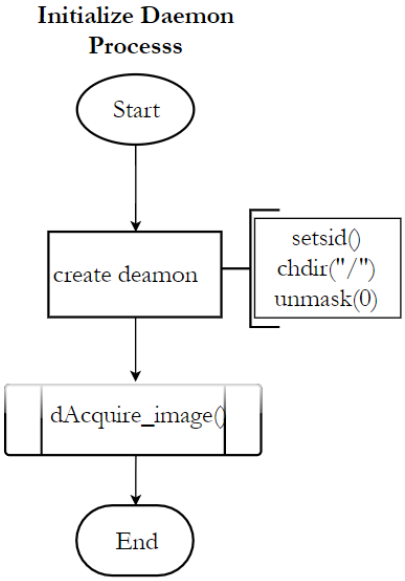


Figure 42- Flowchart of initialization of daemon

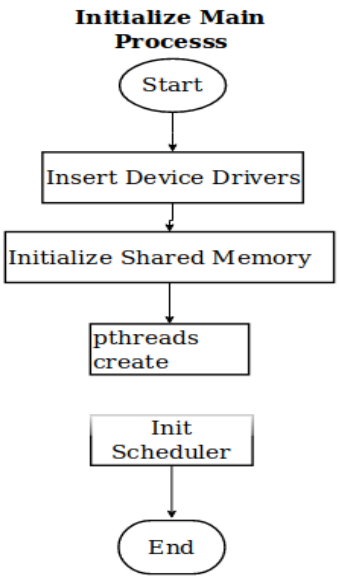


Figure 43- Flowchart of initialization of main process

4.7.2 Daemon

The deamon (*dAcquire_image*) runs in the background focused on making the acquisition of the camera image. It starts on a condition checking if a Image Queue message was sent for a capture image. If yes, it opens the camera (*Open_Camera()*) and effect the image capture (*Capture_Frame()*). After capturing it sends a queue message, warning the capture of a image.

The *Open_Camera()* function prepares the camera for the capturing, opening the camera through *cap(0)* function and clearing the last frame captured.

The *Capture_fram()* prepares a Cap type frame to storage the capture. If it capture a image with success (*cap.read(frame)*), it saves it on the shared memory (*shm_write()*).

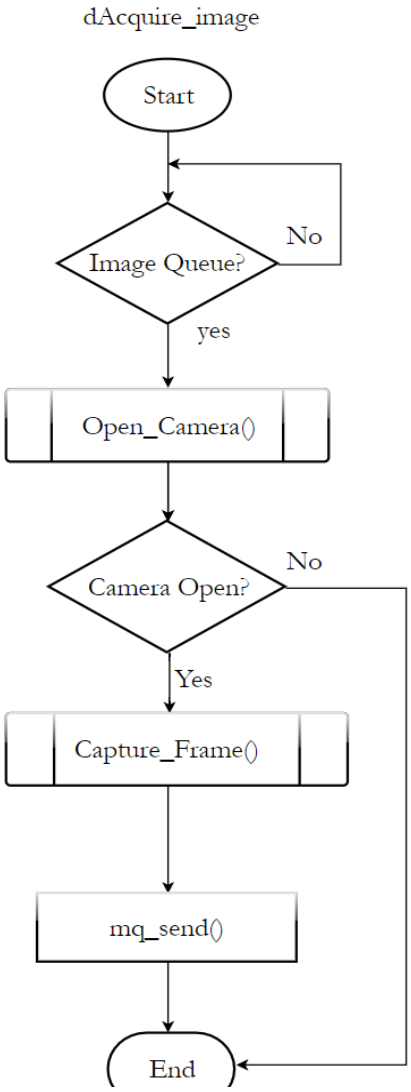


Figure 46- Flowchart of daemon

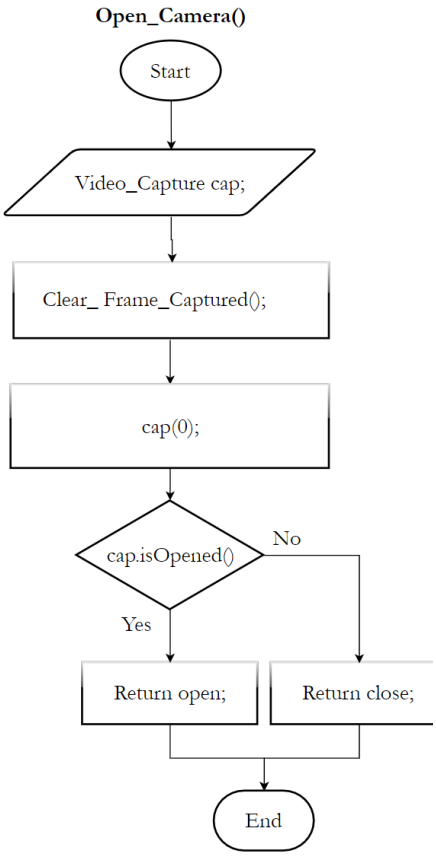


Figure 45- Flowchart of Open Camera

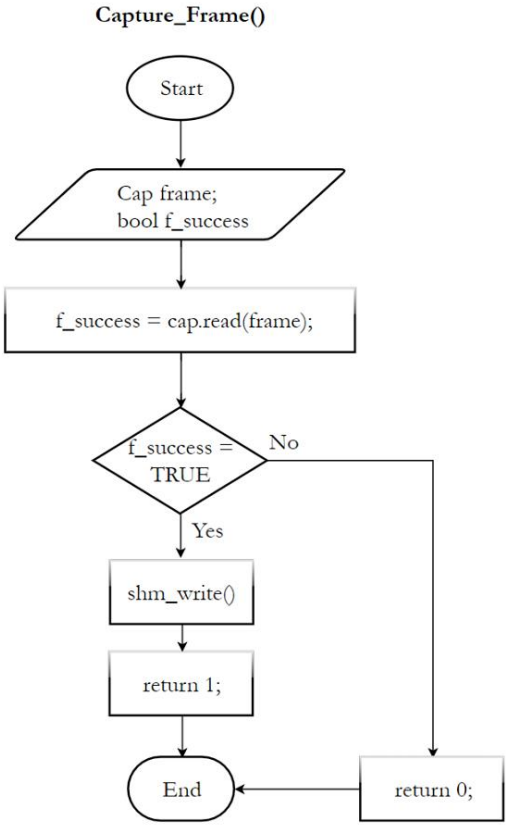


Figure 44- Flowchart of capture a frame

4.7.3 Tasks Flowcharts

tDetect_Face

The flowchart presented has a more detailed description of how the *tDetect_Face* is architected. As said before, it will get the captured image and try to find a face.

First it initializes the necessary variables, opens the Mutex to limit other threads access and awaits for a message in queue from the daemon to notify that there are capture images. If received, and if the memory not empty (*shm_empy()*) it gets the image from the shared memory and saves it to the *Ref_Frame* variable, before closing the mutex. Then the function *evtColor()* converts the image to monochromatic, and with *detectFaces(),* saves to *Faces* any detected face (both of the function are from OpenCV API). Then if any face is saved, *face_empty* not true, it checks the recognition mode (*rec_mode_f),* if it's to apply recognition or not. If yes, it stores the face in a vector.and prepares the conditional variable for the Face_recogntion task. Else, prepares the conditional variable for the Analyze_Image task.

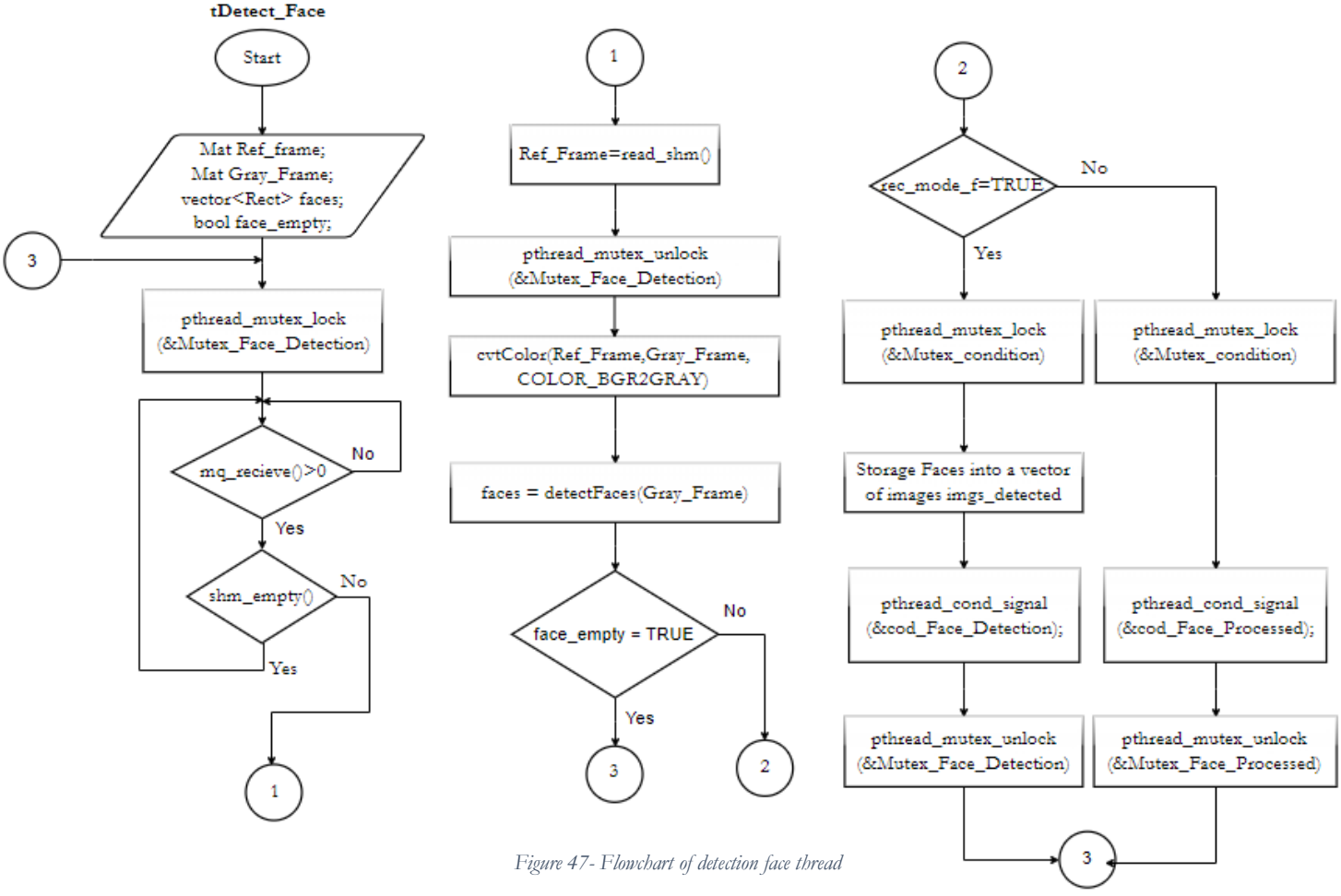
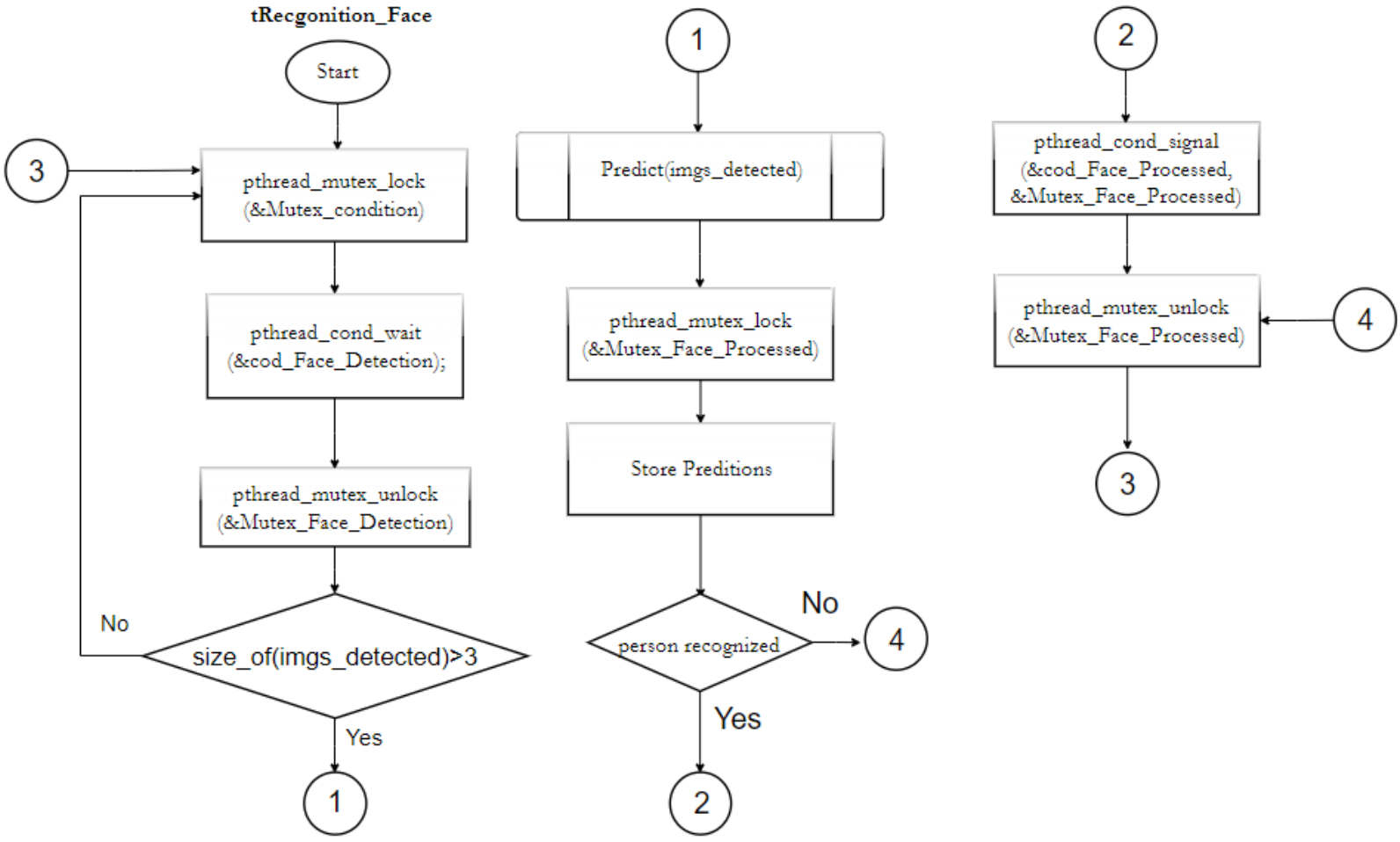


Figure 47- Flowchart of detection face thread

tRecognize_face

In facial recognition, is applied an OpenCV algorithm to calculate prediction value of the set of images captured. If the prediction value represents a strong prediction (higher than an stipulated reference value), it will consider that the faces match.

First, the task waits for the condition variable (*cod_Face_Detection)* triggered by the detection task (*tDetect_Face)*. When a signal of *cod_Face_Detection* is triggered, it verifies if size of images in vector *imgs_detected* is more than three, returning to the beginning if not. Then, calculates the prediction value with *Predict(img_detected)* function (defined below), storages the value and checks its value. If it is high enough to be consider a match. If so, the task proceeds and signals the variable condition *cod_Face_Processed.*



The Prediction of images is the most important task for face recognition. Is responsible to calculate and return the values of the confidence between the images that were saved (and trained) and the images that were collected.

The first step is to crop the images that were detected (imgs_detected) into a vector of faces (vec_of_faces). Next, one by one, the algorithm is applied to each cropped face and a prediction and a confidence is calculated for these images. Finally, is returned the values of confidence and prediction that will be used in tRecognize_Face.

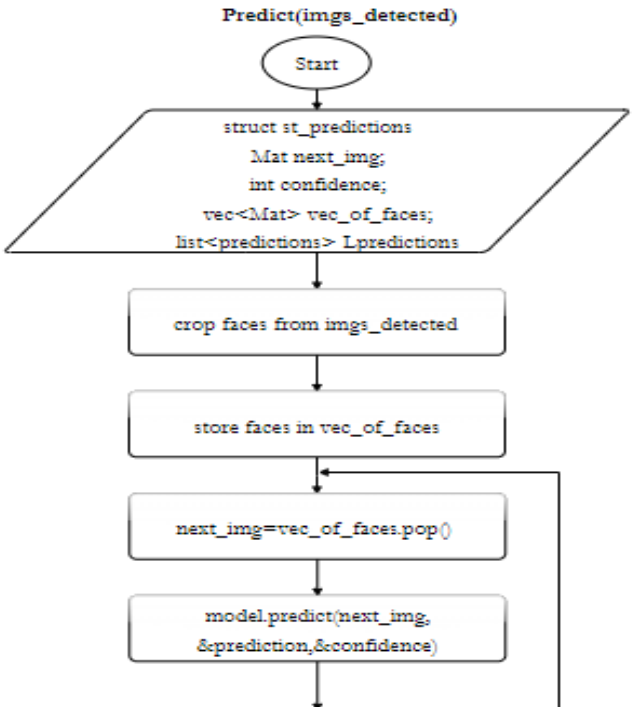


Figure 48- Flowchart of Recognize Face thread

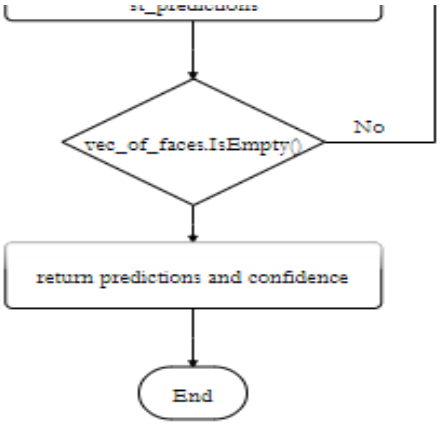


Figure 49- Flowchart of prediction

tAnalyze_Images

After the detect/recognize algorithms are applied, it's necessary to analyse the images, collect the information about the detection and decide if it's necessary to change the direction of the motors.

Whether these algorithms detect/recognize a face, the *tAnalyze_Images* will be responsible to gather the coordinates of faces in the images and prepare the data information to send to *tFace_Center_Control*. After that, in case of recognition/detention a face or not, the tAnalyse_Image will prepare the images to send to interface.

In the beginning of the task it awaits for the conditional variable *cod_Face_Processed* and then checks the operation mode (*auto_mode)*. If manual, it does not analyze the image and jumps to sending the image to the interface. Else if automatic, proceeds to check the recognition mode(*rec_mode),* if it is just to detect faces or to also to recognize a person. Based on the rec_mode, if the face was detected/recognized, it saves the face coordinates (*coodinates_of_face)* and signals the condition variable *cod_Image_analysed.* Then proceeds to send the processed images to interface

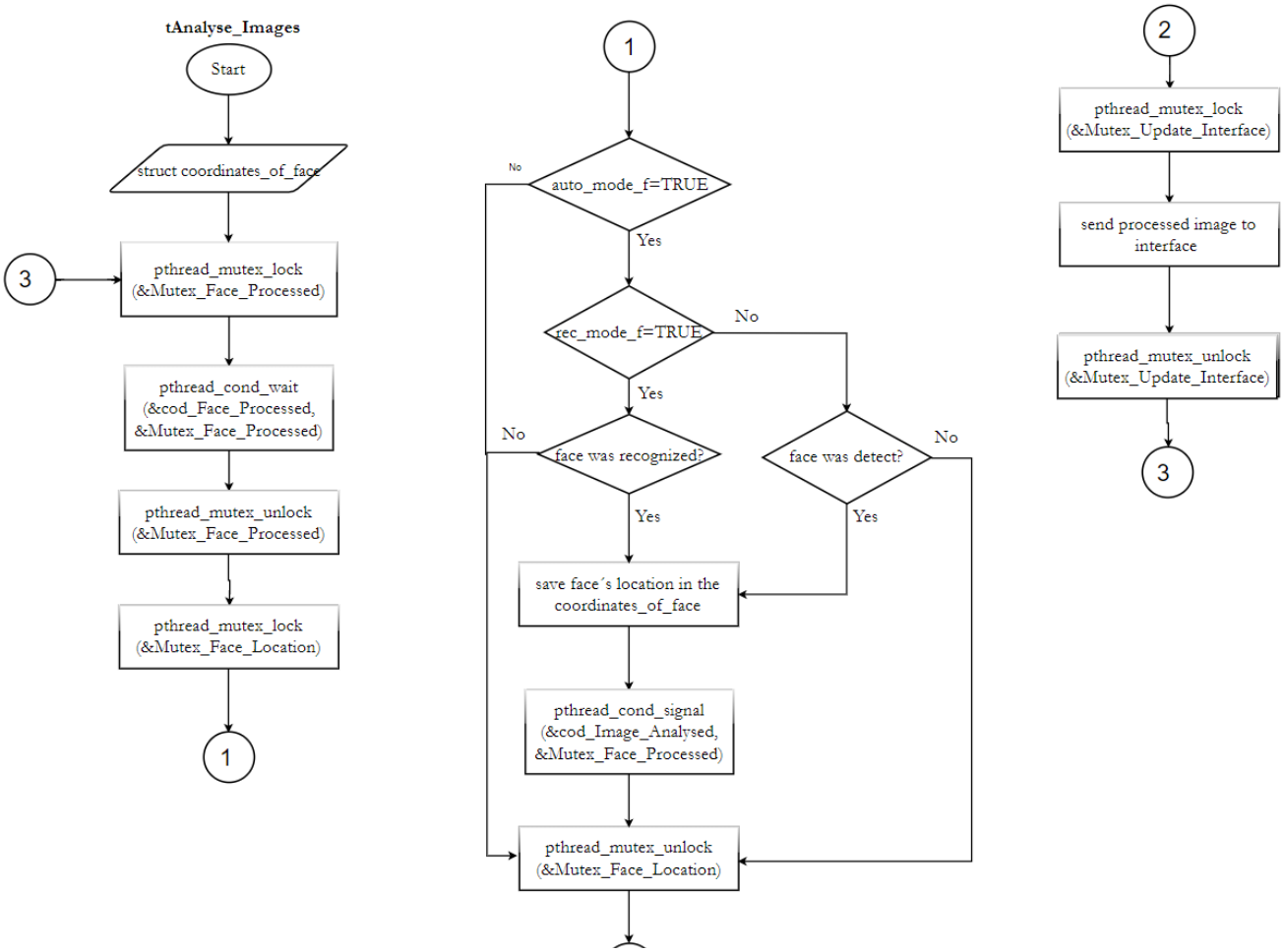
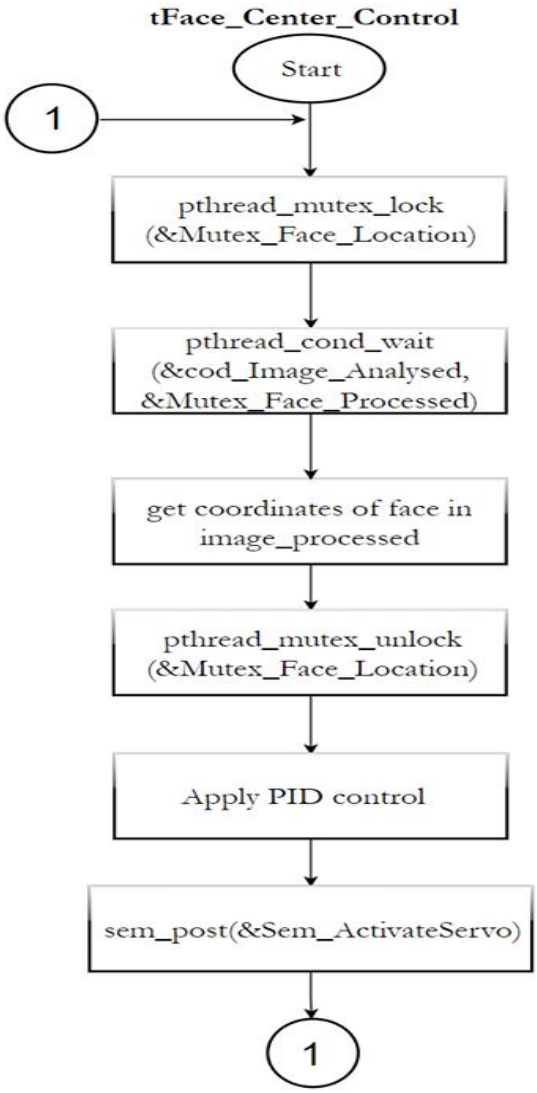


Figure 50- Flowchart of Analyze Image

tFace_Center_Control

This task will await for the signal of the conditional variable *cod_Image_Analysed,* because if it doesn't receive a person detect confirmation, in other words, if there isn't any face in front of camera, it won't change the camera direction, since there's no face to center.

After a face detection/recognition, a PID control is calculated relatively to the distance of the face to the center of image and for the best response of the motors. Finally when this procedure is achieved the semaphore, *Sem_ActivateServo,* will provide the execution of the *tActuate_Servo*.



tActuate_Servo

The tActuate_Servo is the actuation of the servo-motor with the specific PWM value, based on the calculated control. In the first part, this thread will initiate when it breaks from the semaphore waiting state. When a signal for the semaphore is given, it receives the calculated control from the thread tFace_Center_Control and applies the specific PWM value that defines a new angle of the camera.

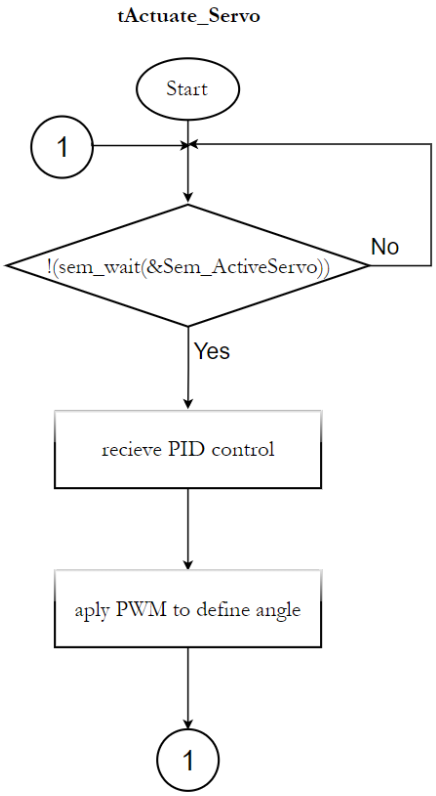


Figure 51- Flowchart of Actuate Servo Motor

tGUI

The graphical user interface will allow the user to interact with the application, defining the operation mode (automatic or manual) and the recognition mode (to apply face recognition or not). The manual mode will allow the User to move the camera direction as he wants to. In automatic mode the system will operate by itself (automatically).

The tGUI, after the modes configurations and after the image process phase, will update the interface with the images on screen. Then, if in manual mode, it will translate the User inputs to control the camera direction, active the Sem_ActivateServo semaphore, and check if the Exit was selected (to return to the interface configuration). Else, if in automatic mode it will jump to the verification of the Exit instruction.

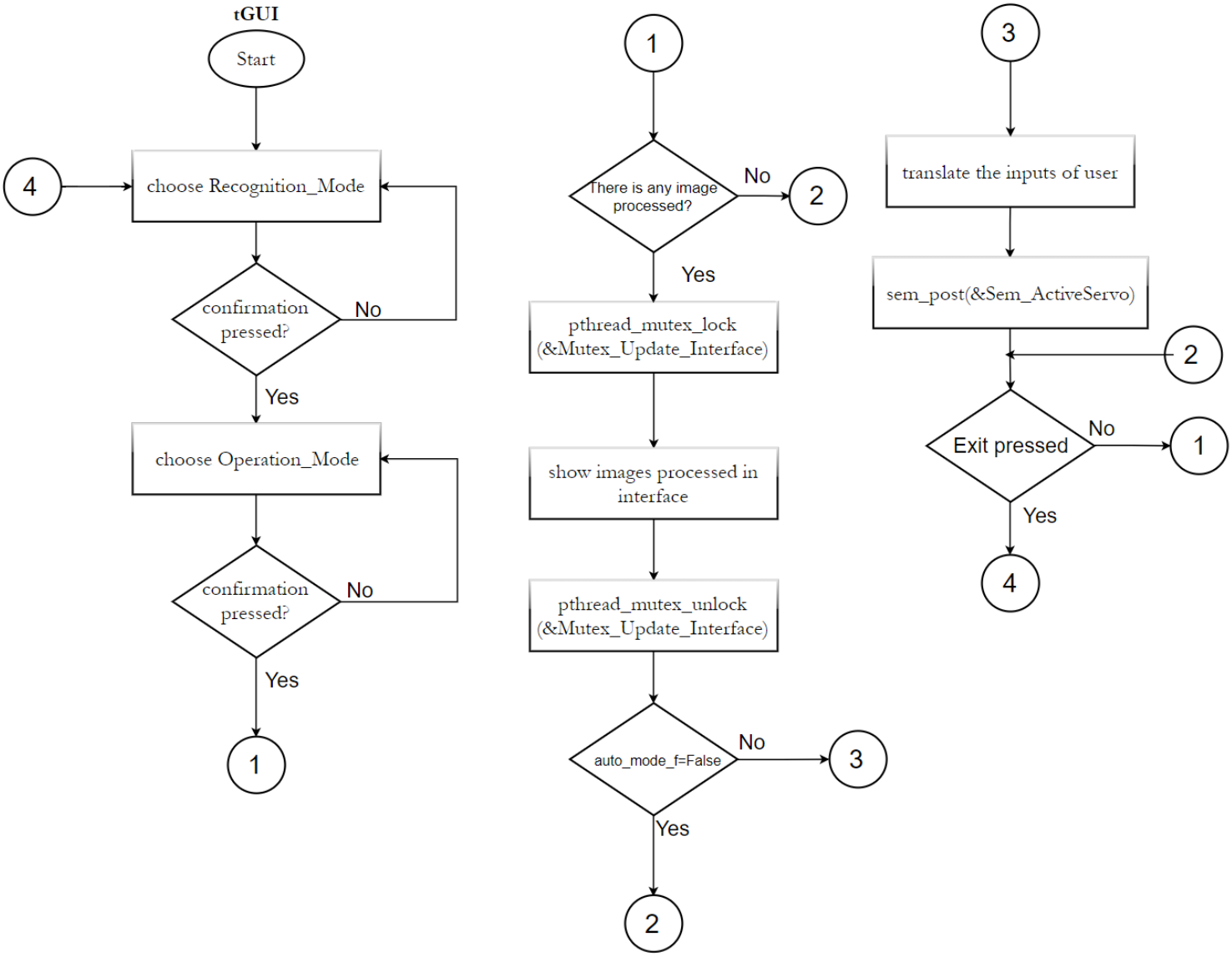


Figure 52- Flowchart of GUI

tIdle

To make our program have lower processing expenses, a thread is necessary to control when the application process can be suspended. We call to this thread *tIdle*. It works through a timer, that triggers a signal when the application spends lot of time inactive (with no face detected or no input inserted).

When a signal is given, the semaphore *Sem_disable* in *tIdle* stops the wait function and resets all of data.

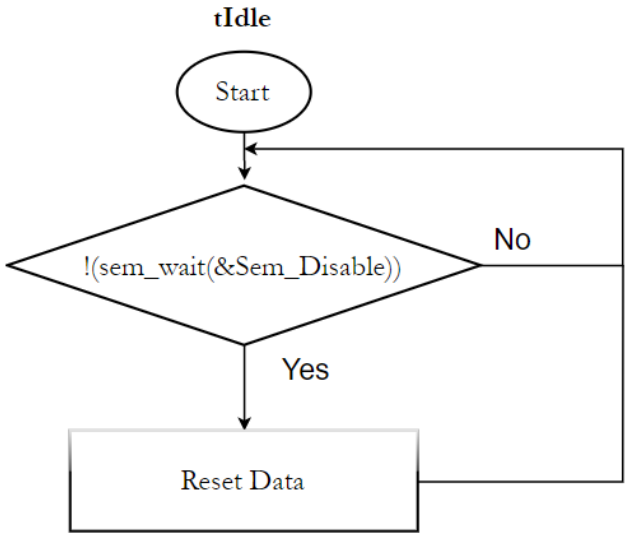


Figure 53- Flowchart of Idle

5. IMPLEMENTATION

5.1. Buidroot configuration

Through the buildroot tool we can configure our image and create the operation system for project devellopment. First is necessary to set the default settings and packages of the board using the command "make raspberrypi3_defconfig".

After the default configurations being added to the image, we must change the firmware in order to support more codecs, camera, etc.

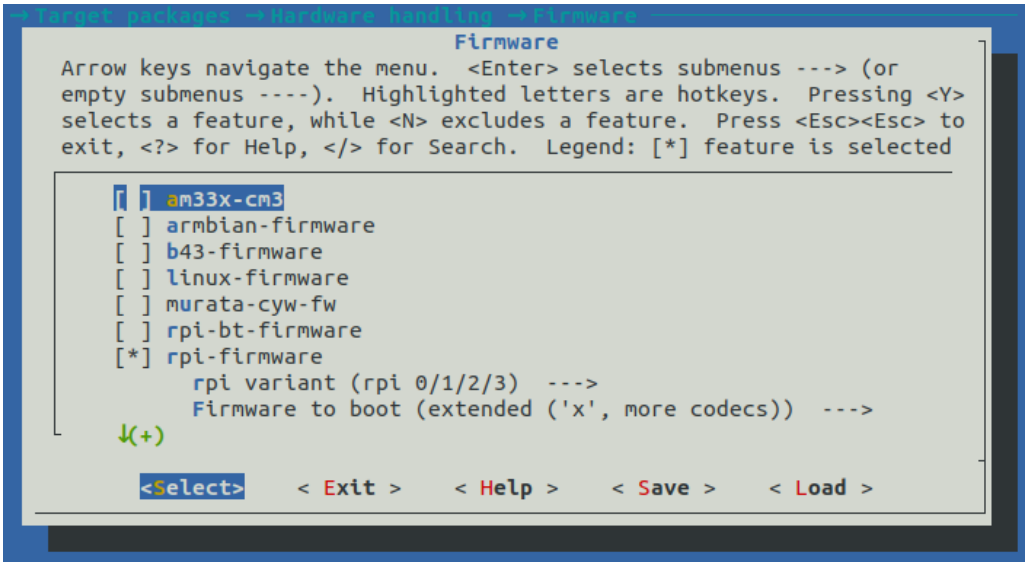


Figure 54 - BuildRoot FirmWare

With the configured firmware, we can add more packages to the image, in this case, the necessary packages for our system are:

- Gstreamer
- Cursors
- Qt5
- Opencv-2.4
- Pwmconfig
- Pigpio
- Rpi-userland
- Bcm2835
- Libv4l
- Wiringpi

5.1.1 Camera Packages:

The essential packages needed to successfully operate the camera are:

- The gstreamer library, as it is the application that Qt uses to play the sound (located at Target Packages > Audio and Video Application);
- The BCM2835 library (located in Target packages > Libraries Hardware Handling) provides an abstraction layer for the hardware peripherals on the board, allowing the manipulation of the registers at user level;
- The libv4l library located in Target packages > Libraries Hardware Handling) is a set of libraries which adds a thin abstraction layer on top of the Raspberry camera.

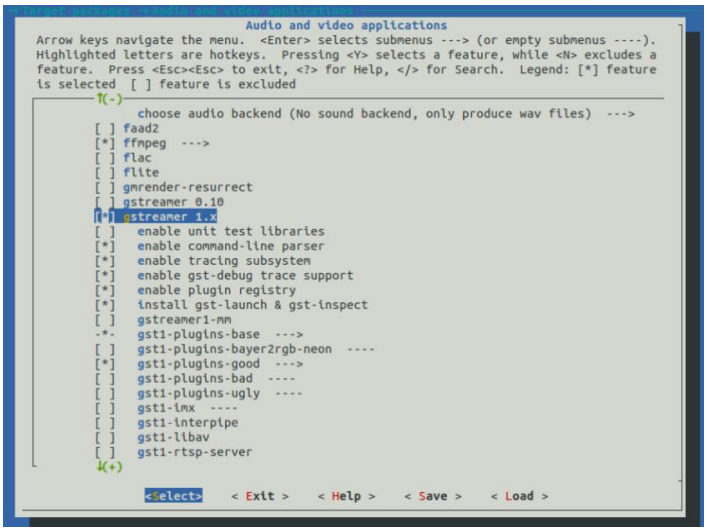


Figure 55- Buildroot gstreamer

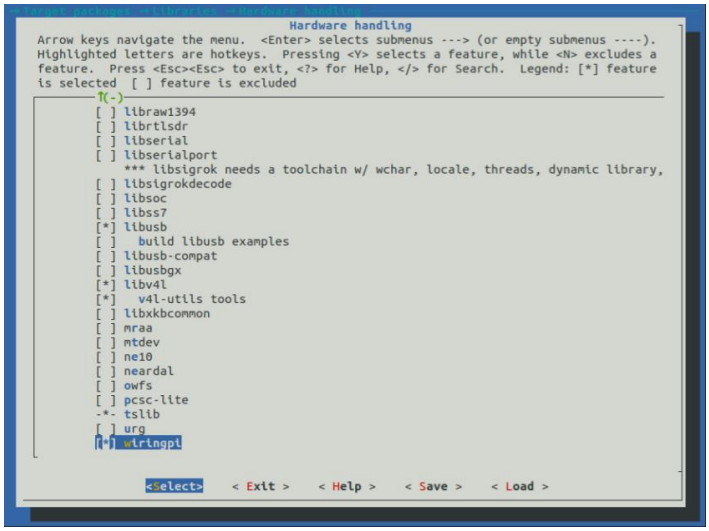


Figure 57 - Buildroot libv4l

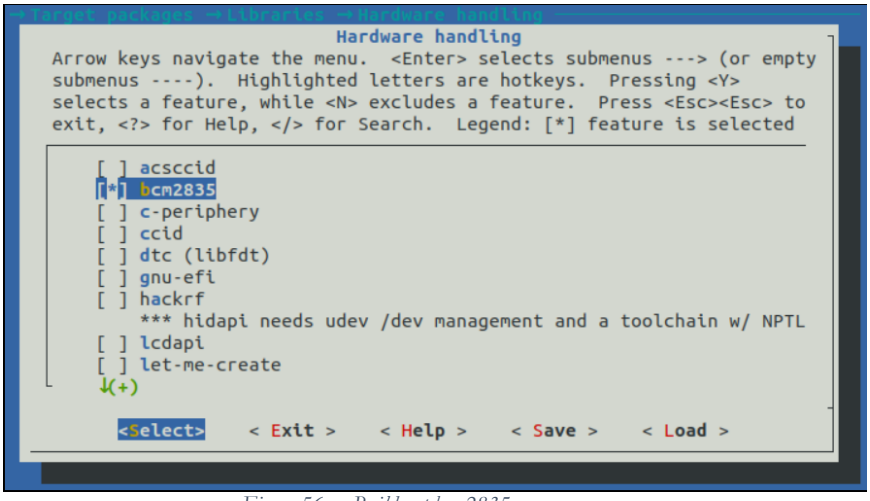


Figure 56 - Buildroot bcm2835

5.1.2 Interface Package:

The graphical library that will be used is Qt. As stated before, Qt supports the design of a graphical interface for embedded systems.

After enabling Qt5 (in Target Packages>Graphical libraries), we decide which packages to remain and which not, consequently to save memory and optimize it.

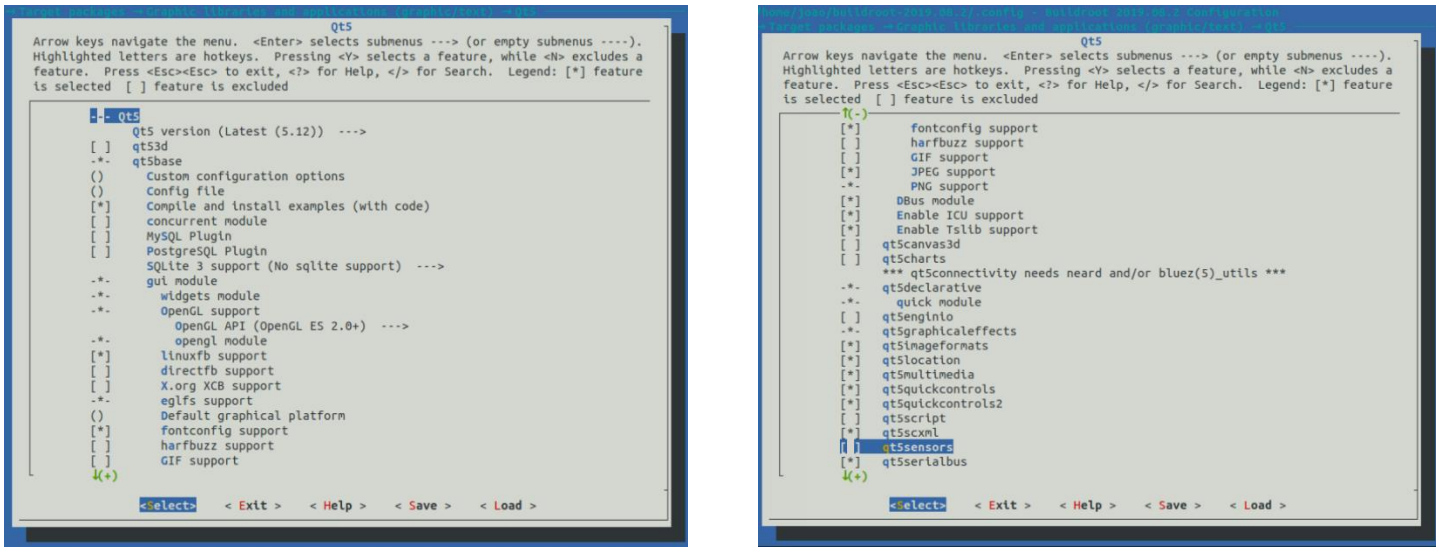


Figure 58 - Qt5 Configurations

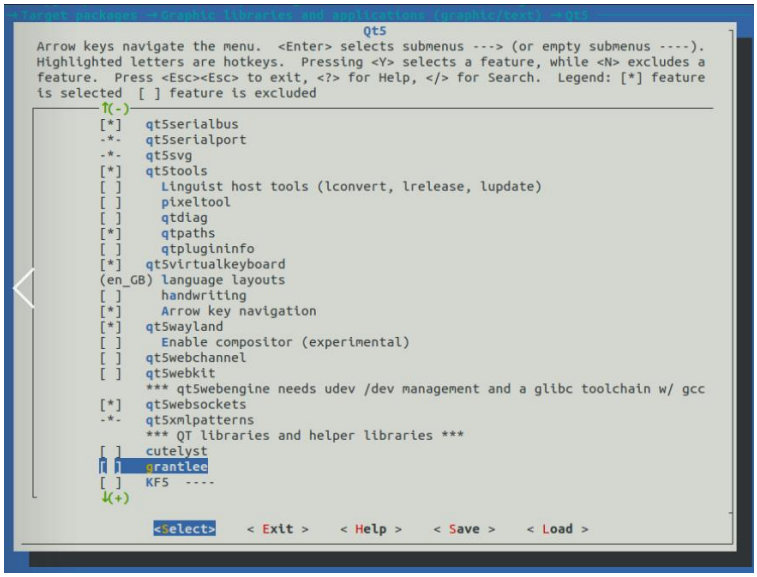


Figure 59 - Qt5 Configurations

Also, in order to display information on the screen, it was added the graphical fonts library to the target, as seen below (in Target packages > Fonts, cursors, icons, sounds and thees).

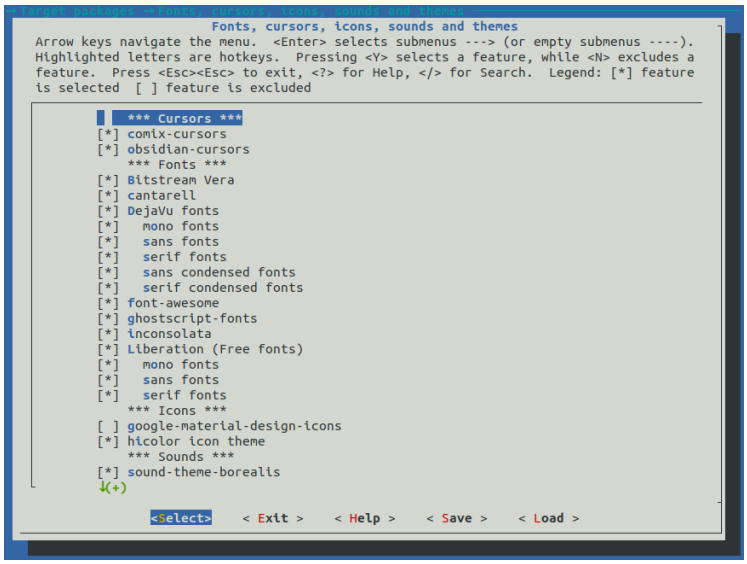


Figure 60 - Buildroot Fonts, cursors, icons

5.1.3 OpenCV – Face Detection/Recognition

The openCV libraries is used for image processing and provides functions for image acquisition, face detection and face recognition. For this project, it will be used the c++ implementations of their API and only be activated the libraries strictly.

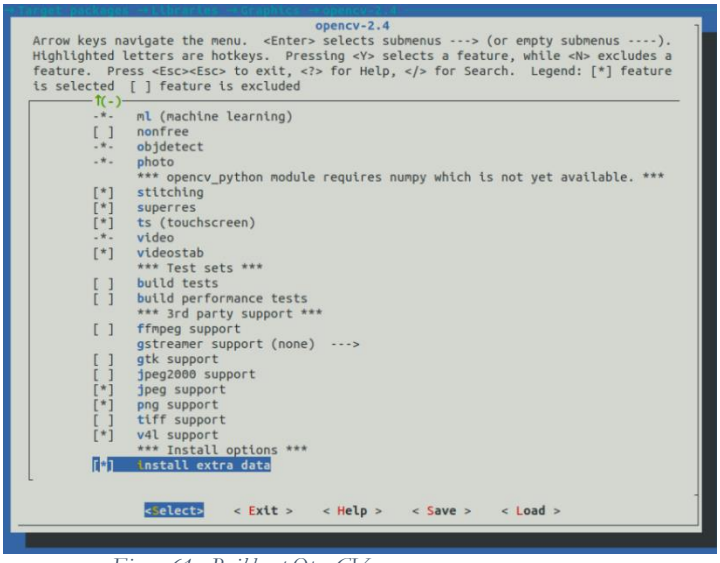


Figure 61 - Buildroot OpenCV

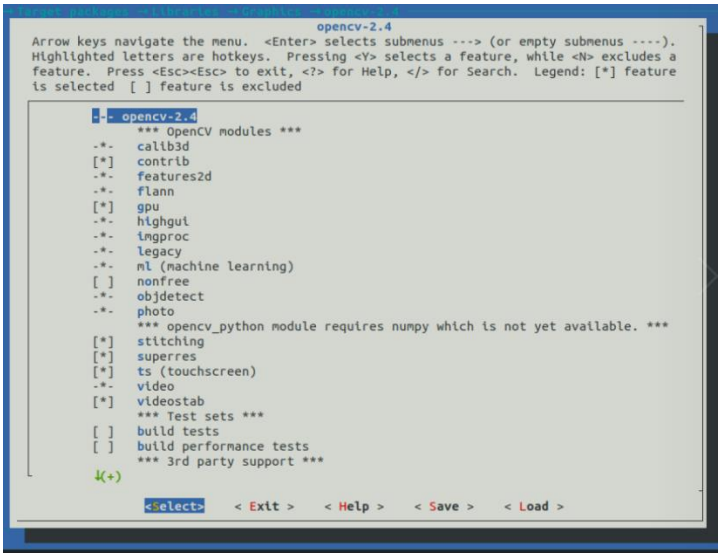


Figure 62 - Buildroot OpenCV Configuration

5.1.4 GPIO Packages:

For the communication with the pins and to enable the pins configuration for pwm and input/output,we enabled **wiringPi**, **pigpio** and **lm-sensors.**

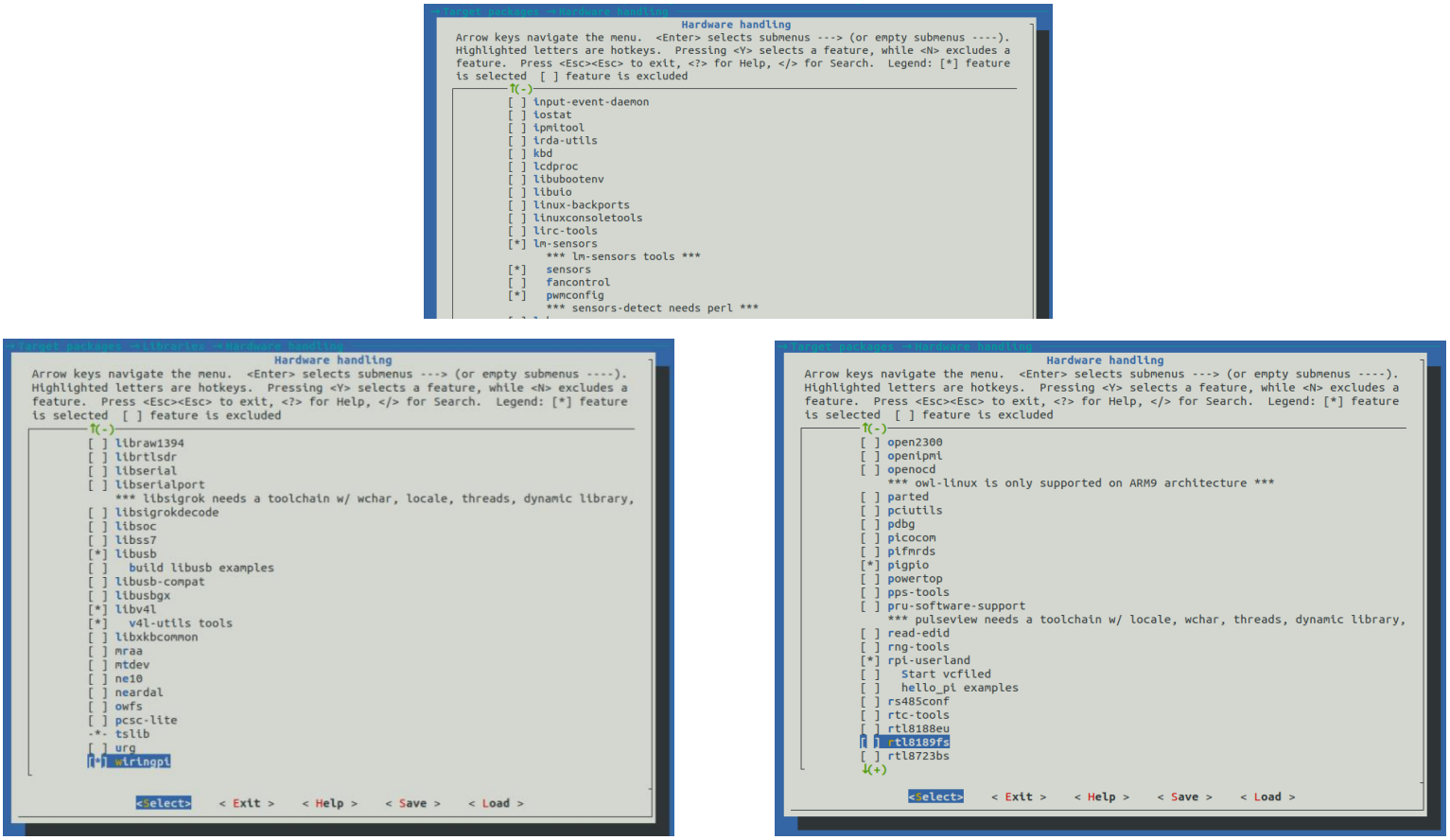


Figure 63 - Buildroot lm-sensor, PiGpio, WiringPi

5.1.5 Mouse Package:

To allow the use of mouse on the Qt graphic interface, it's necessary to enable the mouse input driver (in Target Packages > Graphic libraries and application).

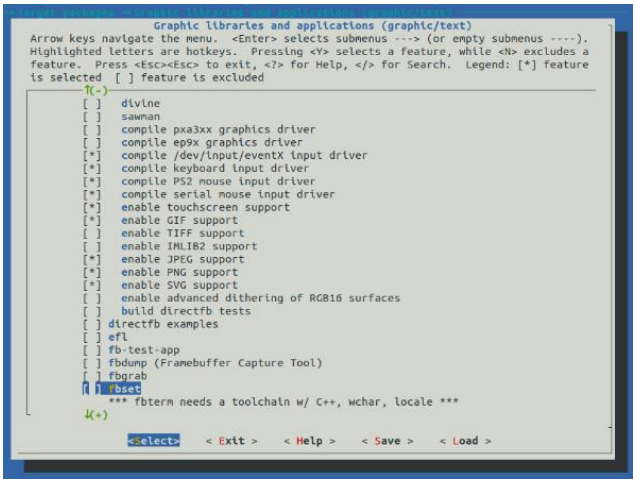


Figure 64 - Buildroot Mouse

5.2 System Initialization File

In order to run the system, it's necessary initialize a group of modules and define its parameters. Those initializations are written in the config.sh file from /etc/inittab.

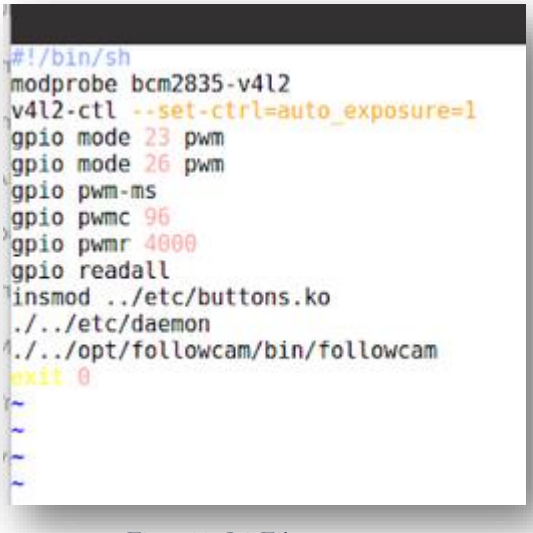


Figure 65 - Init File

The "modprobe" command loads the linux kernel modules like the PWM and the Camera (bcm2835-val2), making them available to the user space to interact with them.

The gpio is configured for 2 pwm signals with a 50hz frequency, according to the formula:

```
pwmFrequency in Hz = 19.2e6 Hz / pwmClock / pwmRange
```

The isnmod command inserts a module into the kernel, in this case, the Button Device Driver. After that, the daemon and the main project is executed.

5.3 Qt Creator – Cross Compiling

As said above, to make the user graphic interface and the code implementation the Qt tool is used. The main feature is that Qt is able to cross-compile, which allows the developer to compile and run directly from the host device to the Raspberry Pi.

As so, the option to run directly from the host must be enable on the configurations. Firstly the "gcc" and "g++" were added to compiler kits.

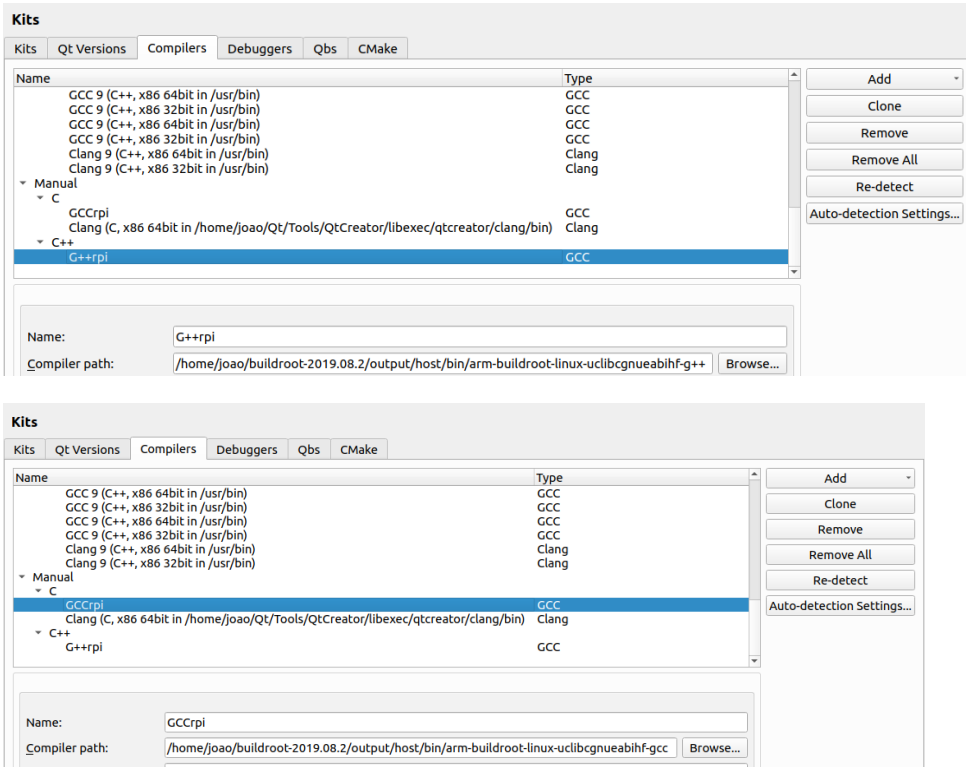


Figure 66 - Qt Compilers

To provide cross-debugging, after the debug configurations on buildroot, we add the Raspberry Pi debugger path to the "Debuggers" subsection.

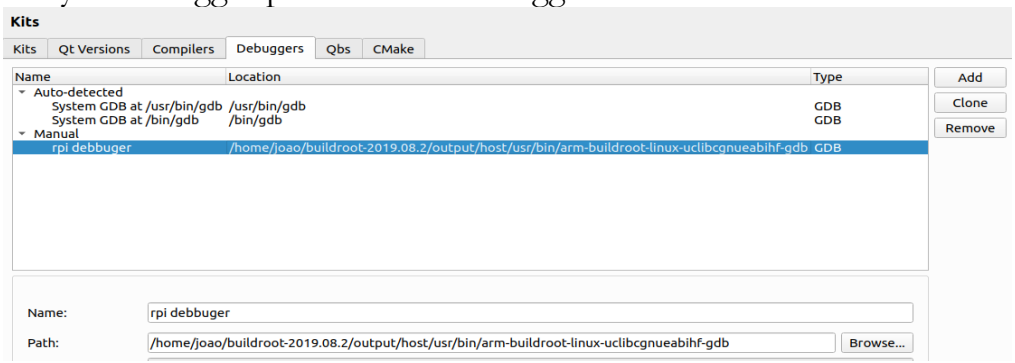


Figure 67- Qt Debugger

To conclude the Qt configuration, a device to associate the debugger, the compilers, the Qt version (among other sections) was added to the kits.

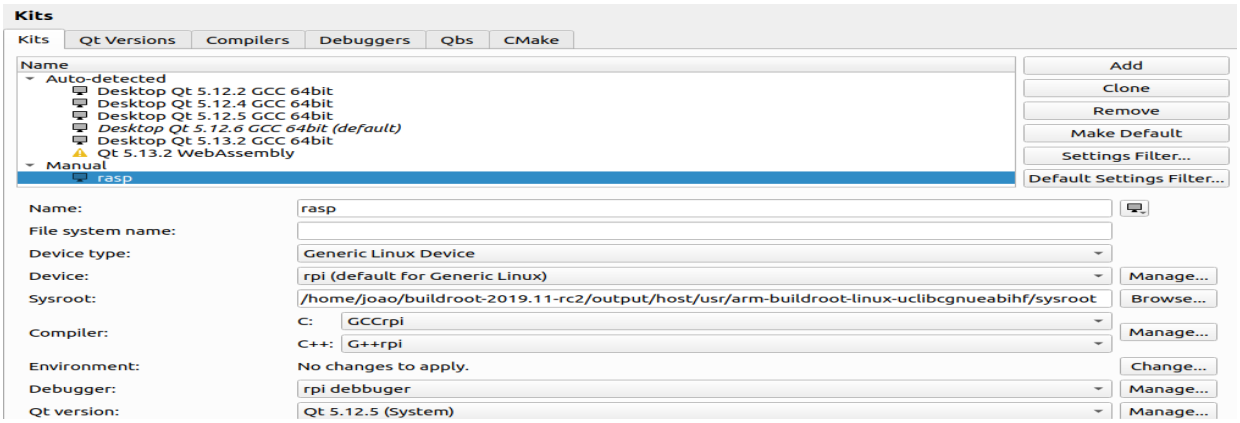


Figure 68 - Qt Kits

5.4 Button Device driver

For our project its necessary to implement a device driver for the power ON/OFF button. This device driver is made using two GPIO pins, one as input, to detect when the button was pressed, and the other as output, to power the elevator.

As so, our device driver the following operation:

```
91  static struct file_operations buttonsDevice_fops = {
92      .owner = THIS_MODULE,
93      .write = buttons_device_write,
94      .read = buttons_device_read,
95      .release = buttons_device_close,
96      .open = buttons_device_open,
97  };
98
```

Code 1 -Device Driver Operations

The write function opens the device file and reads the buffer value. Being it 0 or 1, it executes the "SetGPIOOutputValue". This function access the pins registers and sets the OutputButtonGpioPin (BCM 21) with the read value.

```
42  ssize_t buttons_device_write(struct file *pfile, const char __user *pbuff, size_t len, loff_t *off) {
43
44      struct GpioRegisters *pdev;
45
46      if(unlikely(pfile->private_data == NULL))
47          return -EFAULT;
48
49      pdev = (struct GpioRegisters *)pfile->private_data;
50      if (pbuff[0]=='0')
51          SetGPIOOutputValue(pdev, OutputButtonGpioPin, 0);
52      else
53          SetGPIOOutputValue(pdev, OutputButtonGpioPin, 1);
54      return len;
55  }
```

Code 2 - Device Driver Write function

```c
void SetGPIOOutputValue(struct GpioRegisters *s_pGpioRegisters, int GPIO, bool outputValue) {

    if (outputValue)
        s_pGpioRegisters->GPSET[GPIO / 32] = (1 << (GPIO % 32));
    else
        s_pGpioRegisters->GPCLR[GPIO / 32] = (1 << (GPIO % 32));
}
```

Code 3 - Device Driver SetGPIOOutputValue

The read function reads the On/Off button and returns its value on the user buffer. It executes the "GetGPIOInputValue" function, that access the pins registers, and returns 0 or 1 based on the value on the given GPIO pin (added 48 to the output the character '0' or '1').

```c
ssize_t buttons_device_read(struct file *pfile, char __user *p_buff,size_t len, loff_t *poffset){
    struct GpioRegisters *pdev;

    //pr_alert("%s: called (%u)\n",__FUNCTION__,len);


    if(unlikely(pfile->private_data == NULL))
        return -EFAULT;

    pdev = (struct GpioRegisters *)pfile->private_data;
    p_buff[0]=GetGPIOInputValue(pdev, InputButtonGpioPin)+48;

    //pr_alert("%s: register value is %s\n",__FUNCTION__,p_buff);

    return 0;
}
```

Code 4 - Device Driver read Function

```c
bool GetGPIOInputValue(struct GpioRegisters *s_pGpioRegisters, int GPIO){

    unsigned mask = 1 << (GPIO % 32); //mask used to clear all the bits that doesn't matter

    bool value=(s_pGpioRegisters->GPLEV[GPIO/32] & mask) >> (GPIO % 32); //operation to put the wanted bit on the value
    return value;
}
```

Code 5 - Device Driver GetGPIOInputValue

The open and release functions are responsible to associate the init and exit of the device driver to the respective functions of the kernel. When the insmod is done, the init will be called, when the rmmod is done, the exit is called.

```c
int buttons_device_close(struct inode *p_inode, struct file * pfile){

    //pr_alert("%s: called\n",__FUNCTION__);
    pfile->private_data = NULL;
    return 0;
}


int buttons_device_open(struct inode* p_indode, struct file *p_file){

    //pr_alert("%s: called\n",__FUNCTION__);
    p_file->private_data = (struct GpioRegisters *) s_pGpioRegisters;
    return 0;

}
```

Code 6 - Device Driver Open and Close

5.5 Daemon

The Daemon is used for the On/Off button, operating on the created device driver. First, the daemon initialize the father process and creates a new process with function fork(). If successful, kills the father process, creates a new session (setsid()) and closes all the descriptor files.

```
13  int main(int argc, char *argv[]){
14
15
16      pid_t pid, sid;
17      const time_t timebuf;
18
19      pid = fork(); // create a new process
20
21      if (pid < 0) { // on error exit
22          perror("fork");
23      exit(EXIT_FAILURE);
24      }
25
26      if (pid > 0){
27          printf("Deamon PID: %d\n", pid);
28          exit(EXIT_SUCCESS); // parent process (exit)
29      }
30      sid = setsid(); // create a new session
31
32      if (sid < 0) { // on error exit
33          perror("setsid");
34          exit(EXIT_FAILURE);
35      }
36      // make '/' the root directory
37      if (chdir("/") < 0) { // on error exit
38          perror("chdir");
39          exit(EXIT_FAILURE);
40      }
41      umask(0);
42      close(STDIN_FILENO);  // close standard input file descriptor
43      close(STDOUT_FILENO); // close standard output file descriptor
44      close(STDERR_FILENO); // close standard error file descriptor
45  printf("Deamon PID: %d\n", pid);
46
```

Code 7 - Daemon initialization

In loop, the daemon, through the button device driver, continuously reads the device driver buffer with the input pin value and writes on other pin (for the elevator) the value read. When the system closes, it executes the exit() function.

```
49      char buffer[1]={0};
50      FILE* fd;
51      while (1) {
52
53      fd = fopen (BUTTONS, "r" );
54      if(fd == NULL){
55          perror ("Device doesn't exist\n");
56          exit(EXIT_FAILURE);
57      }
58      fread(buffer,sizeof(int),1,fd);
59      if(buffer[0] == '1'){
60      system("echo 1 > ../dev/buttons");
61      }
62      printf ("Value on buffer: %s\n", buffer);
63      fclose (fd);
64      }
65  exit(EXIT_SUCCESS);
66  }
67
```

Code 8 - Daemon loop

5.6 Classes:

As a constraint, the code was developed in C++ language, divided in various classes. Each class is responsible for encapsulating its own data in the most private way possible. According to this method of encapsulating, the following classes were designed and though out:

- Ccamera
- Cdetection
- Cgui
- Cprediction
- Cthread
- Cthreads
- Followcam

5.6.1. Ccamera:

The camera class is responsible to instantiate the objects related to the camera. Each object initiated by this class, provides the open/shutdown of the camera, the checking if the camera is open/closed and the capturing of a frame.

The function "read()" that is executed just when the camera is opened, captures a frame.

```
class ccamera
{
public:
    ccamera();
    ccamera(int);
    ~ccamera();
    bool captureFrame(Mat&);
    bool isActive(void);
    bool open(void);
    void shutdown(void);
private:
    VideoCapture m_cap;
    int m_device;
};


#endif
```

```
bool ccamera::captureFrame(cv::Mat& image)
{
    Mat frame;
    if( m_cap.isOpened())
    {
        m_cap.read(frame);
        image = frame.clone();
        return true;
    }
    return false;
}
```

Code 9 - Class Ccamera

5.6.2. Cdetection:

The detection class is responsible to instantiate the objects that are related to the face detection on an image.

Each object initiated by this class, loads the classifier, checks if the classifier is empty or not and identifies faces on an image.

It is in this class where the path of haarcascade is saved, as well as the xml file responsible for detecting face algorithm, and is the cascade classifier object that is provided by the <opencv2/opencv.hpp> library.

```
class CDetection
{
public:
    CDetection(const string classifier_path);
    vector<Rect_<int>> identifyFaces(Mat image);
    bool loadClassifier();
    bool classifierEmpty();

private:
    string m_classifier_path;
    CascadeClassifier m_classifier;
};

#endif // CDetection_H
```

Code 10 - Class Cdetection

The function that identifies faces returns a face position to a vector of rectangles. The image is inputted as a parameter and with "detectMultiscale" function the face is detected.

The minimum face size is defined by the last parameter of the "detectMultiscale" function, with the value of 50 by 50 for the project. This is a value related to the distance from the face to the camera, we consider that 1.5 meters was acceptable.

```
vector<Rect_<int> > CDetection::identifyFaces(Mat frame_gray)
{
    vector<Rect_<int>> faces;
    if(!frame_gray.empty() && !m_classifier.empty())
    {
        m_classifier.detectMultiScale( frame_gray, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(50, 50));
    }
    return faces;
}
```

Code 11 - Classes identifyFaces()

5.6.3. Cgui:

The cgui class is responsible to instantiate the objects that are related to the user interface.

Each object initiated by this class, puts captured images on the screen, updates the number of faces that were detected and gets the value of window´s position that is used to calculate the center of the window on tdetect_face.

```
class CGUI: public QObject
{
    Q_OBJECT
public:
    explicit CGUI(/*QObject *parent = 0*/);
    ~CGUI();
    void newimagerecord(const QImage& image);
    void newnumberoffaces(int num_of_faces);
    int* get_win_posx();
    int* get_win_posy();


signals:
    void displaynewimagerecord(const QImage&);
    void displaynumberoffaces(int num_of_faces);

private:
    int window_posx;
    int window_posy;
    int *xptr;
    int *yptr;
};
```

Code 12 - Class CGUI

5.6.4. Cprediction:

The cpredition class is responsible to instantiate the objects that are related to the results of face recognition.

Each object initiated by this class provides to recognition the predictions and confidences level that are used to verify if a face is valid or not.

```
class CPrediction
{
public:
    CPrediction(int label = 0,double confidence = 0);
    bool check_confidence(int threshold);
    int get_label(void);
    double get_confidence(void);
    void set_confidence(double confidence);
    void set_label(int label);
private:
    int m_label;
    double m_confidence;
};

#endif // CPREDICTION_H
```

Code 13 - Class Cprediction

5.6.5. CThread and CThreads:

Both classes are responsible to instantiate the objects that are related to the threads.

CThread has all the functions that were needed to manipulate a thread, such as: “create()”, “shutdown()”, “join()”, “check_run_state()” and “setup_thread()”. To complement, each created thread was added to a CThreads object that contains a list of CThread, where all of threads initiated are saved.

```
class CThread
{
public:
    CThread();
    virtual ~CThread() = 0;
    bool create();
    void shutdown();
    bool check_run_state();
    void setup_thread(int);
    int join();
protected:
    virtual void run(void*) = 0;
private:
    static void* thread_func(void*);
    static bool m_run_flag;
    pthread_t m_thread;
    pthread_mutex_t m_mutex_run_flag;
    pthread_attr_t m_attr;
    struct sched_param m_thread_param;
};

#endif // CTHREAD_H
```

```
class CThreads
{
public:
    CThreads();
    ~CThreads();
    bool addThread(CThread* thread = nullptr,int prio = 2);
    bool run(void);
    int join(void);
private:
    list <CThread*> m_list_threads;
};

#endif // CTHREADS_H
```

Code 14 - Classes CThread and CThreads

5.6.6. Followcam:

The Followcam class contains all the objects, definitions of variables, protections and methods of synchronizations that turns our system complete and ready to run.

To initialize this class code, its necessary some includes, represented on the figure bellow. We also define some Macros prior to turning our code easier to understand. Thereafter, a structure that was shared between threads (tdetect_face and face_center_control) is declared. To finalize, a set of conditional variables and a semaphore were designed to establish synchronization between threads.

```
#include"ccamera.h"
#include"cgui.h"
#include "mainwindow.h"

#include"tacquire_frames.h"
#include"tdetect_face.h"
#include"trecognize_face.h"
#include"tactuateservo.h"
#include"tface_center_control.h"
#include"tgui.h"

#include <semaphore.h>
#include"cthreads.h"
#include<mutex>

#define SIZE_OF_FRAMES_WIDTH 640
#define SIZE_OF_FRAMES_LENGHT 480

#define IDLE 0
#define MANUAL 1
#define AUTOMATIC_W_DETECT 2
#define AUTOMATIC_W_RECOGN 3

#define MAX_X_POS 510
#define MIN_X_POS 110
#define MAX_Y_POS 400
#define MIN_Y_POS 100

class tacquire_frames;
class tdetect_face;
class trecognize_face;
class tactuateservo;
class tgui;
class tface_center_control;

typedef struct info_detection{
    Mat last_frame, gray_frame_recized;
    vector<cv::Rect_<int>> vec_of_faces;
}info_tdectect_face;
```

```
class tacquire_frames;
class tdetect_face;
class trecognize_face;
class tactuateservo;
class tgui;
class tface_center_control;

typedef struct info_detection{
    Mat last_frame, gray_frame_recized;
    vector<cv::Rect_<int>> vec_of_faces;
}info_tdectect_face;

class followcam
{
public:
    followcam();
    ~followcam();

    bool init_threads();

//objects threads
    tacquire_frames *m_tacquireframes;
    tdetect_face *m_tdetectface;
    trecognize_face *m_trecognizeface;
    tactuateservo *m_tactuateservo;
    tgui *m_tgui;
    tface_center_control *m_tfacecentercontrol;
    CThreads m_threads;

//other objects
    ccamera m_cam;
    CGUI updater;

//variables
    Mat last_framecaptured;
    int mode;
    int x_position;
    int y_position;
    double x_deltaerror;
    double y_deltaerror;
    int x_actuate_value;
    int y_actuate_value;

//struct's of threads
    info_tdectect_face data_tdetect_face;

//condition variables
    pthread_cond_t condition_img_acquired = PTHREAD_COND_INITIALIZER;
    pthread_cond_t condition_img_detected = PTHREAD_COND_INITIALIZER;
    pthread_cond_t condition_print_image = PTHREAD_COND_INITIALIZER;
    pthread_cond_t condition_face_center_control = PTHREAD_COND_INITIALIZER;
    pthread_mutex_t condition_img_acquired_mutex = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_t condition_img_detected_mutex = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_t condition_print_image_mutex = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_t condition_face_center_control_mutex = PTHREAD_MUTEX_INITIALIZER;

//mutexes
    pthread_mutex_t mutex_operation_mode = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_t mutex_lastframecap = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_t mutex_detection_data = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_t mutex_face_center_control = PTHREAD_MUTEX_INITIALIZER;

//semaphore
    sem_t sem_servo_manage;
};
```

Code 15 - Class FollowCam

5.7. Main Process:

As the main process we have a set of threads that use all the classes, that were mention before, to get to project goal. So, the main function, as demonstrated bellow, runs those threads.

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    followcam FOLLOWCAM;
    MainWindow w(nullptr, &FOLLOWCAM);
    w.show();
    FOLLOWCAM.updater.set_win_posx(w.Get_window_xpos());
    FOLLOWCAM.updater.set_win_posy(w.Get_window_ypos());
    return a.exec();
}
```

Code 16 - int Main()

At the beginning, the objects of followcam and mainwindow are instantiated, and consequently the threads start running. When the constructer of mainwindow is called, the function responsible to run the threads is executed as well (init_threads()).

```
MainWindow::MainWindow(QWidget *parent, followcam * orig_followcam)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    m_followcam = orig_followcam;
    connect( &m_followcam->updater , SIGNAL(displaynewimagerecord(QImage)), this, SLOT(addnewframerecord(QImage)));
    connect( &m_followcam->updater ,SIGNAL(displaynumberoffaces(int)), this, SLOT(updatenumberoffaces(int)));
    orig_followcam->init_threads();
    Disable_manual_buttons();
    ui->label->setStyleSheet("color: white;");
    ui->label_2->setStyleSheet("color: white;");
    ui->label_6->setStyleSheet("color: white;");
    ui->label_7->setStyleSheet("color: white;");
    ui->label_8->setStyleSheet("color: white;");
    m_followcam->x_position = MAX_X_POS/2;
    m_followcam->y_position = MAX_Y_POS/2;
}
```

Code 17 - MainWindow()

Right after this function is executed, the following threads start running:

- Tacquire_frames
- Tdetect_face
- Trecognize_face
- Tfacecenter_control
- Tgui
- Tactuate_servo

5.7.1. Tacquire_frames:

When the interface is presented to user, the default mode is activated to the system (IDLE mode). In this mode our system doesn't needs to acquire frames, such as in MANUAL operation mode. The system doesn´t needs to send signals to tdetect_face thread. Consequently, on tdetect_face thread it´s necessary to check the operation mode and verify the following steps.

To do so, we use a "switch case", that consonant to the operation mode a different action is executed.

As demonstrated bellow, in IDLE mode the system does nothing to decrease power consumption and in the MANUAL mode the system takes frames and sends a signal to print the captured frame on the GUI.

```
void tacquire_frames::run(void* param)
{

Mat current_frame;
while (1) {

    pthread_mutex_lock( & (m_followcam -> mutex_operation_mode));
    switch (m_followcam -> mode) {
        case IDLE:
            usleep(100000);
            break;

        case MANUAL:
            if (m_followcam -> m_cam.isActive()) {
                m_followcam -> m_cam.captureFrame(current_frame);
            } else {
                m_followcam -> m_cam.open();
                while (!(m_followcam -> m_cam.isActive())) {}
                m_followcam -> m_cam.get_camera().set(CV_CAP_PROP_FRAME_WIDTH, SIZE_OF_FRAMES_WIDTH / 2);
                m_followcam -> m_cam.get_camera().set(CV_CAP_PROP_FRAME_HEIGHT, SIZE_OF_FRAMES_LENGHT / 2);
                m_followcam -> m_cam.captureFrame(current_frame);
            }

            pthread_mutex_lock( & (m_followcam -> mutex_lastframecap));
            m_followcam -> last_framecaptured = current_frame;
            pthread_mutex_unlock( & (m_followcam -> mutex_lastframecap));

            pthread_cond_signal( & (m_followcam -> condition_print_image));
            break;
```

Code 18 - Thread Tacquire_frames

Both in automatic detection mode or in recognition mode, when activated, it continuously captures a frame and sends it to the tdetect_face thread, so it can analyse the image and detect a face.

```
        case AUTOMATIC_W_DETECT:
            if (m_followcam -> m_cam.isActive()) {
                m_followcam -> m_cam.captureFrame(current_frame);
            } else {
                m_followcam -> m_cam.open();
                while (!(m_followcam -> m_cam.isActive())) {}
                m_followcam -> m_cam.get_camera().set(CV_CAP_PROP_FRAME_WIDTH, SIZE_OF_FRAMES_WIDTH / 2);
                m_followcam -> m_cam.get_camera().set(CV_CAP_PROP_FRAME_HEIGHT, SIZE_OF_FRAMES_LENGHT / 2);
                m_followcam -> m_cam.captureFrame(current_frame);
            }

            pthread_mutex_lock( & (m_followcam -> mutex_lastframecap));
            m_followcam -> last_framecaptured = current_frame;
            pthread_mutex_unlock( & (m_followcam -> mutex_lastframecap));

            pthread_cond_signal( & (m_followcam -> condition_img_acquired));
            break;
```

```
        case AUTOMATIC_W_RECOGN:
            if (m_followcam -> m_cam.isActive()) {
                m_followcam -> m_cam.captureFrame(current_frame);
            } else {
                m_followcam -> m_cam.open();
                while (!(m_followcam -> m_cam.isActive())) {}
                m_followcam -> m_cam.get_camera().set(CV_CAP_PROP_FRAME_WIDTH, SIZE_OF_FRAMES_WIDTH / 2);
                m_followcam -> m_cam.get_camera().set(CV_CAP_PROP_FRAME_HEIGHT, SIZE_OF_FRAMES_LENGHT / 2);
                m_followcam -> m_cam.captureFrame(current_frame);
            }

            pthread_mutex_lock( & (m_followcam -> mutex_lastframecap));
            m_followcam -> last_framecaptured = current_frame;
            pthread_mutex_unlock( & (m_followcam -> mutex_lastframecap));

            pthread_cond_signal( & (m_followcam -> condition_img_acquired));
        }
        pthread_mutex_unlock( & (m_followcam -> mutex_operation_mode));
    }
}
```

Code 19 - Thread Tacquire_frames

5.7.2. Tdetect_face:

After the frame is acquired and the mode chosen (detection or recognition), the captured image must be analysed to verify if any face was encountered on the picture.

Before the thread enters in loop, it loads the classifier with an object created by the class Cdetection. After that, the thread is ready to receive signals from tacquired_frames and proceed to analyse the captured picture.

In loop, the last frame captured is saved to an internal variable and the image is prepared for detection.

```
face_detection.loadClassifier();

while (1) {
    pthread_cond_wait( & (m_followcam -> condition_img_acquired), & (m_followcam -> condition_img_acquired_mutex));
    if (m_followcam -> mode == AUTOMATIC_W_DETECT) {
        y_center = (SIZE_OF_FRAMES_LENGHT / 6);
        x_center = (SIZE_OF_FRAMES_WIDTH / 6);

        //prepare the image acquired for detection
        pthread_mutex_lock( &(m_followcam -> mutex_lastframecap));
        cvtColor(m_followcam -> last_framecaptured, gray_frame, CV_BGR2GRAY);
        cv::resize(m_followcam -> last_framecaptured, frame_resized, Size(SIZE_OF_FRAMES_WIDTH / 3, SIZE_OF_FRAMES_LENGHT / 3), 1.0, 1.0, INTER_CUBIC);
        pthread_mutex_unlock( & (m_followcam -> mutex_lastframecap));

        cv::resize(gray_frame, gray_frame_resized, Size(SIZE_OF_FRAMES_WIDTH / 3, SIZE_OF_FRAMES_LENGHT / 3), 1.0, 1.0, INTER_CUBIC);
    } else {
        //prepare the image acquired for detection that will be used in recognition
        pthread_mutex_lock( & (m_followcam -> mutex_lastframecap));
        cvtColor(m_followcam -> last_framecaptured, gray_frame, CV_BGR2GRAY);
        frame_resized = m_followcam -> last_framecaptured;
        gray_frame_resized = gray_frame;
        pthread_mutex_unlock( & (m_followcam -> mutex_lastframecap));

        //collect the original image for recognition
        pthread_mutex_lock( & (m_followcam -> mutex_detection_data));
        m_followcam -> data_tdetect_face.gray_frame_recized = gray_frame;
        pthread_mutex_unlock( & (m_followcam -> mutex_detection_data));
    }
```

Code 20 - Thread Tdetect_face

Next, the system converts and identifies faces on the grey scale (monochromatic) and returns a vector of faces that is saved in "faces". The value of the number of faces found is printed on the GUI. If any face was detected, it is surrounded with a green square and the face's coordinates are calculated, with the purpose to be used on Tface_center_control thread.

```
    //detect faces
    faces = face_detection.identifyFaces(gray_frame_resized);

    pthread_mutex_lock( & (m_followcam -> mutex_detection_data));
    //print on the interface how many faces was found
    m_followcam -> updater.newnumberoffaces(faces.size());
    pthread_mutex_unlock( & (m_followcam -> mutex_detection_data));

    if (m_followcam -> mode == AUTOMATIC_W_DETECT) {
        for (unsigned int i = 0; i < faces.size(); i++) {
            //Create rectange with the image
            Rect face_i = faces[i];
            Mat face = gray_frame_resized(face_i);

            //Create rectange to attach the image -> just in detect mode
            if (m_followcam -> mode == AUTOMATIC_W_DETECT) {
                rectangle(frame_resized, faces[0], CV_RGB(0, 255, 0), 1);
            }

            //get distance from center
            xdistance_from_center = (x_center - (face_i.x + (face_i.width * 0.5)));
            //if a face is on right the value is positive otherwise the value is negative
            ydistance_from_center = (y_center - (face_i.y + (face_i.height * 0.5)));
            //if a face is on up the value is positive otherwise the value is negative
            face_was_found = true;
        }
    }
```

Code 21 - Thread Tdetect_face

After all information is gathered on detection, it signals to trecognize_face in case the current mode is Recognition. Otherwise, if a face is found it signals the tfacecenter_control, so the PID control can be applied and the image showed on the interface.

```
    //save the information about detection
    pthread_mutex_lock( & (m_followcam -> mutex_detection_data));
    m_followcam -> data_tdetect_face.vec_of_faces = faces;
    pthread_mutex_unlock( & (m_followcam -> mutex_detection_data));

    if (m_followcam -> mode == AUTOMATIC_W_RECOGN) {
        pthread_cond_signal( & (m_followcam -> condition_img_detected));
    } else {
        pthread_mutex_lock( & (m_followcam -> mutex_lastframecap));
        m_followcam -> last_framecaptured = frame_resized;
        pthread_mutex_unlock( & (m_followcam -> mutex_lastframecap));

        pthread_cond_signal( & (m_followcam -> condition_print_image));

        if (face_was_found) {
            //apply control to center camera
            pthread_mutex_lock( & (m_followcam -> mutex_face_center_control));
            m_followcam -> x_deltaerror = xdistance_from_center;
            m_followcam -> y_deltaerror = ydistance_from_center;
            pthread_mutex_unlock( & (m_followcam -> mutex_face_center_control));

            pthread_cond_signal( & (m_followcam -> condition_face_center_control));
        }
        face_was_found = false;
    }
}
```

Code 22 - Thread Tdetect_face

5.7.3. Trecognize_face:

This thread is responsible to apply the facial recognition on current pictures that contains the successful detection. To initialize this thread, we must prepare the recognition in following way:

-First, include all internal variables in this function thread, that will be used later.

-Second, collect the images that are saved in the dataset to a vector of images called "images_of_dataset". To make this possible it was used an open cv function called "imread()" that allows the collect images from a specific path.

-Third, create the labels, that are responsible for the number of images collected.

-For last, train the images collected with defined algorithm on the constructor of this class, which is "LBPHFaceRecognizer()".

```
void trecognize_face::run(void *)
{
    string path;
    vector <Mat> images_of_dataset, images_of_dataset_recized;
    Mat img_to_recize, img_recized;
    vector <int> labels;
    Rect face_i;
    Mat face, face_recized, frame_w_recognition;
    Mat recognized_frame_recized, detected_gray_frame_recized;
    int predicted_label = -1;
    double predicted_confidence = 0.0;
    bool person_recognized = false;
    double x_center = SIZE_OF_FRAMES_WIDTH / 4;
    double y_center = SIZE_OF_FRAMES_LENGHT / 4;
    double xdistance_from_center, ydistance_from_center;
    //########### images aquisition ##############
    int j = 0;
    int i = 1;
    for (i = 1; i <= 30; i++) {
        path = "/etc/dataset/user1_" + to_string(i) + ".jpg";
        images_of_dataset.push_back(cv::imread(path, CV_LOAD_IMAGE_GRAYSCALE));
        img_to_recize = images_of_dataset.at(j);
        images_of_dataset.pop_back();
        images_of_dataset.push_back(img_to_recize);
        j++;
    }
    i = 1;
    for (i = 1; i <= 30; i++) {
        labels.push_back(i);
    }

    //########### train of current dataset ##############
    model -> train(images_of_dataset, labels);
```

Code 23 - Thread Trecognize_face

After the initialization is completed and the recognition ready, this thread enters loop, waiting for a signal sent by tdetect_face.

When the signal is received, every single face is verified to predict and check if that face corresponds to the person who the system should follow. The face is accepted only when contains a confidence level lower than 70, otherwise this face is considered unknown. If the person was recognized, its name is printed on the GUI.

```
    while (1) {
      pthread_cond_wait( &(m_followcam -> condition_img_detected), &(m_followcam -> condition_img_detected_mutex));
      pthread_mutex_lock( &(m_followcam -> mutex_lastframecap));
      recognized_frame_recized = m_followcam -> last_framecaptured;
      pthread_mutex_unlock( &(m_followcam -> mutex_lastframecap));
      pthread_mutex_lock( &(m_followcam -> mutex_detection_data));
      //cout << "num faces to detect = "<< m_followcam->data_tdetect_face.vec_of_faces.size() << endl;
      detected_gray_frame_recized = m_followcam -> data_tdetect_face.gray_frame_recized;

      if (m_followcam -> data_tdetect_face.vec_of_faces.size()) {
        for (unsigned int i = 0; i < m_followcam -> data_tdetect_face.vec_of_faces.size(); i++) {
          // Create rectange with the imaCDataSet mydataset;ge
          face_i = m_followcam -> data_tdetect_face.vec_of_faces[i];
          face = detected_gray_frame_recized(face_i);

          model -> predict(face, predicted_label, predicted_confidence);
          prediction = CPrediction(predicted_label, predicted_confidence);
          if (predicted_confidence <= 70) {
            xdistance_from_center = (x_center - (face_i.x + (face_i.width * 0.5)));
            //if a face is on right the value is positive otherwise the value is negative
            ydistance_from_center = (y_center - (face_i.y + (face_i.height * 0.5)));
            //if a face is on up the value is positive otherwise the value is negative
            person_recognized = true;
            cv::putText(recognized_frame_recized, "Joao", cvPoint(face_i.x + (face_i.width * 0.5) - 26, face_i.y - 25), FONT_HERSHEY_COMPLEX_SMALL, 0.6, cvScalar(0, 255, 0), 2, CV_AA);
          }
        }

        cout << "confidence: " << predicted_confidence << endl;
      }
```

Code 24 - Thread Trecognize_face

If a person was successfully recognized the control can be applied (otherwise not) so, the data relatively to the recognition is saved, a signal is sent to Tfacecenter_control thread and the image printed on the interface.

```
      pthread_mutex_unlock( &(m_followcam -> mutex_detection_data));
      //if a person was recognized apply the face center control
      if (person_recognized) {
        //apply control to center camera
        pthread_mutex_lock( &(m_followcam -> mutex_face_center_control));
        m_followcam -> x_deltaerror = xdistance_from_center;
        m_followcam -> y_deltaerror = ydistance_from_center;
        pthread_mutex_unlock( &(m_followcam -> mutex_face_center_control));

        pthread_cond_signal( &(m_followcam -> condition_face_center_control));
      }
      //reset the variable
      person_recognized = false;
      pthread_mutex_lock( &(m_followcam -> mutex_lastframecap));
      m_followcam -> last_framecaptured = recognized_frame_recized;
      pthread_mutex_unlock( &(m_followcam -> mutex_lastframecap));
      pthread_cond_signal( &(m_followcam -> condition_print_image));
    }
```

Code 25 - Thread Trecognize_face

5.7.4. Tfacecenter_control:

This thread is responsible for the application of PID control and centre the face of person on the middle of the screen, based on the coordinate's values received from other threads. The control is calculated and applied for the two axis of space (vertical and horizontal).

To begin, this thread waits for the signal that is sent by the tface_recognition or tface_detection. After receiving the signal, the gains values that will be applied to the face centre control are instantiated. This value was calculated and tested with the next procedure:

- First, we reset all gains values.
- We increase the kp value until the system oscillates around the reference value. When the camera doesn't stop moving, we collect the number of kp and split it in half.
- Next, to get the Integral Gain value we increase the number until the system lost their stability. With this gain is possible to the system moving faster and reach the correct value of output faster too.
- In way to get the correct value of the kd we increase the kd value too. With this final gain it can be possible watch the camera founding the correct position with less oscillations than when we test the ki value.

```
void tface_center_control::run(void *){
    value_pwm=MAX_X_POS/2;
    y_value_pwm=MAX_Y_POS/2;

    while (1) {

        pthread_cond_wait( &(m_followcam -> condition_face_center_control), &(m_followcam -> condition_face_center_control_mutex));

        if (m_followcam -> mode == AUTOMATIC_W_DETECT) {
            ki = 0.2;
            kp = 0.3;
            kd = 0.08;
            y_ki = 0.03;
            y_kp = 0.25;
            y_kd = 0.05;
            deltatime = 0.5;
        } else {
            ki = 0.1;
            kp = 0.25;
            kd = 0.09;
            y_ki = 0.02;
            y_kp = 0.15;
            y_kd = 0.05;
            deltatime = 0.8;
        }
```

Code 26 - Thread Tfacecenter_control

Below is presented how the PID calculated. Firstly, the variables are defined (error, delta error, prev_error, ci, output, deltatime and value_pwm). The variable error is calculated previously in another thread, briefly is the distance between the face location from centre of the screen. For proportional gain is the only variable necessary, in other way, for the integral gain we also need the previous errors. We make the sum of past errors and with them, we build the integral action.

Since final project doesn't represent a perfect system, it was necessary to delimit maximum and minimum ki evolution in both axis.

To conclude, we gather all of actions and merge them to build the variable output, that represents the value of angle and put it on pwm value by sending a semaphore to another thread that call´s “tactuate_servo”.

```
      pthread_mutex_lock( & (m_followcam -> mutex_face_center_control));
      error = m_followcam -> x_deltaerror;
      y_error = m_followcam -> y_deltaerror;

      delta_error = error - prev_error;
      y_delta_error = y_error - y_prev_error;

      prev_error = error;
      y_prev_error = y_error;

      ci = ci + (error * deltatime * ki);
      y_ci = y_ci + (y_error * deltatime * y_ki);

      if (ci > 25) {
        ci = 25;
      } else if (ci < -25) {
        ci = -25;
      }
      if (y_ci > 20) {
        y_ci = 20;
      } else if (y_ci < -20) {
        y_ci = -20;
      }

      output = (error * kp) + (ci) + (delta_error / deltatime) * kd;
      y_output = (y_error * y_kp) + (y_ci) + (y_delta_error / deltatime) * y_kd;

      value_pwm += (int) output;
      y_value_pwm -= (int) y_output;

      if (value_pwm > 510) {
        value_pwm = 510;
      } else if (value_pwm < 110) {
        value_pwm = 110;
      }

      if (y_value_pwm > 400) {
        y_value_pwm = 400;
      } else if (y_value_pwm < 100) {
        y_value_pwm = 100;
      }
```

Code 27 - Tfacecenter_control

5.7.5. Tgui

This thread is responsible to update the images on the screen when it receives the signal to do so. The thread has a delay of 200ms in order to balance between the frame rate of images and the mouse position update, allowing the user to feel comfortable when interacting with the interface.

```
void tgui::run(void *){

  while(1){

    pthread_cond_wait(&(m_followcam->condition_print_image), &(m_followcam->condition_print_image_mutex));
    pthread_mutex_lock(&(m_followcam->mutex_lastframecap));
    cv :: resize(m_followcam->last_framecaptured, m_followcam->last_framecaptured , Size(SIZE_OF_FRAMES_WIDTH,SIZE_OF_FRAMES_LENGHT), 1.0, 1.0, INTER_CUBIC);
    m_followcam->updater.newimagerecord(QImage(m_followcam->last_framecaptured.data, m_followcam->last_framecaptured.cols, m_followcam->last_framecaptured.rows, m_followcam->last_framecaptured.step, QImage::Format_RGB888));
    pthread_mutex_unlock(&(m_followcam->mutex_lastframecap));
    usleep(200000);
  }
}
```

Code 28 - Thread Tgui

5.7.6. Tactuate_servo:

This thread is responsible to actuate the servos. It will be always in loop waiting in the semaphore until a semaphore post is made.

Prior to the thread, the maxim x and y positions are respectively, 510 and 400, and minimum x and y positions are respectively, 110 and 100.

```
void tactuateservo::run(void *param){
    while(1){
        sem_wait(&m_followcam->sem_servo_manage);


        std::string pwm = "gpio pwm 23 ";
        pwm += std::to_string(m_followcam->x_actuate_value);
        system(pwm.c_str());


        pwm = "gpio pwm 26 ";
        pwm += std::to_string(m_followcam->y_actuate_value);
        system(pwm.c_str());

    }
}
```

Code 29 - Thread Tactuate_servo

5.8. Final Result

Guiding through the waterfall method, we develop the following project. As the structure to copulate all the components was used the following wooden box, a making it quite rigid and solid.

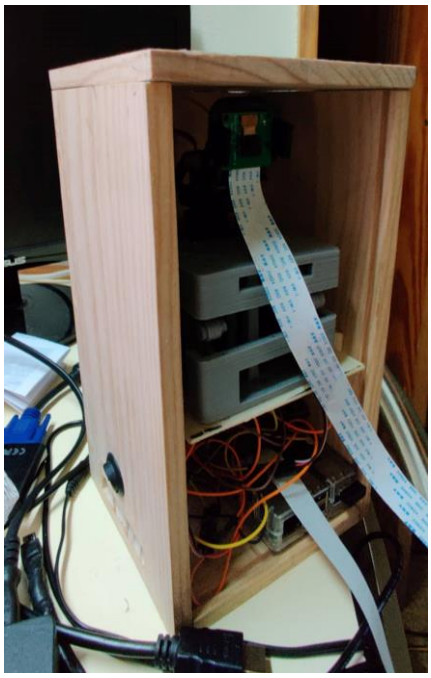


Figure 69 - Inside Box Final Product

The space was measured so that everything would we compact, occupying the less space possible. Regarding the resto of the components, we 3D printed and added an elevator for the tilt and pan mechanism so that the interaction with the user was more appealing. In manner to raise or lower the elevator we add a button.

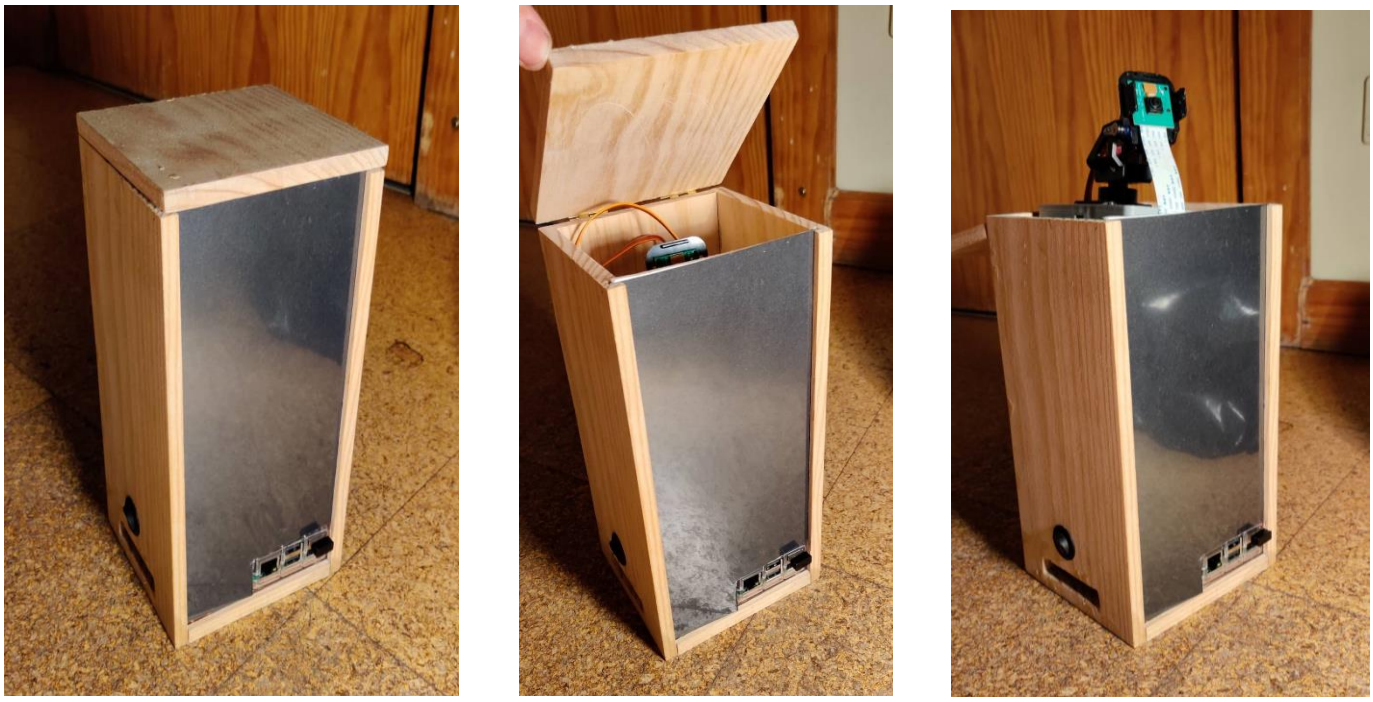


Figure 70 - Final Product

Regarding the interaction with the user, we also created an interactive and attractive graphical interface that allows the user to choose different operating modes or even shut down the system.

This is the main menu, which inform to the user witch mode is activated and how many faces the system detected.

The user can in any instant click on play button to change the mode for IDLE mode. On settings button to open setting menu and define the pretended mode.

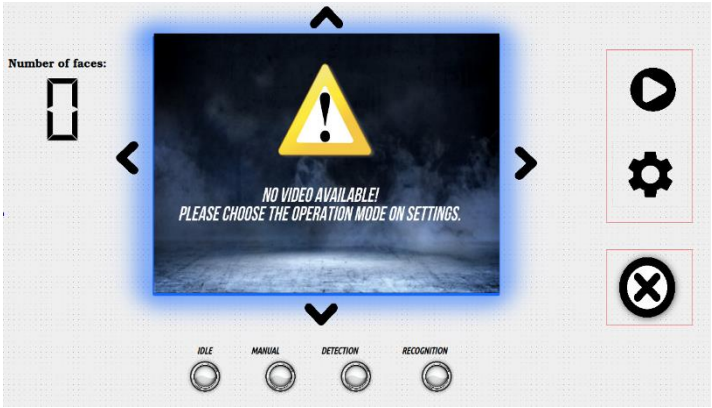


Figure 71 - Graphic User Interface

This is the menu that pops up for the settings menu. It allows to the user choose an automatic operation mode or manual operation mode.

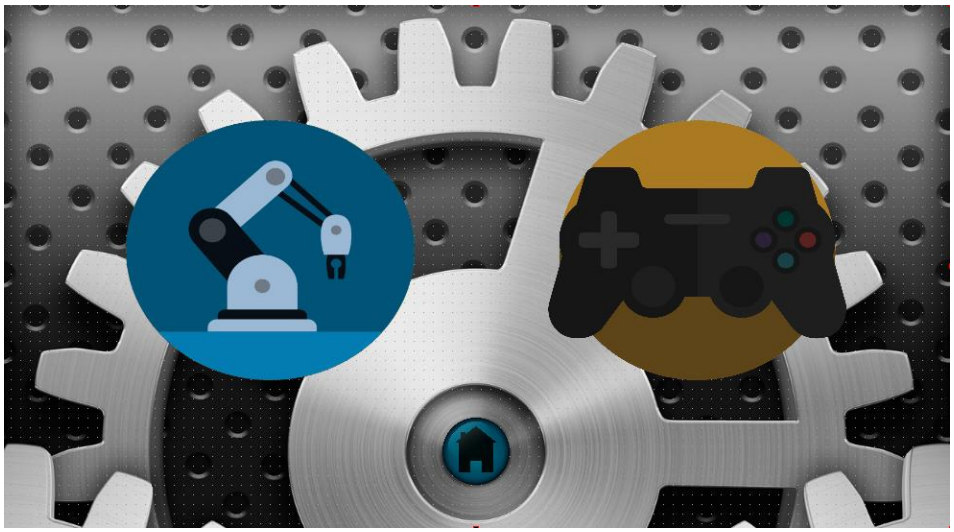


Figure 72 - Interface Mode Menu

The automatic menu allows the user to choose if it wants the system to work on automatic with face detection mode or in automatic with recognition mode.

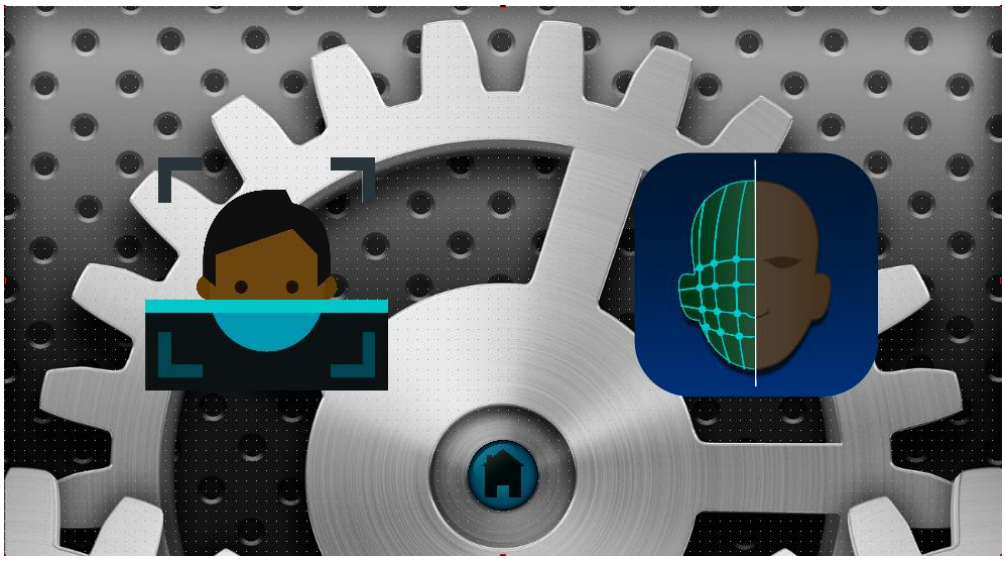


Figure 73 - Interface Automatic Mode

6. Verification Phase

With this chapter we pretend to present all of tests made upon the project to evaluate the final product. Some are mention on design phase and others on implementation phase.

The tests are essential in the development of a project. If they are carrier out in an orderly and organized way, they can even make the implementation part faster and more efficient.

6.1 Hardware Test cases

Here is tested all the hardware components to determine if they are working correctly and shape to be used in the project.

6.1.1. Raspberry Pi camera Test cases

In raspberry pi camera test cases we test essentially the resolution, the colors, the effects, and so ever. Briefly, we learn how to work with camera in perfect conditions.

To do this test we use the next command lines:

"v4l2-ctl -d/dev/video0-list-ctrls"

Test	Expected Output	Real Output
Take a Picture	Picture appear on the screen.	**The picture matches with the reality;**
Take a video with different resolutions	The change in video quality;	**The quality of video was changed;**
Take a set of pictures and show them on the screen	See the video effect	**The video effect was presented, but with low frame rate**

Table 9 - Pi camera Test cases

6.1.2 Tilt and pan Test Cases

The tilt and pan mechanism after mounted, was tested and its limits defined.

Test	Expected Output	Real Output
Input a 50Hz square Wave	The pwm move a servo	The servo moves when pwm was 2.5% of dutty-cyle
Input pwm in x-axis	The servo move into horizontal axis	The servo moves 180º in horizontal axis
Input pwm in y-axis	The servo move into vertical axis	The servo moves 180º in vertical axis

Table 10 - Tilt and pan Test Cases

6.1.3 Power supply Test Cases

According to our calculations the current that power supply release is enough for supply raspberry, camera and motors, however this did not happen.

Test	Expected Output	Real Output
Raspberry supplies all components	All devices work in perfect conditions	The raspberry pi would turn itself off
Supply all of components with an extra power supply	All devices work in perfect conditions	All of components works in perfect conditions

Table 11 - Power supply Test Cases

6.1.4 Button Test cases

For the elevator to go up it was necessary to apply a button, consequently it was necessary to carry out its tests.

Test	Expected Output	Real Output
Put led instead of the button to test device driver	Led light turn on and turn off	The led turned on and off
Apply the button to turn on the led	When clicked on button the led turn on	The led turned on
Apply the button to turn on an pwm value	When clicked a pwm value be applied on elevator servos	The elevator goes up

Table 12 - Button Test cases

6.1.5 Mouse Test Cases

To interact with interface the user uses a USB mouse.

Test	Expected Output	Real Output
Input the usb mouse	Cursor mouse appear and works	Mouse appear and works

Table 13 - Mouse Test Cases

6.1.6 Elevator Test Cases

The elevator was 3D printed. However due to design flaws on the 3D schematic the first assembly of the elevator wasn't successful.

Test	Expected Output	Real Output
Fisrt assembly	Well assembly	The screws were small
Second assembly	Well assembly	All of components works well

Table 14 - Elevator Test Cases

6.2. Software Test cases

Just like the hardware tests, there are also tests regarding the software that belongs to our project. Here we can measure the efficiency of our implementation.

6.2.1 Face detection Test cases

After taking a picture there was some necessary changes on image to put the detection in its maximum conditions.

Test	Expected Output	Real Output
Print the image in gray scale	Print the image in black and white	The image was printed in black and white
Resize the image in order to becomes the detection faster	The size of the images printed was changed	The size of image change
Detect faces on the picture	Put a square around face founded	A square was printed around a face
Increase Frame Rate	Frame rate increased to 15 FPS	Frame rate between 5-10 FPS

Table 15 - Face detection Test cases

6.2.2 Face Recognition Test cases

For the recognition it was necessary to successfully implement the detection. The recognition test evolved to save images to dataset path, train the data set, and continuously recognize the faces in real time.

Test	Expected Output	Real Output
Save one image to dataset path	An image appears on that specific path	The image was created on this path
Train of dataset	The train successfully done	The train was successfully done
Make a prediction of an image captured with a face	The confidence result being printed on the screen	The confidence low
Make another dataset better and with more images	The confidence result being more high	The confidence increase
Increase frame rate	Frame rate of 10 FPS	It presents very low rate of 1 FPS

Table 16 - Face Recognition Test cases

6.2.3 PID Test cases

In order to carry out these tests the entire structure had to be assembled in perfect condition and the manual operation mode to operate without any type of error. However, the PID control is limited to frame rate. Higher the frame rate, higher the gains and more accurate is the control.

Test	Expected Output	Real Output
Increase kp (horizontal)	The angle of camera oscillates around our face	The camera oscillates around my face
Increase ki (horizontal)	The camera moves faster when the error value is bigger	The camera found my face faster, the speed of moving increase when the error value is bigger
Increase kd (horizontal)	The camera do not oscillate, move faster and stop in the expected angle	The camera moves faster, didn´t fluctuate and stop in expected output
Do the same tests for the vertical axis	Works well, with a soft controller	Work in perfect conditions

Table 17 - PID Test cases

6.2.4 Interface Test cases

For the user to control the system and visualize what the system execution, it's necessary a graphic interface. The interface is tested to it determine if it was well developed without a deformation.

Test	Expected Output	Real Output
Clicked on buttons	The image of buttons change and entry in a new menu interface	The image of buttons became clearer and entry in a new menu interface
Send an image to the screen	An image was successfully sent	The image was printed on the interface
Open various windows and menus	Various menus available and open if clicked	All the windows/menus open successfully when clicked

Table 18 - Interface Test cases

7. Conclusion and Future Work

Making a retrospective view on the project result, we can assure that we take great proud on the outcome. It was a hard and costly project, that provided us with the challenges that overcome through though dedication. At the end, the contents and skills gathered in the making of the project are more valuable than the physical outcome.

The most challenging part of our project was in fact generating an image. Since the lack on documentation and information, the creation of the image was based on many attempts and errors, where each attempt took many hours.

The report as well took a lot of dedication and time. With the teacher guidance, the implementation of a Waterfall Method based report was perfected and represent a useful for the future.

With our image created, we began the raw code development of the project, testing each module at a time. Here we started the classes concepts on Real Time Systems, implementing various Task using pThread, used synchronization techniques such as Mutexes, Semaphores and Signals. A Device Driver and a Daemon for the On/Off Button was implemented as well.

However, when making the design phase there were a lot of factors that weren't well thought through, resulting in being implemented differently. For example, the Daemon initially was intended for the acquisition of frames, but with problems on the share of the image with the main process, the idea was aborted and changed to continuously read the On/Off button. The PWM for the pan&tilt and the elevator was successfully implemented, such as the box final structure and elevator (used to enhance the project as a product). Both the detection and the recognition success in their job, they didn't achieve a peak performance with high frame-rate resolution. The Frame takes an huge role in our project, with more frames per second better the tracking. Since we couldn't achieve a higher resolution of 10 FPS, in makes us feel as an incomplete project.

Therefore, there is always room for improvement, and the case above represents one of the major improvements: providing a higher Frame rate. The capsule and the structure, and even the graphic interface, could be improved to be more consumer attractive, but it wouldn't matter if framerate stayed low.

An other future add-on that we envisioned, would we to video stream the video via WiFi LAN to a local host, stepping ino the IoT world

8. Appendix

8.1 V. Gantt Diagram

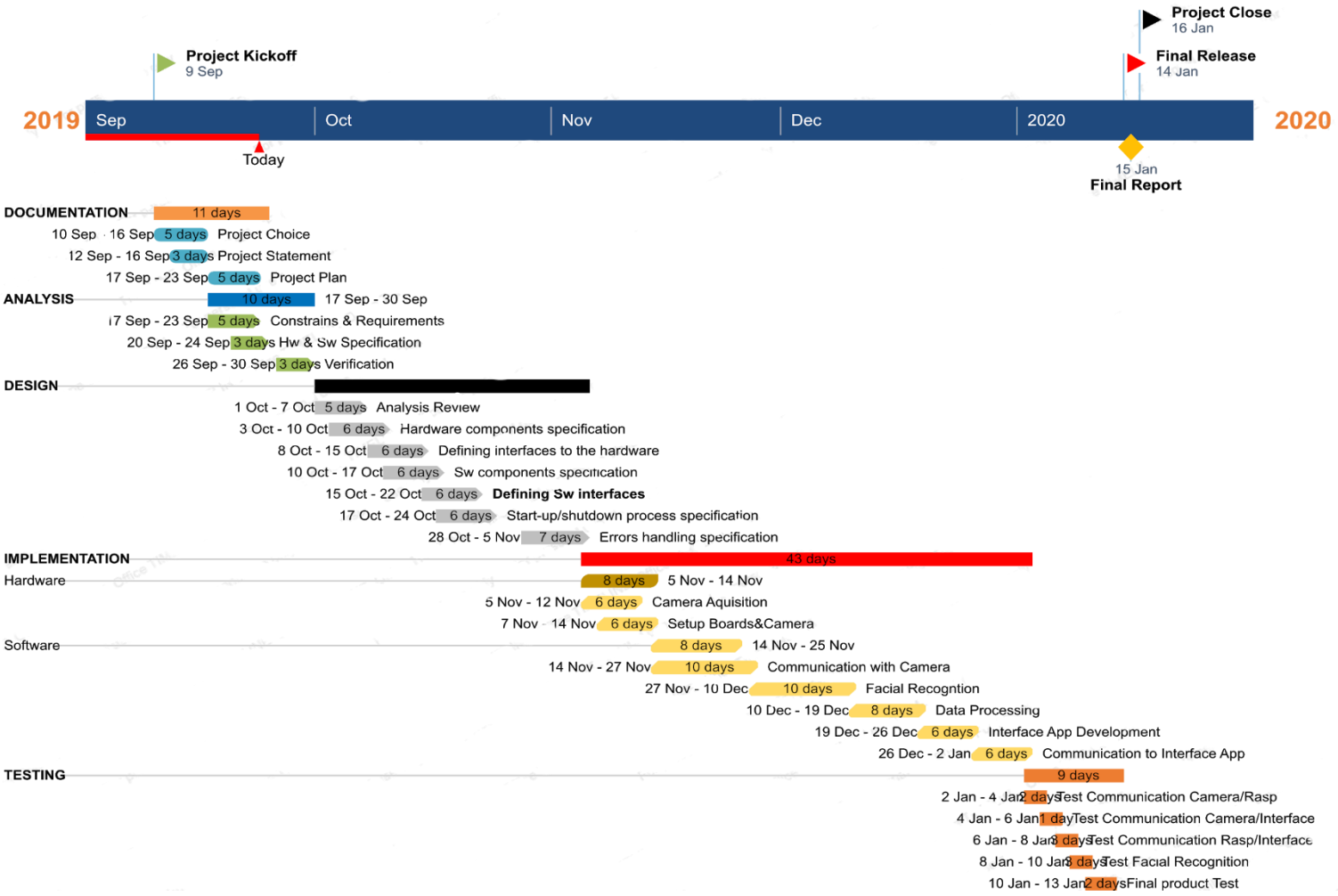
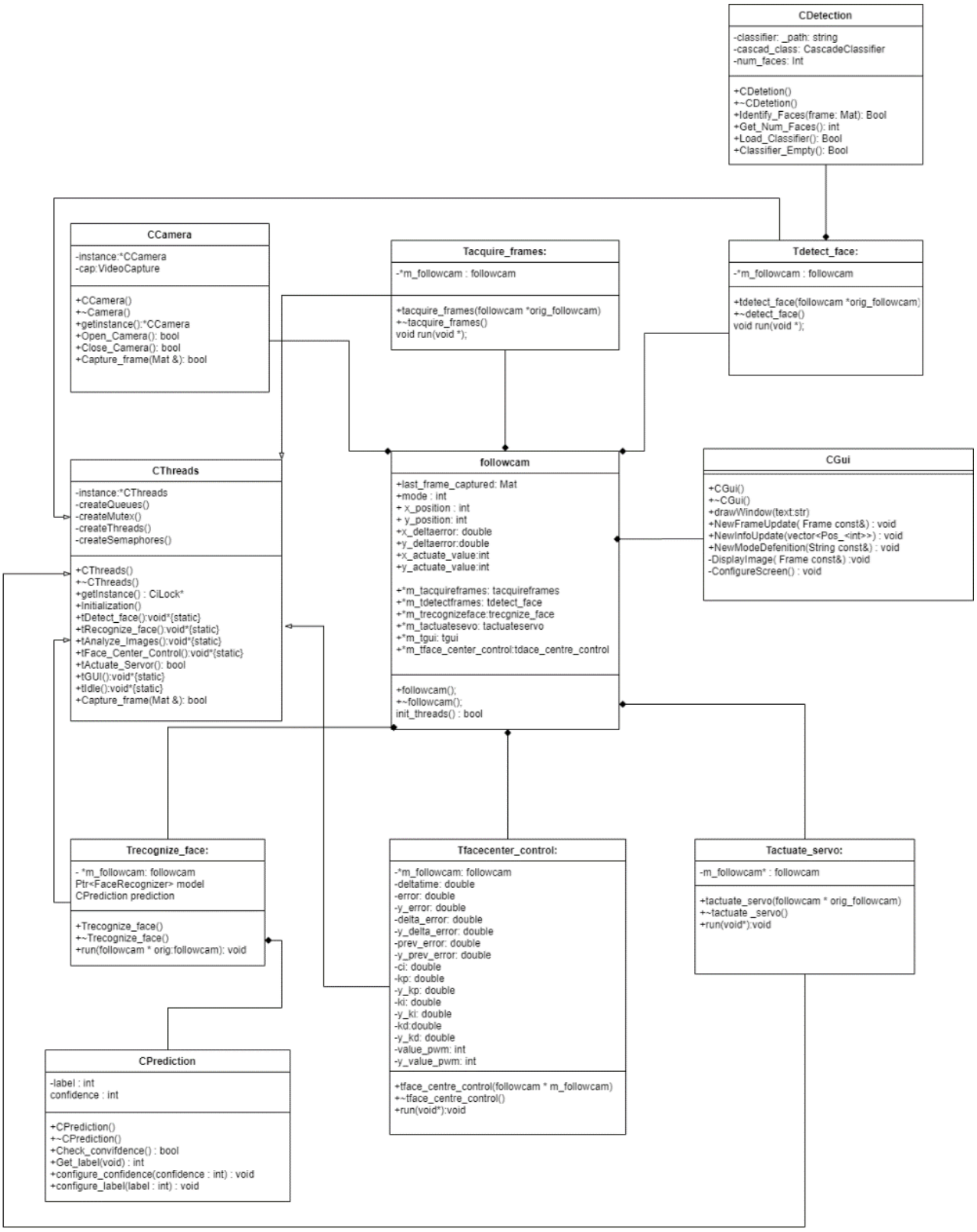


Figure 74 - Gantt Diagram

8.2 Classes Diagrame



8.3 References

-Embedded Software book - by Jean J. Labrosse, Jack Ganssle, Robert Oshana

-The Linux Programming Interface - by Michael Kerrisk

-aishack.in/tutorials/servo-motors/

-wikipedia.org/wiki/Device_driver

-elinux.org/Buildroot:DeveloperDaysFOSDEM2014

-wikipedia.org/wiki/C_(programming_language)

-wikipedia.org/wiki/C%2B%2B

-OpensCV.com

-https://circuitdigest.com/microcontroller-projects/raspberry-pi-and-opencv-based-face-recognition-system

-https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9