Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

---

Embedded Systems

Collision Avoidance and Transit Signals Detection Device

Professor: Adriano Tavares

Authors: João Faria A85632
Tiago Costa A86122

February 23, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

With the ever-evolving environment in this day and age, some efforts were being made regarding the self-driving capability.
Moreover, with the necessity of mass promoting autonomous driving, in this course, the system developed will facilitate that reality by being able to detect transit signals and by avoiding obstacle collision in real-time.
That sensed information is to be sent to the driver by the infotainment system, a display attached to the Raspberry.
The developed product application is twofold. Firstly, in an industry environment, it will be able to detect and recognize QR codes in real-time.t Secondly, after proving the initial concept, one will feed a pre-trained neural network with a data-set composed with a few different transit signals, for example, stop, blue crosswalk and prohibition to also detect and recognize them in real-time an alert the user, the driver, with appropriated text and audio alert of the respective signal.
Regarding obstacle avoidance, distance sensors will be used for obstacle detection and send commands to stop the actuating part of the target navigation system (robot vehicle prototype).

Figure 1.1: A car with a transit sign recognition system

1.2 Problem Statement Analysis

After considering the problem the next step is to identify the features of the problem by detailing what entities are involved and which relations they share. The centre of data processing (in ESRG course) is Raspberry Pi that collects information about the camera, distance sensors and transmits it to speakers, stm, and display. All of that is represented in the diagram below.

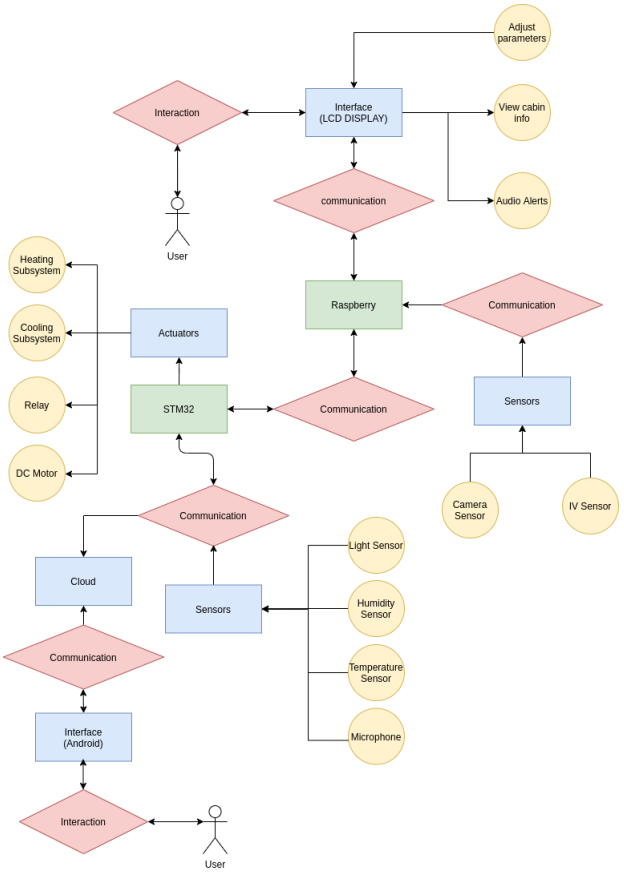


Figure 1.2: Problem Statement Analysis

1.3 Market Research

1.3.1 Target Market

The product developed will aid navigation for all types of drivers.
By combining transit signal detection and recognition with distance sensors the user (driver) can sense near obstacles approaching while it's informed about specific transit signals, increasing safety in the driving experience overall.

1.3.2 Market Growth and Concentration

Figure 1.3 depicts the market concentration for Transit Sign Detection and Recognition (TSDR) and figure 1.4 displays its concentration, erring towards a consolidated type of market.[1]

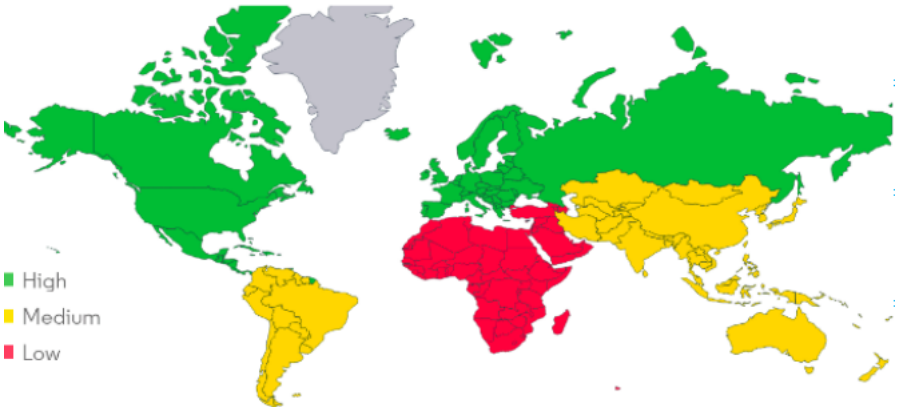


Figure 1.3: TSDR market growth ranging from 2020 to 2025

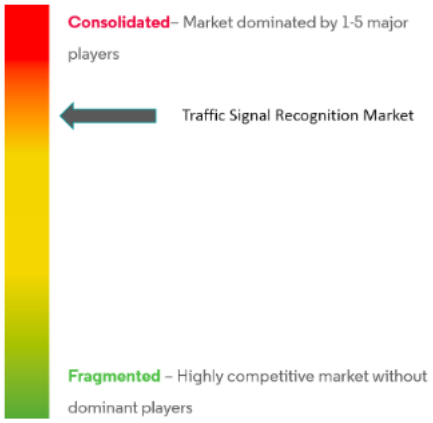


Figure 1.4: Market concentration

1.3.3 What is in the market?

After some research being made in the TSDR area one encountered the following product suited for Audi:[2]

Figure 1.5: Audi A7 Transit Sign Recognition

Pros:

- Effective transit speed limits detection and recognition;
- Lane detection;
- Transit signs can be updated to local system;

Cons:

- Expensive, up to 1000$ with camera and support apparatus;
- Does not warm about car proximities;
- Only for audi vehicles;

1.3.4 Added Value

Regarding Advanced driver-assistance systems, or ADAS, one found more solutions in the TSDR department, specifically for Ford, and Fiat vehicles. Yet, since this is pretty much a novel niche, there is not much diversity for general car transit sign recognition models, and with that in mind, one seeks to produce a **low-cost**, **easy to mount device for any vehicle**.

1.4 System

1.4.1 System Overview

To read and control the forementioned features, the input/output list is below described:

Inputs:

- Distance sensor, to sensor obstacules near the vehicle;
- Camera, to detect and recognize the target and transit signals;
- Display, allowing user input interation;

Outputs:

- Speakers, to transmit audio alerts back to the user;
- Display, works as an output too, displaying car signals, both inside and outside;
- STM-32, to send control commands to the respetive actuators;

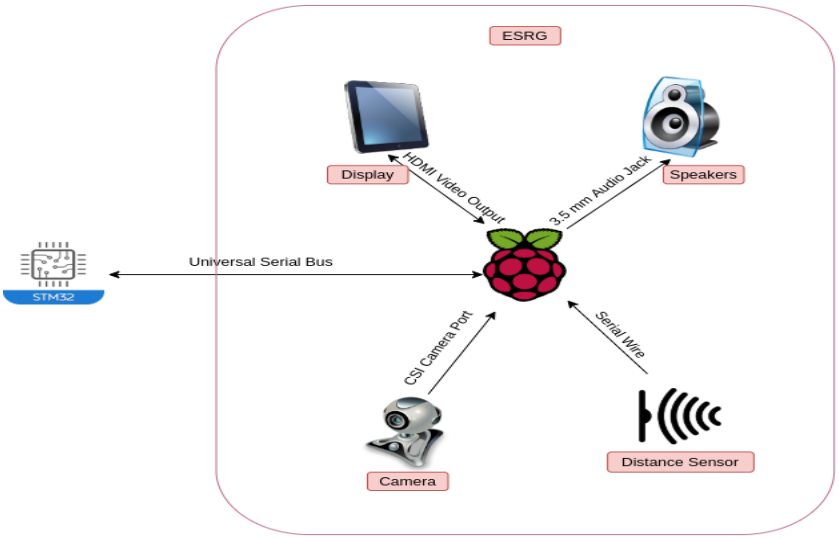


Figure 1.6: System Overview

1.4.2 Requirements and Constraints

Here we have 2 different types of requirements, functional requirements and non functional requirements. The functional requirements must specifies what the system is capable of doing and the non functional requirements specifies how the system will perform a certain action.

Project Requirements

Functional Requirements

- Avoid obstacule collision;
- Detect and recognize transit signals via camera vision;
- Displaying back to user driving related information inside and outside car cabin;
- Reproduce audio alerts;

Non Functional Requirements

- Low-cost;
- Responsive;
- Reliable;
- Interoperability;
- Power Efficient;
- Adaptability;

Project Constraints

The constraints define the product limitations. They can be technical and non-technical. Technical constraints are associate with the technical part of the project whilst non-technical ones are project management related.

Technical Constraints

- Use RaspberryPi V4 board;
- Use pthreads;
- Use object-oriented programming languages;
- Embedded Linux by buildroot;
- Cross compiling toolchain;
- Device Drivers;

Non-Technical Constraints

- Group composed by 2 members;
- Deadline at end of semester;
- Limited budget;
- Time to prototype;
- Soft Real-Time System;

1.4.3 System Architecture

Hardware Architecture

In this section, we take a view of Hardware Architecture and how the inputs and outputs are grouped and connected to Raspberry V4.

Image data, distance sensor and touchscreen input are multiplexed into the Pi, sending then commands back to STM for motor actuation and for displaying and reproducing driving-related signals. Sensors data from STM-32 are displayed too. Figure below illustrates that:

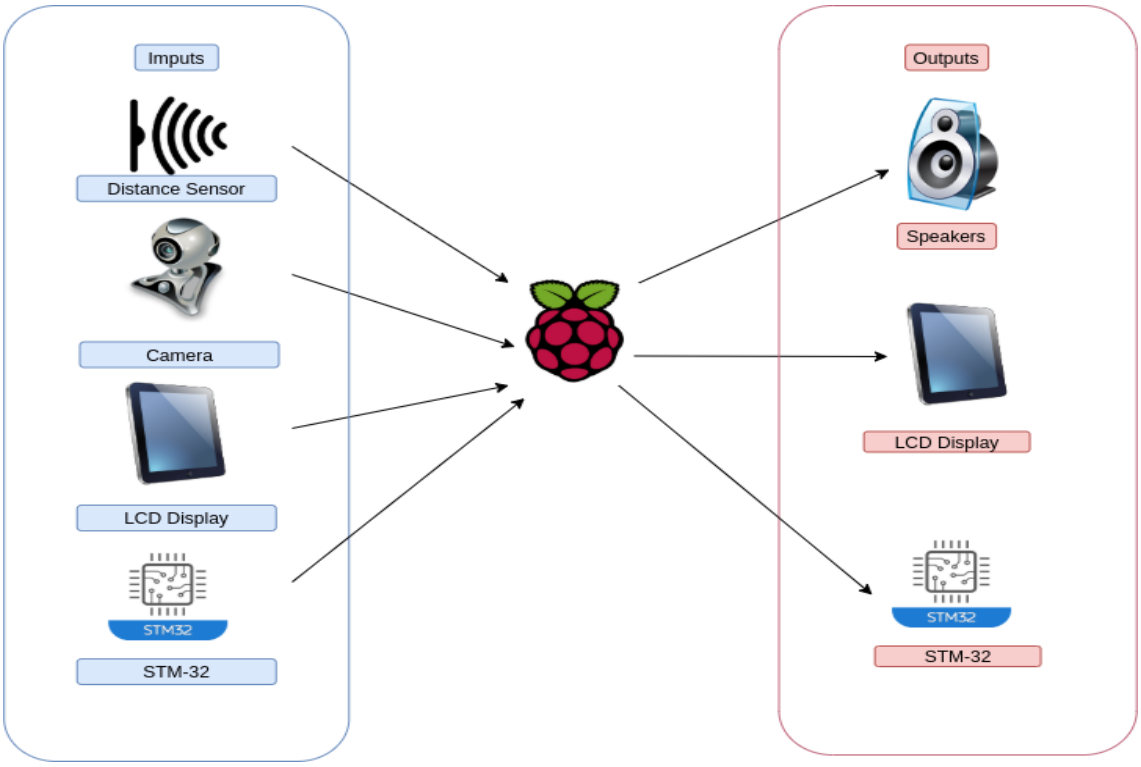


Figure 1.7: Hardware Architecture

Software Architecture

In the software architecture, specifically on the lower layer are represented the device drivers required for camera, USB, distance sensors as well as for the Touchscreen display and Audio.

The middle layer consists of a data acquisition/processing component, pthreads, Tensorflow and openCV for image processing.

Finally, the top application layer is composed of the reinforcement learning, GUI and sound output part.

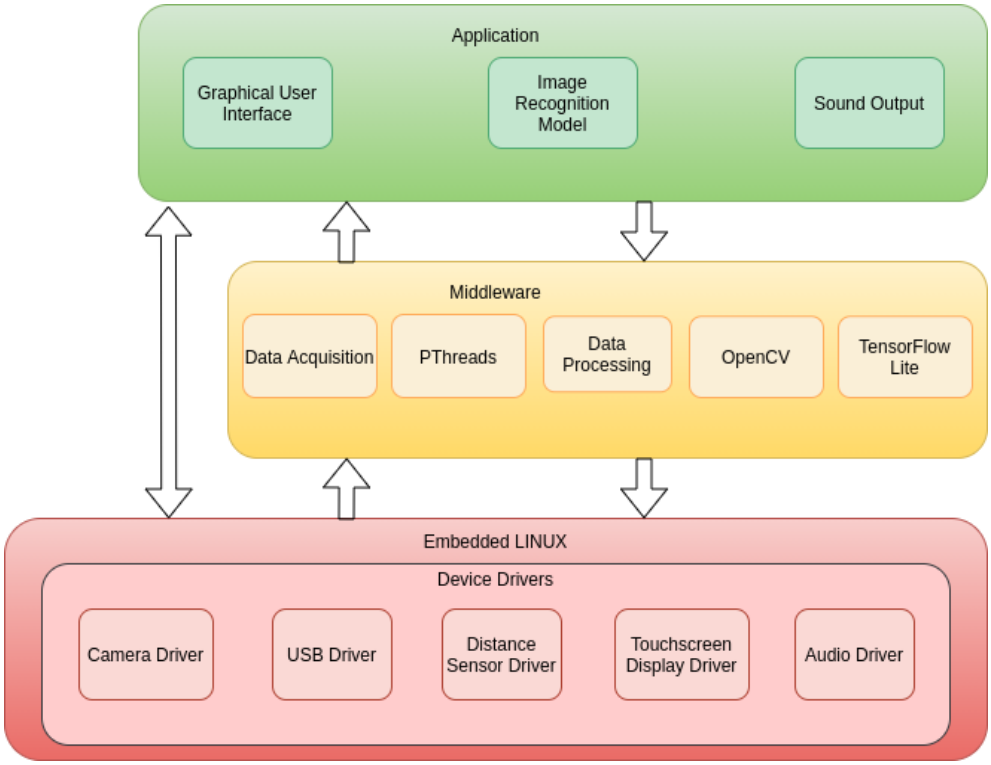


Figure 1.8: Software Architecture

Chapter 2

Analysis

2.1 Machine Learning (supervised)

Figure 2.1 is an overview of the steps taken into account from the moment a picture is taken to its corresponding alert display on the touchscreen dashboard. The chosen framework is Tensorflow [3] for training the ML model in the local computer, which then will be deployed in raspberry after the training is complete, to make inferences in real-time, figure 2.2.

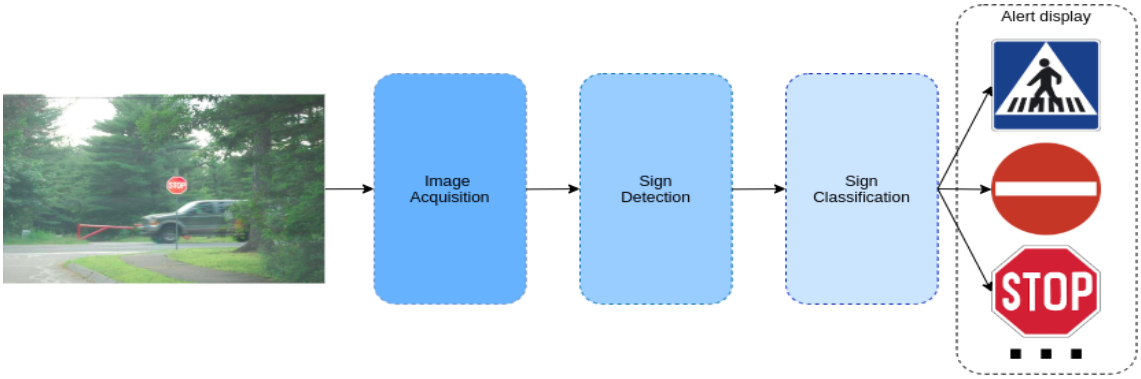


Figure 2.1: TSDR Overview

2.1.1 Framework

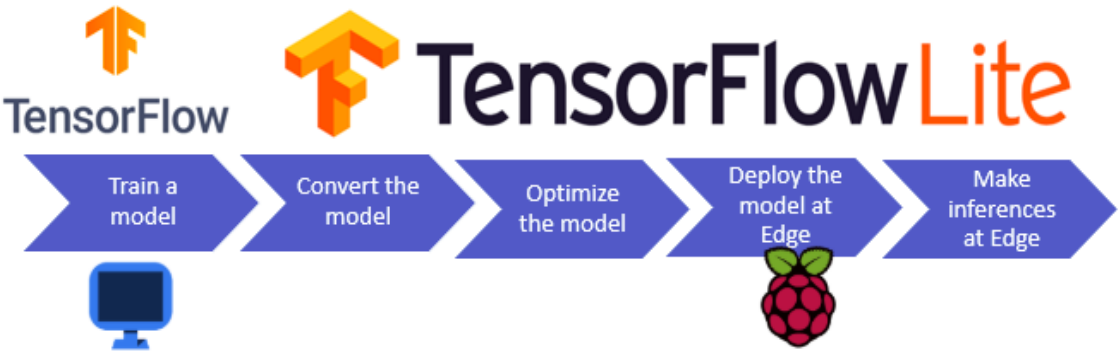


Figure 2.2: Tensorflow workflow: from training to inference

2.2 Main Events

The system has two types of events, asynchronous when the user changes reference variables or when the camera reads a signal or obstacle, and synchronous when the timer activates the camera or the remote system (STM32) sends the value of the variables.

The answer to each event is described in the image below.

Event	System Response	Source	Type
Set References Sensors Data	Send New Values to STM	User	Asynchronous
STM Data Received	Display Sensor Inputs	STM32	Synchronous
Obstacules Detected	Show Audio Alerts & Send Motor Actuation Commands to STM	Camera Vision System	Asynchronous
Image Sampling Time Elapsed	Takes and stores a picture on local system database	Camera Vision System	Synchronous
Image Available	Process Image for Detection/Recognition of Transit Signals	Camera Vision System	Asynchronous
Signal Detected	Display an Corresponding Sign Alert	Camera Vision System	Asynchronous

Table 2.1: Vision System Events Table

2.2.1 Use Cases

The local system interacts with the user and the remote system, with the user being able to change the reference variables and receive system alerts.
The remote system interacts with the local system receiving the reference variables and the information to stop, giving back the value of the variable measures to be transmitted to the user.

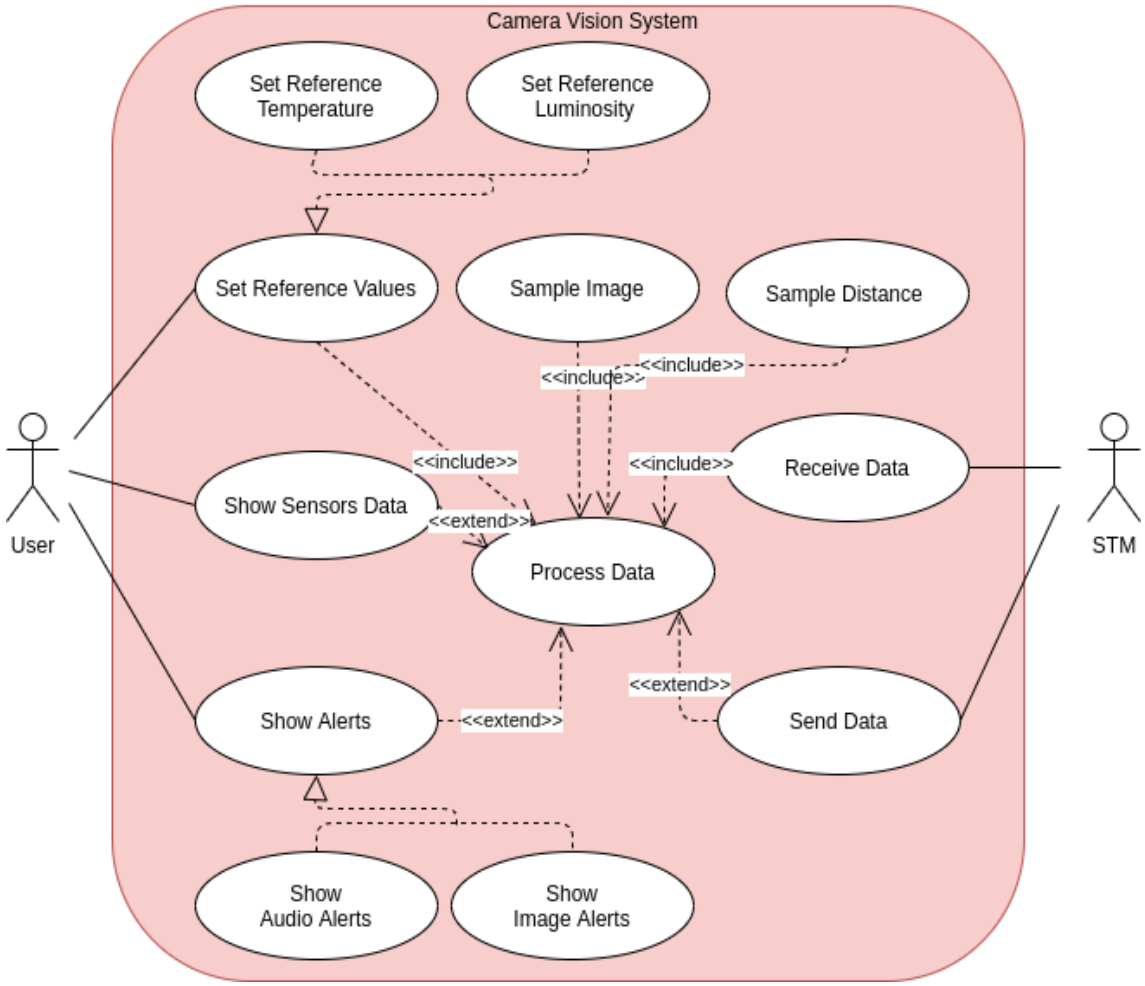


Figure 2.3: Use Cases Diagram for Vision System

2.2.2 State Chart

After initialization, the system is waiting for the timer to make the image acquisition that is later processed. When in Idle state, it can be interrupted to read data transmitted by the remote system, or by the user when he wants to change the reference variables of the remote system, and this information will be processed.

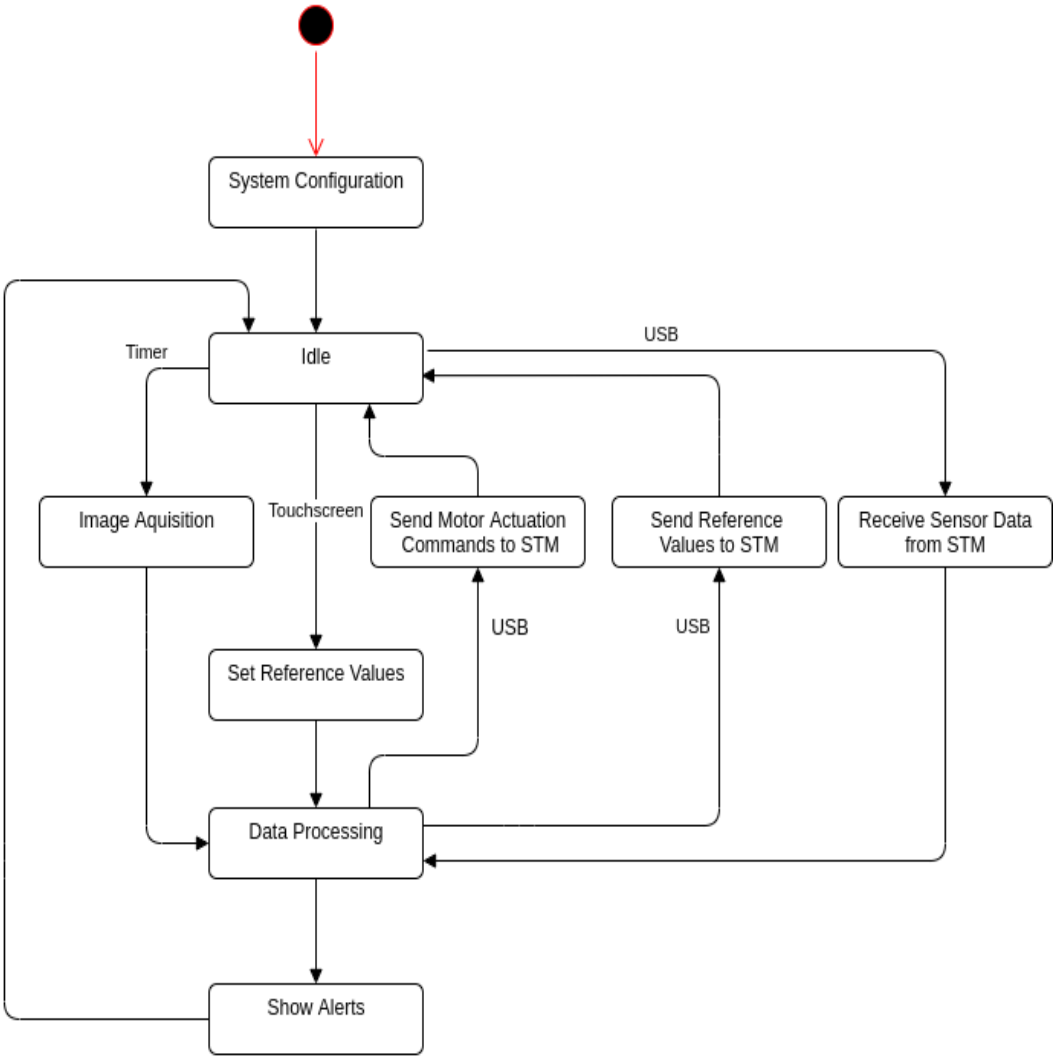


Figure 2.4: State Chart Diagram for Vision System

2.2.3 Sequence Diagram

In figure 8, we can see how the image capture trigger and its processing is done. We can see if a signal is recognized the information will be transmitted to the user, and in the case of an obstacle or a stop, the car will also stop.

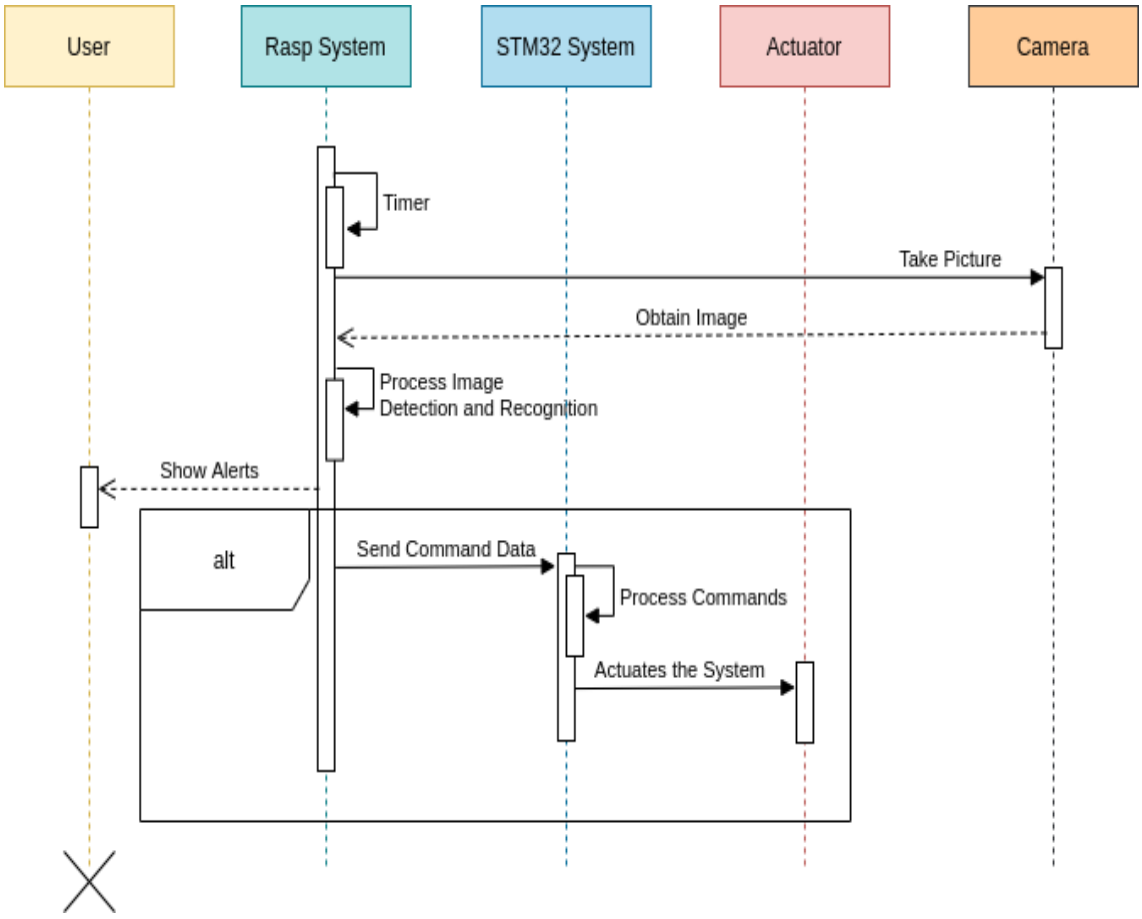


Figure 2.5: Sequence Diagram for Vision System

2.2.4 System Stack Diagram

The system consists of the main processing unit (Raspberry Pi 4B), a camera, an infrared sensor, a touchscreen display, and a speaker.
The camera, placed on the dashboard, will collect images of the road at regular intervals, which will later be analyzed in order to understand if there are traffic signs or obstacles on the road, further distinguishing the different signs according to a database created and transmitting its meaning in auditory and visual form. If it is a STOP signal, for example, or an obstacle it will transmit information to the control unit (STM-32) so that the car stops, while the STOP information is shown.
The touch screen will be used by the user to adjust the comfort of the passenger compartment.

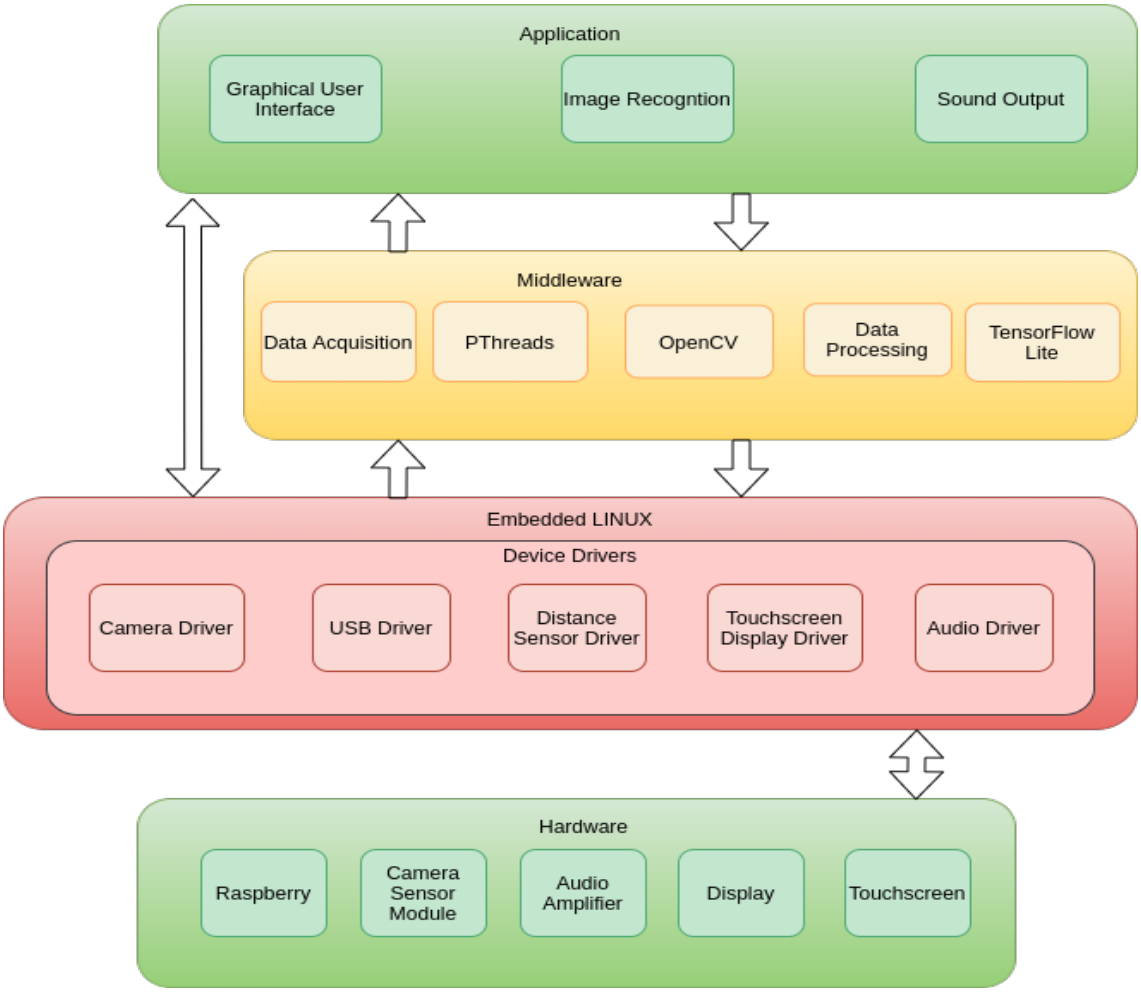


Figure 2.6: Stack Diagram for the Vision System

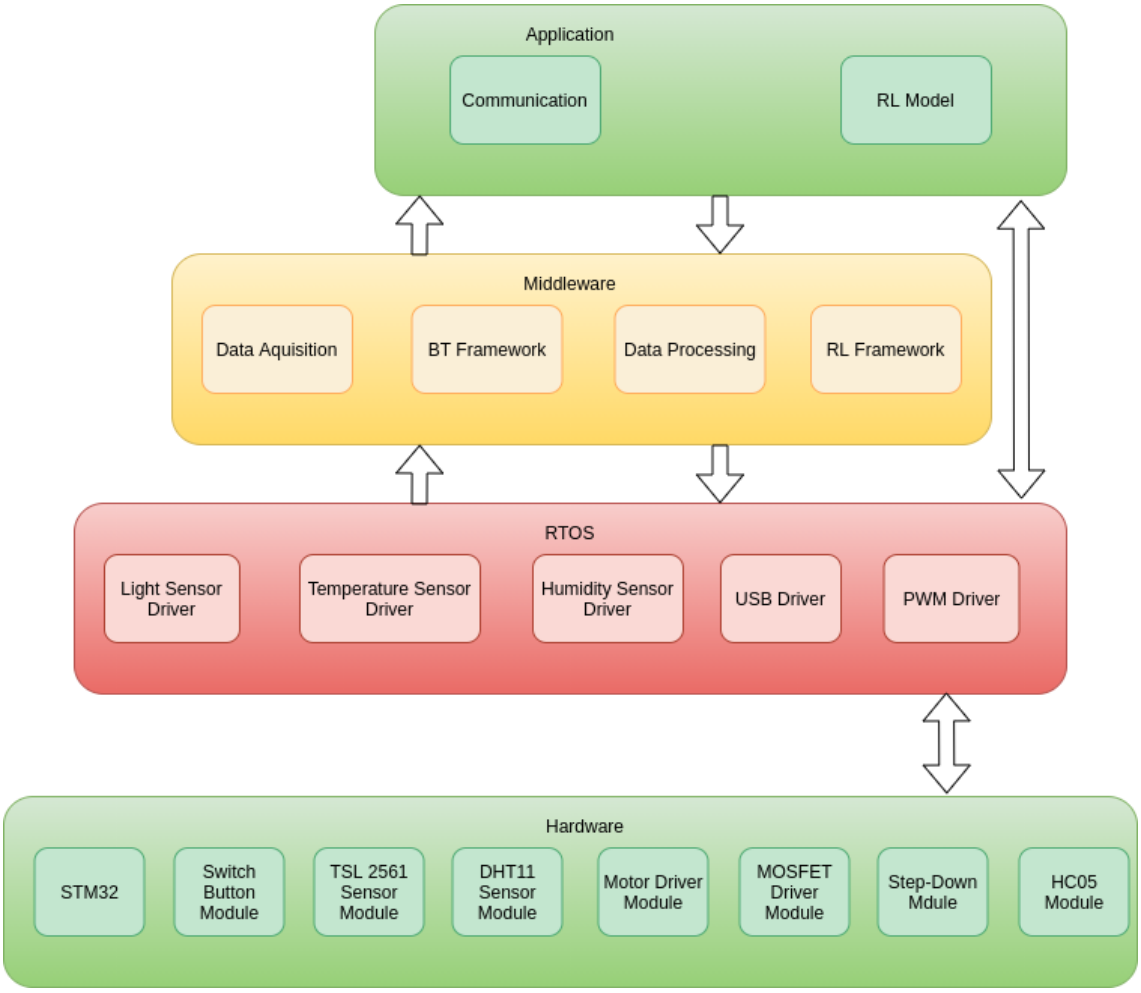


Figure 2.7: Stack Diagram for Cabin System (developed in Project1)

Chapter 3

Design

3.1 Tools and COTS

Tools

For this project one will use the following tools:

- Tensorflow is a machine learning framework utilized for the training the neural network utilized for transit sign detection and recognition. Tensorflow lite is the lightweight option for embedded systems;
- OpenCV is framework specifically for computer vision to utilize together with Tensorflow since the latter object detection examples use matplotlib library for image display and this framework is easier to work with and is less error-prone in that regard;[4]
- CUDA and cuDNN for running Tensorflow on a Graphics Processing Unit (GPU);
- QTcreator for graphical user interface associated with sensor data presentation to user in touchscreen display;
- Buildroot is a tool that simplifies and automates the process of building a complete Linux system
- Google Sound API for an embedded system, using cross-compilation;[5]

3.1.1 COTS

The Commercial-Off-The-Shelf (COTS) used for this project used are:

- POSIX threads, to emulate paralellism. Besides, is a constraint in the developed project;
- QT Framework;
- ALSA API to manage sound;

3.2 Hardware Specification

In the following sections, the hardware required for the project will be explained, as well as its main features and the interaction with other components.

3.2.1 Raspberry Pi

The development board used in this project will be Raspberry pi 4b with 2Gb of RAM. The board uses an Quad core ARM Cortex-A72 from Broadcom, with maximal operation frequency at 1.5GHz. Some of the features are presented below:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- 40 pin GPIO header
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- Micro-SD card slot
- 5V DC via USB-C conector
- Size (L x W x H): Approx. 9 x 6 x 2 cm;

3.2.2 Camera

To make the image collection will be used a 5 Mp camera with a sensor that allows night vision. Some of the main features presented below:

- Shot:1/4 5M;
- Aperture: 2.9;
- Focal length: 3.29;
- FOV: 65 degree;

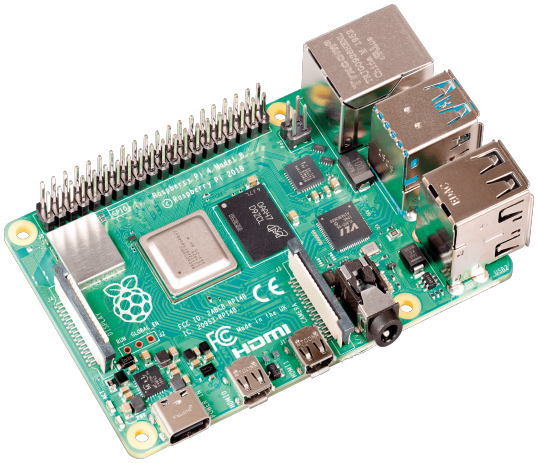


Figure 3.1: Raspberry pi 4b

- Sensor type: OmniVision OV5647 Color CMOS QSXGA (5-megapixel);
- Sensor size: 3.67 x 2.74 mm (1/4" format);
- Pixel Count: 2592 x 1944;
- Pixel Size: 1.4 x 1.4 um;
- Lens: f=3.6 mm, f/2.9;
- Angle of View: 54 x 41 degrees;
- Field of View: 2.0 x 1.33 m at 2 m;
- Full-frame SLR lens equivalent: 35 mm;
- Fixed Focus: 1 m to infinity;
- Video: 1080p at 30 fps with codec H.264 (AVC) Up to 90 fps Video at VGA;
- Size (L x W x H): Approx. 2.5 x 2.5 x 1 cm;

Interface

For the interface between the camera and the Raspberry, will be used the flat cable, it will be connected to te CSI camera port on the Raspberry.

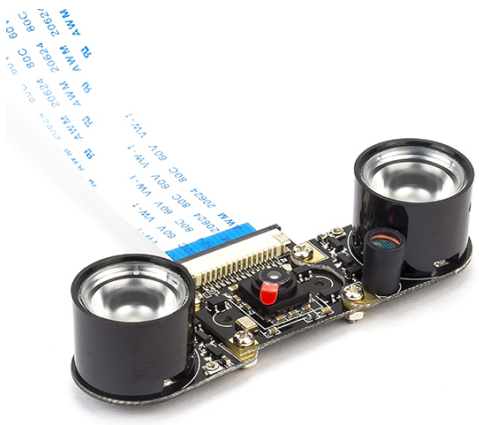


Figure 3.2: Camera Module

3.2.3 Speaker

To promote safe driving, and in such a way that the driver does not need to look away from the road, the system will transmit audio alerts. A speaker will transmit this information. Some of the main features presented below:

- Resistence: 2 ohm;
- RMS Output Power: 4W;
- Size (L x W x H): Approx. 5 x 5 x 4 cm;

Interface

For the interface between the Speaker and the system, two pins need to be used. Next we present the pins assigned for that interaction.

Speaker Pin	Amplifier Pin	Pin func.	Comment
+	Out+	Power Pin	Power Sound
-	Out-	Power Pin	Power Sound

Table 3.1: Speaker Conection

3.2.4 Audio Amplifier

To be able to hear system alerts, even in noisy environments, the sound must be amplified. The amplifier used is the TDA 2030. Some of the main features presented

Figure 3.3: Speaker

below:

- Input voltage: 6 to 12V;
- Power Output: 18W;
- Mono Output;
- 10 k adjustable resistance, to adjust the volume;
- On board power indicator light;
- Size (L x W x H): Approx. 3.2 x 2.4 x 2.2 cm;

Interface

For the interface between the Amplifier and the Speaker, some pins need to be used. It will be connected to the Raspberry using the audio port. Next we present the pins assigned for that interaction.

3.2.5 Touchscreen Display

To view and change, the value of the variables that control the comfort of the cabin, as well as to show the recognized signals, a touch screen display will be used that will allow the user an easy interaction with the device. Some of the main features presented below:

- Diagonal: 3.5" (8.89cm);

Figure 3.4: TDA2030 Audio Amplifier

Amplifier Pin	Speaker Pin	Pin func.	Comment
Vcc		12v	Positive Power Supply
GND		GND	Ground
In		Signal	Positive Audio Signal
Out+	+	Power	Power Sound Output
Out-	-	Power	Power Sound Output

Table 3.2: Amplifier Conection

- Resolution: 480x320 pixels;
- Touchscreen Type: Resistive;
- Touch screen controller: XPT2046;
- Colors: 65536;
- Backlight: LED;
- Connection: GPIO header;
- Aspect ratio: 8:5;
- Size (L x W x H): Approx. 8.5 x 5.6 x 0.5 cm;

Interface

For the interface between the Display and the Raspberry, some pins need to be used. Next we present the pins assigned for that interaction.

Figure 3.5: Display

Display Pin	Raspberry Pin	Pin func.	Comment
Vcc	5v	5v	Positive Power Supply
GND	GND	GND	Ground Power Supply
DC	18	Data/Comand	Data/Comand Selection
RST	22	Reset	Reset
D_CS	24	Chip Select	Display Selection
T_CS	26	Chip Select	Touchscreen Selection
MOSI	19	Data in	Data Transmition
MISO	21	Data out	Data Transmition
SCK	23	Clock	Clock Source

Table 3.3: Display Conection

3.2.6 Power Supply

To power all the systems will be used a battery with a voltage of 12v. Some of the main features presented below:

- Output voltage 12V;
- Output current 7A (maximum);
- Weight: 2.8 Kg;
- Size (L x W x H): Approx. 15 x 6.5 x 9.5 cm;

Interface

Figure 3.6: Battery

For the interface between the battery and the system, two pins need to be used. Next we present the pins assigned for that interaction.

Battery Pin	Systems Pin	Pin func.	Comment
12v	In+	Power Pin	Power Supply
0v	In-	Power Pin	Power Supply

Table 3.4: Battery Conection

3.2.7 Step-Down

To power the Raspberry, a step-down module will be used to reduce the battery voltage from 12v to 5v. Some of the main features presented below:

- Input voltage 3.2V 40V;
- Output voltage 1.25V 30V;
- Output current 3A (maximum);
- Conversion efficiency is 92
- Output ripple <30mV;
- Switching frequency 65KHz;
- Operating temperature -45°C +85°C ;

- Size (L x W x H): Approx. 4.3 x 2.2 x 1.4 cm;

Figure 3.7: Step-Down

Interface

For the interface between the Step Down module and the Raspberry, some pins needed to be used. Next we present the pins assigned for that interaction.

Step-Down Pin	Raspberry Pin	Pin func.	Comment
IN+		12v	Positive Power Supply
IN-		GND	Ground
OUT+	5v	5v	Positive Power Supply
OUT-	Ground	GND	Ground

Table 3.5: Step-Down Conection

3.3 Software Specification

Device Drivers

Device drivers are a very important class/module on the kernel space. It's through them that we can access the system resources on the user space (with an application) without needing to know all the details about the resource we're requesting access. There are character device drivers, block device drivers and other types of device drivers that define a set of functions (APIs) provided by the Linux kernel to the user space for the interaction with hardware, allowing data transfer between kernel space and user space.

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
Remove a device	rmmod()	module_exit()

Table 3.6: API of a device driver - Events and Kernel Functions

APIs functions of a device driver should have a duality quality, that is, for each functionality there is a dual that performs the reverse operation: open/close, read/write, initialize/remove module. Also they should have defined an user defined function ioctl() for handling specifical needs not already covered with the ones depicted in table 3.6.

The module init() function registers the struct file operations with VFS (Virtual Filesystem Switch) and reserves a major number to the device. The VFS is a table that stores modules functions that perform the events requested from a device. It allows dinamically module insertion on /dev of the kernel. The major number maps the I/O request to driver code, associating driver with the respective device.

3.3.1 Processes and Threads

A multiprocessing OS like Linux can have more than one process executing at any given time. That can be performed with true paralellism where multiple processes run at the same time in multiple processors or fake paralellism or concurrent if multiple processes in a run state can be swapped in and out by the OS.

A process is, in simple an instance of a program that is being executed. Each process has its own file handlers, registers, along with a program counter (Instruction Pointer), a process id, a process group id, a process stack, one or more data segments, a heap for dynamic memory allocation and a process state (running, ready, waiting and zombie).

The possible states in a given process are:

- Running - the process is running or ready to run, waiting for dispatcher to handle it to the CPU;
- Waiting - the process waits for an event or system resource. In this state, processes can be interruptible or not;
- Stopped - the process has been stopped, usually with a signal;
- Zombie - a dead process that was halted but still has a entry in a process table;

A thread is a subset of a process that encapsulates a flow of control in a program. A process can have multiple threads, and each one of them has its own thread ID, stack and stack pointer, pointer, signal mask, program counter and registers.

However, if they are within a process, they share resource handlers, static and dynamic memory segments and code.

They differ from process in that:

- process are usually independent and threads are dependent of the one they are within;
- process can only communicate with inter-process communication (IPC) mechanisms;
- threads of a process share its state, memory and other resources;
- the address space is shared between threads, contrary to processes;
- context switching is faster and has less overhead in threads than in process;
- threads can live only in user space;

3.3.2 Daemons

A Daemon is a background process that runs without user input. It can provides services such as:

- System Loggers;
- Web Servers;
- Database Servers;
- ...;

Daemons are orphan processes that inherit and detach from init process that is a process group and session leader with no controlling terminal.

Process groups are processes that share the same process group identifier (PGID). A process group has a process group leader, which is the process that creates the group and whose process ID becomes the process group ID of the group. A new process inherits its parent's process group ID and a process can leave the group by terminating or joining another group.

Sessions are a collection of process groups, whose session identifier (SID) is the process identifier (PID) responsible for the session creation, the session leader. A new process inherits its parent's session ID. All processes in a session share a single controlling terminal, established when the leader opens a terminal device.

To create a daemon there are some steps needed to be followed:

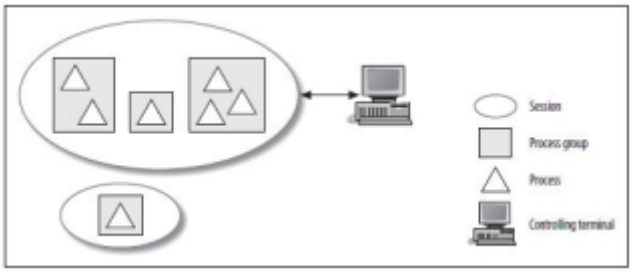


Figure 3.8: Sessions and Process Groups Representation. The daemon is the triangle that is isolated from the controlling terminal.

- create an orphan process - using **fork()** and **exit()** the parent process so that its closer to lose dependency to a controlling terminal;
- create a new session - using **setsid()** to completely dissociate of a controlling terminal;
- change to the root directory - using **chdir("/")** to safely unmount the system from process that prevent it and whose current directory is on a mounted file system;
- change child umask - using **umask(0)** making newly created directories and files readable, writable and executable for every process;
- close file descriptors - using **close(STDxxx_FILENO)** for closing, xxx can mean for input, output or error. Put each one inside of a different close() call. That is no reason for keeping open descriptors from parent process;

Note that the first three steps should be verified, i.e., if they execute successfully, or if they return an error.

Once a daemon calls setsid, it no longer has the controlling terminal and so it has nowhere to send output that would normally go to stdout or stderr (such as error messages). Fortunately, the standard utility for this purpose is the syslog service, provided by the system logging daemon, syslogd. This library provides API for open, close and write to a log file with: openlog(), closelog() and syslog(). The former two are optional to include in a program. With that utility, it is possible to save log messages from other daemons, as well as attach to any message the daemon PID with LOG_PID or identify messages priority with LOG_INFO, LOG_WARNING,...

Finally, it is possible to communicate with daemons via signals, even tougth they have no controlling terminal associated to them. That will be covered in the next section.

3.3.3 Signals

A signal is a notification to a process that an event has occurred. Signals can be sent from the kernel space to a given process, between processes or even to the process itself. Signals occur when a program tries to perform a division by 0, referencing an inacessible buffer, a timer went off or even when user types *ctrl-C*.

When a signal occurs, it can be:

- ignored;
- accepted and perform its default action;
- handled by a user-defined handler function. The program jumps to the handler when immediately signaled;

Signals can be distinguished between standard and realtime signals, and are identified with an unique small integer, starting from 1. The former ones, whose source is the kernel, are used for process notifications relative to a given event, and numbered from 1 to 31. The latter 33 of them can be used for application-defined purposes.

The range of supported real-time signals is defined by the macros SIGRTMIN and SIGRTMAX. Programs should never refer to real-time signals using hardcoded numbers, but instead should always refer to real-time signals using the notation SIGRTMIN+n, and include suitable (run-time) checks that SIGRTMIN+n does not exceed SIGRTMAX. When real-time signals are unhandled, their default action is to terminate the target process. They are different from the initial ones by the following:

- Multiple instances of real-time signals can be queued. By contrast, if multiple instances of a standard signal are delivered while that signal is currently blocked, then only one instance is queued;
- If the signal is sent using sigqueue(), an accompanying value (either an integer or a pointer) can be sent with the signal. If the receiving process establishes a handler for this signal using the SA SIGINFO flag to sigaction(), then it can obtain this data via the si value field of the siginfo t structure passed as the second argument to the handler. Furthermore, the si pid and si uid fields of this structure can be used to obtain the PID and real user ID of the process sending the signal;
- Real-time signals are delivered in a guaranteed order. Multiple real-time signals of the same type are delivered in the order they were sent. If different real-time signals are sent to a process, they are delivered starting with the lowest-numbered signal. (I.e., low-numbered signals have highest priority.) By contrast, if multiple standard signals are pending for a process, the order in which they are delivered is unspecified;

Lastly, if both standard and real-time signals are pending for a process, Linux gives priority to standard signals.

3.3.4 Protocols

A communication protocol is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. The protocol defines the rules, syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both. In this project one will implement the following:

- **USB protocol**

  Universal Serial Bus (USB) is an industry standard that establishes specifications for cables and connectors and protocols for connection, communication and power supply (interfacing) between computers, peripherals and other computers. A broad variety of USB hardware exists, including eleven different connectors, of which USB-C is the most recent.

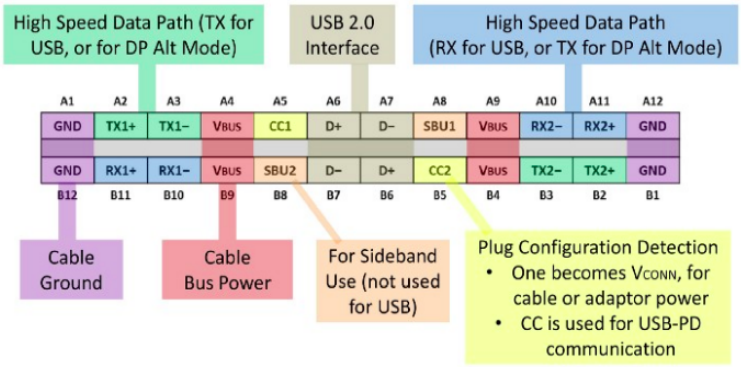


Figure 3.9: USB 3.0 Configuration Channel

On figure 3.9 [6] one can see USB 3.0 pin assignment. The packets associated with this protocol are, from first to last [7]:

– USB 3.0 General Packet Structures - Packets in USB 3.0 generally come in 3 different patterns;

– Header Packet - Header Packets consist of three parts: header packet framing, packet header, and a link control word. Note that "SHP" is a

a K-symbol which stands for "start header packet." The header is protected by a Cyclic Redundancy Check with 17-bit polynomial (CRC-16) and the link control word is protected by CRC-5 for error detection;

- Data Payload Packet - Data Payload Packets send application data and are protected by CRC-32;

- Link Command Packet - Link Command Packets are used to control various link-specific features, including low power states and flow control;

- Link Management Packets (LMP) - Link Command Packets are used to control various link-specific features, including low power states and flow control;

- Transaction Packets (TP) - Transaction Packets (TP) are a type of header packet used to control the flow of data packets end-to-end between the host and device;

- Data Packets (DP) - Data Packets (DP) are used to transmit application data and are comprised of two parts: a data packet header (DPH) and a data packet payload (DPP);

- Isochronous Timestamp Packets (ITP) - sochronous Timestamp Packets (ITP) are used to send timestamps to all devices for synchronization;

This project will utilize USB protocol for allowing communication between Raspberry and STM-32 board sensor data, from the former to the latter and any temperature reference changes and motor commands in the reverse order, providing a bidirectional channel.

- **SPI**

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems.

Figure 3.10 depicts SPI configuration where:

- Master - typically a microcontroller, is the one that configures clock and initializes communication. There is always only one master;
- Slave - the other device(s) at the receiving end, that is connected directly to master or other slaves;
- SCLK - the clock line utilize for synchronization between master and slaves;
- MOSI - "Master Out Slave In": Data output from master;
- MISO - "Master In Slave Out": Data output from slave;
- SS - "Slave Select" works as a chip select allowing the master to enable communication only with the chosen slaves by setting the line to 0 or with a falling edge;

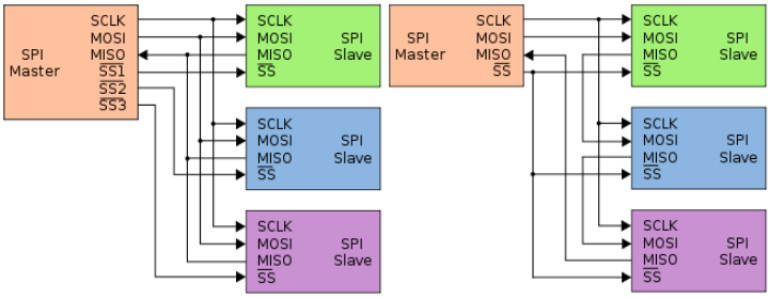


Figure 3.10: SPI independent slave (left) and daisy chain (right) configurations

On the left picture, the master talks independently with each slave, activating the corresponding SS line. Data from each shift register is interchanged in one clock cycle. On the right one, the SS line is shared between all devices and data transmited passes to each one of the slaves before finally reaching the master again. In this case, instead of copying the content of the shift register from a selected slave to the one in master, the whole chain acts as a communication shift register, where data from one device is copied to another, in a circular fashion with only one clock cycle.

This project will utilize SPI protocol for communication between touchscreen and raspberry Pi, for display sensor data, collision warnings and alerts regarding the identified transit signs.

- **I2C**

I2C (Inter-Integrated Circuit), is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial communication. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. With I2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes:

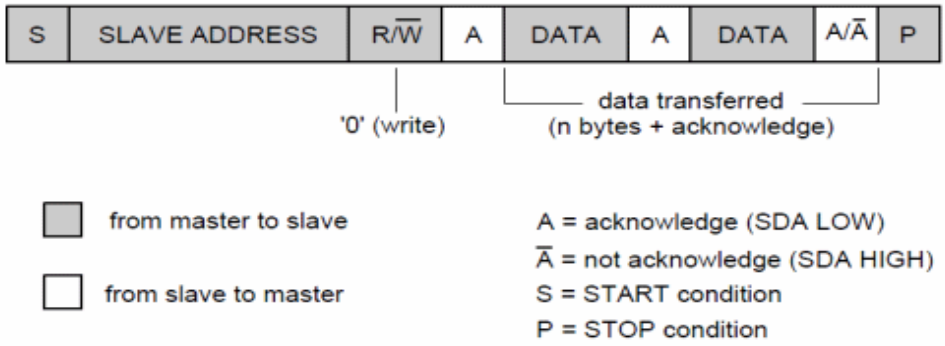


Figure 3.11: I2C packet

- **Start Condition, S**: The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low;
- **Stop Condition, P**: The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high;
- **Address Frame**: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it;
- **Read/Write Bit**: A single bit specifying whether the master is sending data to the slave (low voltage level, write) or requesting data from it (high voltage level, read);
- **ACK/NACK Bit**: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device;

The I2C protocol will be used for communication between camera and Raspberry Pi.

3.3.5 Tensorflow Lite

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep

neural networks. Tensorflow Lite is the proposed option tool for working with machine learning models inside of embedded systems. In the context of this project, tensorflow lite is the framework for exporting the pre-trained ML model of transit signs into the target device (Raspberry).

Figure 3.12: Tensorflow lite logo

3.3.6 OpenCV

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. The proposed project will use OpenCV for camera interface, providing a set of APIs from opening to taking a picture with a pi camera.

Figure 3.13: OpenCV logo

3.3.7 Neural networks and Deep learning

One component of the project is the traffic sign detection and recognition. There are several ways to implement a TSDR system, ranging from traditional computer vision solutions that uses techniques such as color thresholding and image segmentation to employing different classes of **neural networks** (NN).

Neural Networks, in this case, are simply a bunch of nodes that are interconnected with one another to loosely model the neurons in a biological brain. Neural networks learn (or are trained) by processing examples, each of which contains a known "input" and "result," forming probability-weighted associations between the two, which are stored within the data structure of the net itself.

The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and a target output. This is the error. The network then adjusts its weighted associations according to a learning rule and using this error value. Successive adjustments will cause the neural network to produce output which is increasingly similar to the target output. After a sufficient number of these adjustments the training can be terminated based upon certain criteria [9].

This is known as supervised learning, which is the area one will focus on the project. This solution for transit sign problem is within the **deep learning** subarea of machine learning, a subset of machine learning where the learning comes from using the foremen-tioned networks in areas such as computer vision, audio recognition and whatnot.

3.3.8 Transit Sign Detection and Recognition

In the proposed project, one will utilize a Convolutional Neural Network (CNN) .

CNN is a shift invariant class of deep neural networks, commonly applied to analyzing visual imagery, as its output does not depend on whether the image object to detect is on the center of the image input, for example.

As per detection and recognition, a **SSDmobileNet** will be used since its the **state-of-art solution** for implementing on **embedded systems** in **real-time applications** like driving assistance, according to this study[11]. Altought it does not have the best accuracy comparing to other alternatives mentioned in that study, it is the least computationally expensive solution in both processing power and memory footprint, and with that it is feasible to achieve higher framerates.

Adding a threshold of 60% might be a good choice for the TSDR detection model. Also, the recognized signal warning will be displayed while it is being detected.

Figure 3.14: GTSRB dataset for training SSDmobileNet

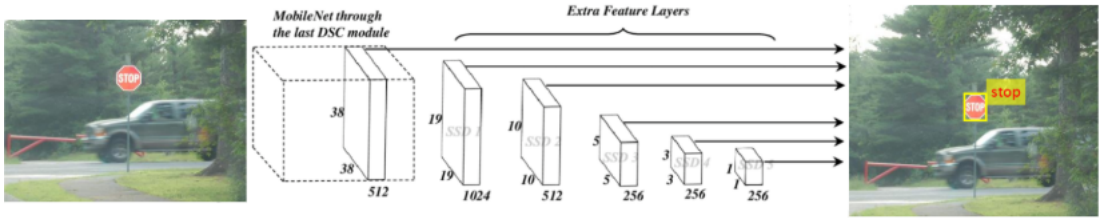


Figure 3.15: TSDR inference pipeline using SSDmobileNet

3.3.9 Vehicle Detection

For obstacule detection counterpart, more precisely, vehicle detection aiding collision warning, one will use a pre-trained SSDmobileNet CNN. The code will be implemented on python and with the forementioned frameworks this funcionality will be integrated on the system.

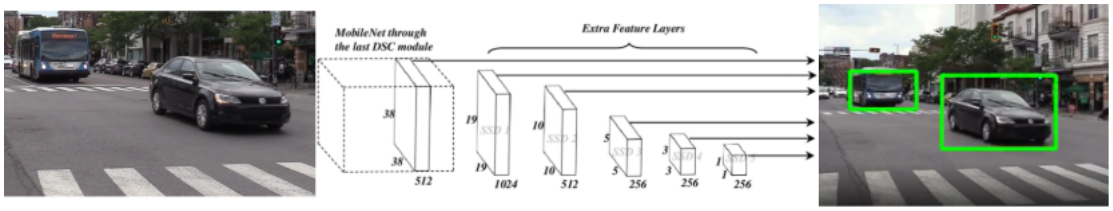


Figure 3.16: Vehicle detection inference pipeline using a pre-trained SSDmobileNet

Adding a threshold of 60% might be a good choice for the car detection model. Moreover, the collision warning will pop up if the detected car in the front has a label whose width occupies a certain percentage relative to the acquired image frame width and it is not on the located on the edges of the picture. For starting, the relative percentage will be 30% and then probably adapted on the verification phase.

3.3.10 GUI

Concerning the Graphical User Interface, one will use QTcreator. Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which simplifies GUI application development. It is part of the SDK for the Qt GUI application development framework and uses the Qt API, which encapsulates host OS GUI function calls.

Figure 3.17: Qt creator logo

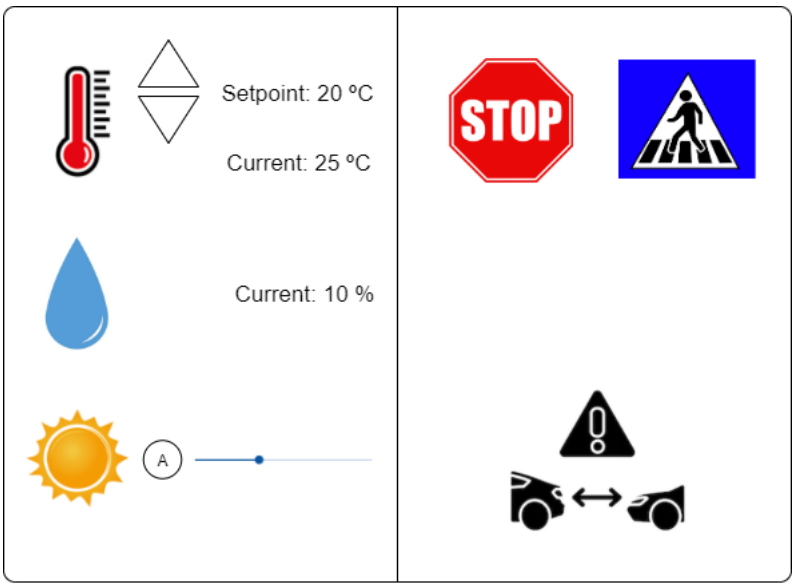


Figure 3.18: Touchscreen Display: On the left the car cabin elements are displayed; on the right, transit signals (above) and collision warnings (below) are displayed only while are being detected

3.3.11 Sound Output

Regarding the sound output, one will utilise ALSA framework for providing audio and MIDI functionality to the targeted operating system so that it is possible to reproduce hardcoded audio messages such as: "Collision Warning" or "Stop sign detected".

Figure 3.19: Alsa logo

Figure 3.20 depicts Alsa as the interface, acting as the sound driver to the sound device, the audio amplifier.

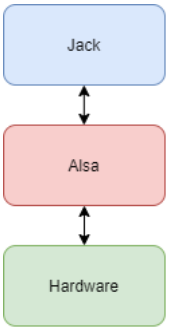


Figure 3.20: Alsa interface between jack and hardware

3.4 Tasks Overview

3.4.1 Processes and Threads

The cabin system will have the two daemon processes and one main process with the following tasks:

- **dAcquireImage** - task that peridoically captures images. It should run on a different process than main so that it does not have to be interrupted by other image processing related tasks.
- **dReceiveFromSTM** - daemon responsible for periodically acquire information sent from STM for screen display;
- **tDetect_transit_signals** - task responsible for detecting any transit signals from image captured.
- **tDetect_obstacles** - task responsible for detecting any obstacles from image captured.
- **tRecognize_transit_signals** - task responsible for recognizing any transit signals from image captured.

- **tRecognize_obstacles** - task responsible for recognizing obstacles nearby.
- **tMeasure_distance** - task that measures distance from nearby objects and is responsible for detecting if the object is in danger zone or not.
- **tProcess_alert** - task that maps recognized traffic sign to the respective symbol.
- **tSendInfoToSTM** - task responsible for sending commands to STM in case recognized obstacles are dangerously close to the ego car.
- **tGUI** - task responsible for image display of information from STM as well as transit signs and obstacles nearby recognized, and for reproducing the associated sound signals.

3.4.2 Mutexes

The following Mutexes will be implemented for :

- **mutex_signdetected** - responsible for protecting image that is a shared resouce between tDetect_transit_signals and tRecognized_transit_signals;
- **mutex_signrecognized** - responsible for protecting sign coordinates that is a shared resource between tRecognized_transit_signals and tProcess_alert;
- **mutex_obsdetected** - responsible for protecting image that is a shared resouce between tDetect_obstacles and tRecognized_obstacles;
- **mutex_obsrecognized** - responsible for protecting obstacle type and coordinates that is a shared resouce between tRecognized_obstacles and tMeasure_distance;
- **mutex_danger_collision** - responsible for protecting obstacle type and coordinates that is a shared resouce between tRecognized_obstacles and tMeasure_distance;

3.4.3 Condition Variables

Conditional Variables will be used along with the forementioned mutexes to control the flow of execution: they are:

- **cond_signdetected** - responsible for protecting image that is a shared resouce between tDetect_transit_signals and tRecognized_transit_signals;
- **cond_signrecognized** - responsible for protecting sign coordinates that is a shared resource between tRecognized_transit_signals and tProcess_alert;
- **cond_obsdetected** - responsible for protecting image that is a shared resouce between tDetect_obstacles and tRecognized_obstacles;

- **cond_obsrecognized** - responsible for protecting obstacle type and coordinates that is a shared resouce between tRecognized_obstacles and tMeasure_distance;
- **cond_danger_collision** - responsible for protecting obstacle type and coordinates that is a shared resouce between tRecognized_obstacles and tMeasure_distance;

3.4.4 Queues

The threads created communicate each other via queue messages. They will be used for passing control information, i.e. there is a message on shared memory, detected object coordinates and its type and symbols to identify which sign was detected so that proper sound could be reproduced and proper image warning to be displayed.

3.4.5 Shared Memory

Shared memory allows fast communication between processes. That is conceived by allocating a memory buffer that can be shared by the intended processes. In this case, an image will be taken by the dAcquireImage daemon and will be stored on shared memory for further access by tDetect_transit_signals, tDetect_obstacles, tRecognize_transit_signals and tRecognize_obstacles.

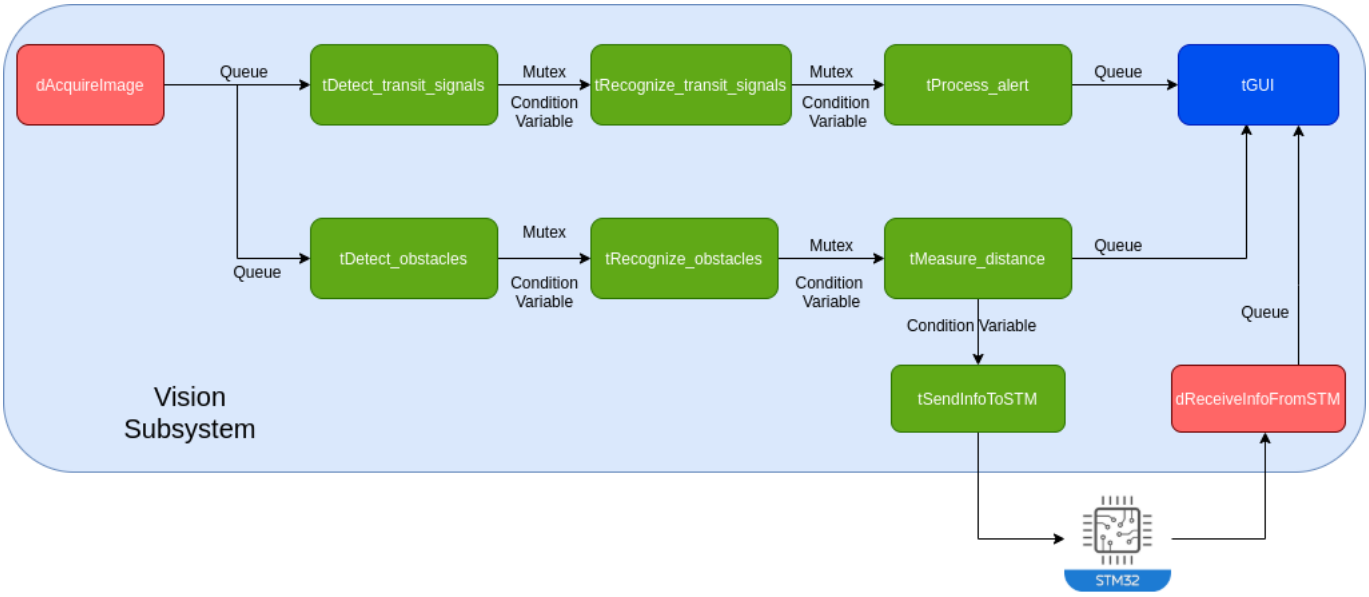


Figure 3.21: Task Overview

Here one can see the task overview of the project where all the daemons, tasks, IPC mechanisms and the two subsystems are interconnected.

3.5 Process Flowchart

dAcquireImage

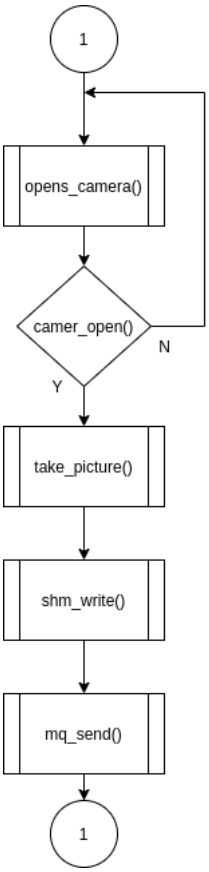


Figure 3.22: Acquire Image Daemon Flowchart

This daemon starts by opening camera and taking a picture if the former succeds. Then, writes the picture to a shared memory location to be accessed by the processing threads inside the main process. Before taking another picture, sends a message to inform there is a new image on the shared memory block.

dReceiveInfoFromSTM

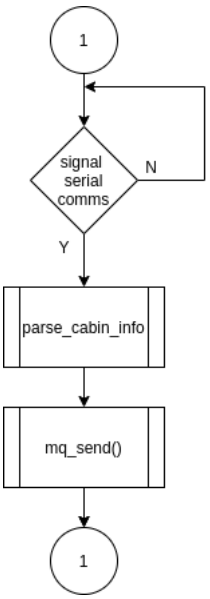


Figure 3.23: Receive Information From STM Flowchart

This daemon is responsible for receiving information on the car cabin system of the project 1 UC. When there is something on the UART, a signal is sent to the daemon to parse the message and send to queue so that the tGUI can process by displaying new values on the screen.

3.6 Task Flowchart

tDetect_transit_signals

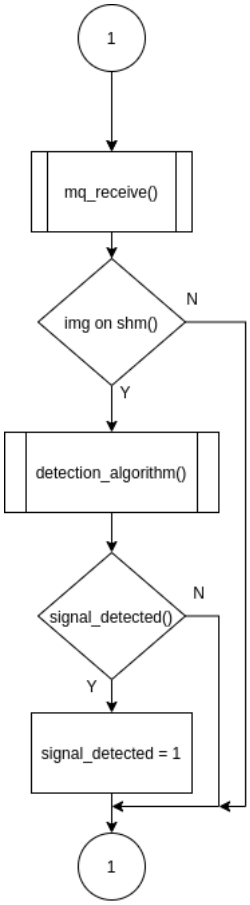


Figure 3.24: Detect Transit Signals Task Flowchart

This task detects transit signals by first receiving an indication from the daemon_-queue that there is a new image to process. It then, checks if it is stored on the shared memory buffer and if so, initiates the detection algorithm that outputs if there is a sign detected.

tDetect_obstacles

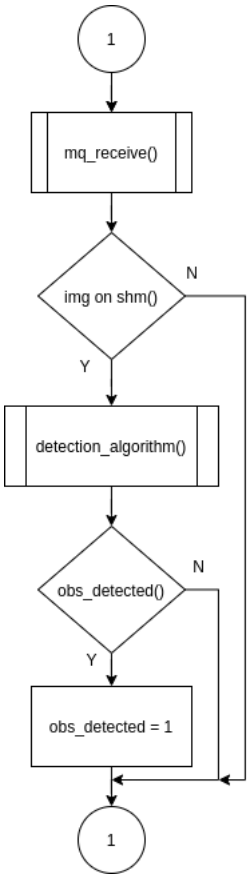


Figure 3.25: Detect Obstacles Task Flowchart

This task detects obstacles nearby by first receiving an indication from the daemon_-queue that there is a new image to process. It then, checks if it is stored on the shared memory buffer and if so, initiates the detection algorithm that outputs if there is a obstacle detected.

tRecognize_transit_signals

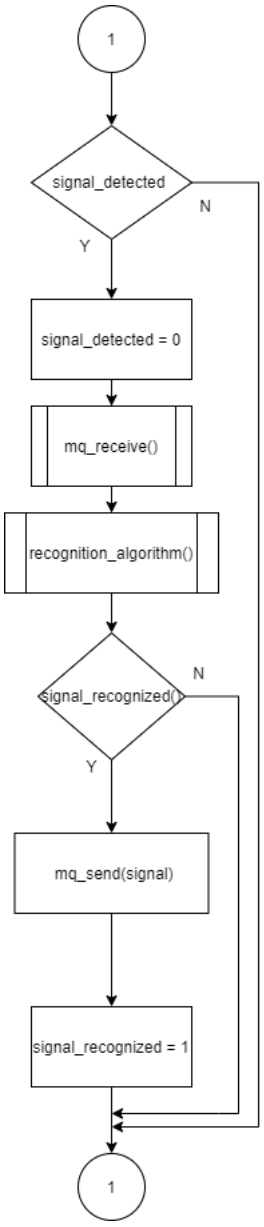


Figure 3.26: Recognize Transit Signals Task Flowchart

This task recognizes transit signals by first waiting until a signal is detected. Then, it clears the condition variable and prooceds to execute the recognition algorithm based on information transmited in the queue about the location of the detected feature. After that, it warns the process alert task by signaling that a signal was recognized.

tRecognize_obstacles

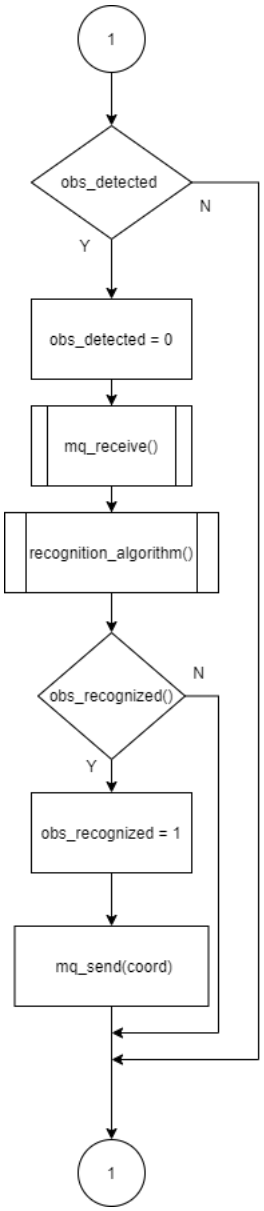


Figure 3.27: Recognize Obstacles Task Flowchart

This task recognizes transit signals by performing the same operations as the previous task, only this time it is for obstacle recognition.

tProcess_alert

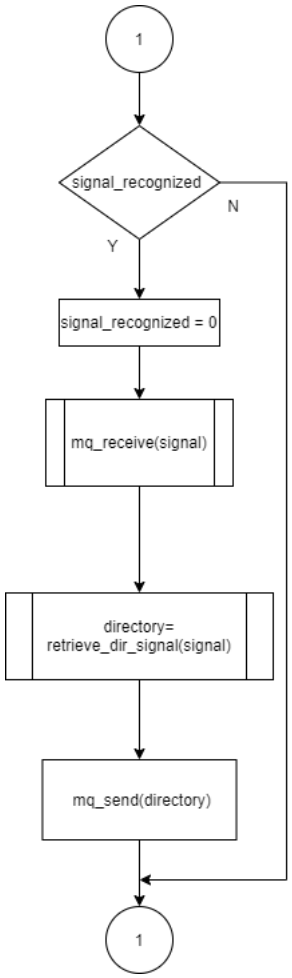


Figure 3.28: Process Alert Task Flowchart

This task only executes after a transit signal is recognized. If so, it clears the condition variable and based on the signal recognized sent via queue, it outputs a directory for the respective folder where it contains not only the signal logo but also its audio warning.

tMeasure_distance

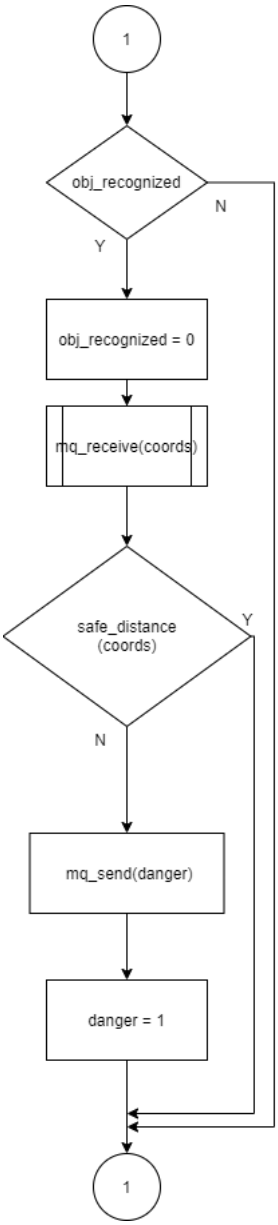


Figure 3.29: Measure Distance Task Flowchart

In case an object is recognized, this thread starts by clearing the respective condition variable and receives the objects' coordinates. If the coordinates of the object are outside of the safe distance, it sends a proximity warning to tGUI via a queue message.

tSendInfoToSTM

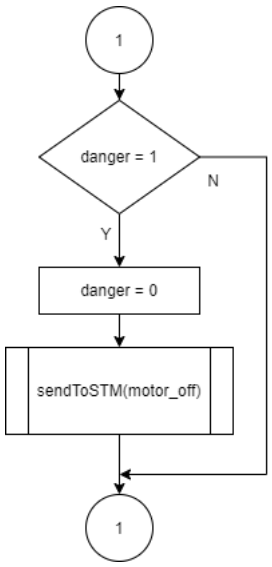


Figure 3.30: Send Info To STM Task Flowchart

This task is responsible for stoping the motor in case the distance is too short. It only sends via UART if danger condition variable is on, being cleared before the trasnmission.

tGUI

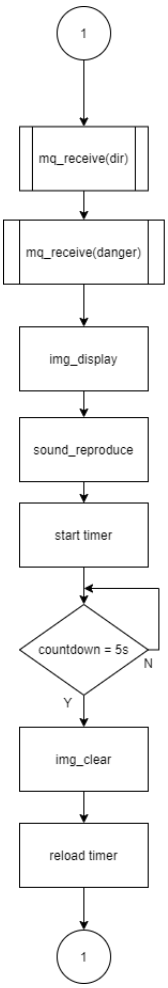


Figure 3.31: GUI Task Flowchart

The GUI task receives the directory of the respective signal and any warning message via different queues. With that, it displays and reproduces the associated image icon and sound warning, and, after that, starts a timer. When it finishes execution, it then clears the image and reloads the timer. It is also responsible for updating sensor values on the screen by fetching its data to the stm_queue.

3.7 Tasks Priority

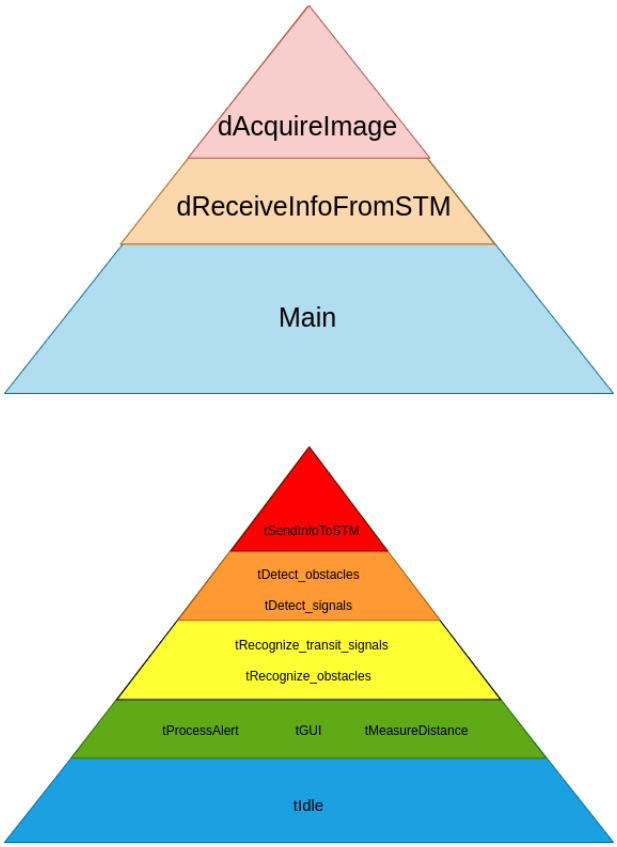


Figure 3.32: Process priority (up) and Task Priority on main process (down)

In the present picture, above there is the process priority and below is the task priority for the main process, where:

- **tSendInfoToSTM** - sending stop commands to block the ego car is of utmost importance that is why it has top priority;
- **tDetect_obstacles/tDetect_signals** - they have the second highest priority because all other tasks are waiting for their input;
- **tRecognize_transit_signals/tRecognize_obstacles** - have normal priority;
- **tProcessAlert/tGUI/tMeasureDistance** - have only slight priority than tIdle task;

3.8 Start-Up Process

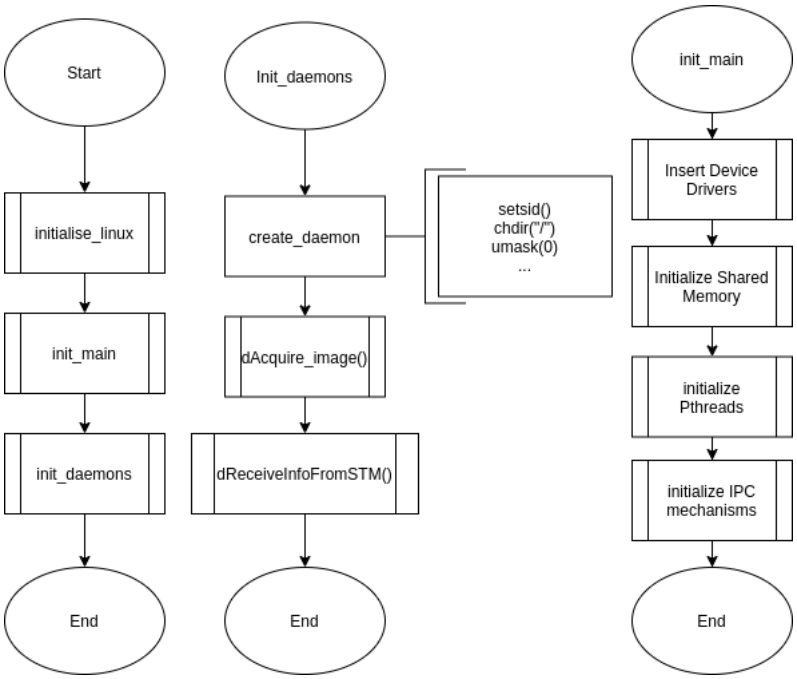


Figure 3.33: Boot Process

When the system boots, the following peripherals are initialized: I2C, UART, PWM, SPI, USB and TIMER. Camera, display and the speakers are initialized and turned on. The initialization order is performed in such a way that it does not jeopardize the system functionalities. After the peripherals initialization, the variables and tasks are initialized and when this step is finished, the scheduler is started and the system starts operating.

3.9 Shut-Down Process

As the system does not obtain any information from the environment, even if the system loses the main power, nothing critical is lost since the dataset of transit signs are pre-stored and any collision warning is just determined on-the-fly. However, as a constraint for this project the device driver that will be implemented will be responsible for shutting down the system.

The following chapter depicts the implementation stage where one initially configures the raspberry 4 with the following buildroot images.

Chapter 4

Implementation

4.1 Buidroot configuration

This section describes the Buildroot configuration for this project.

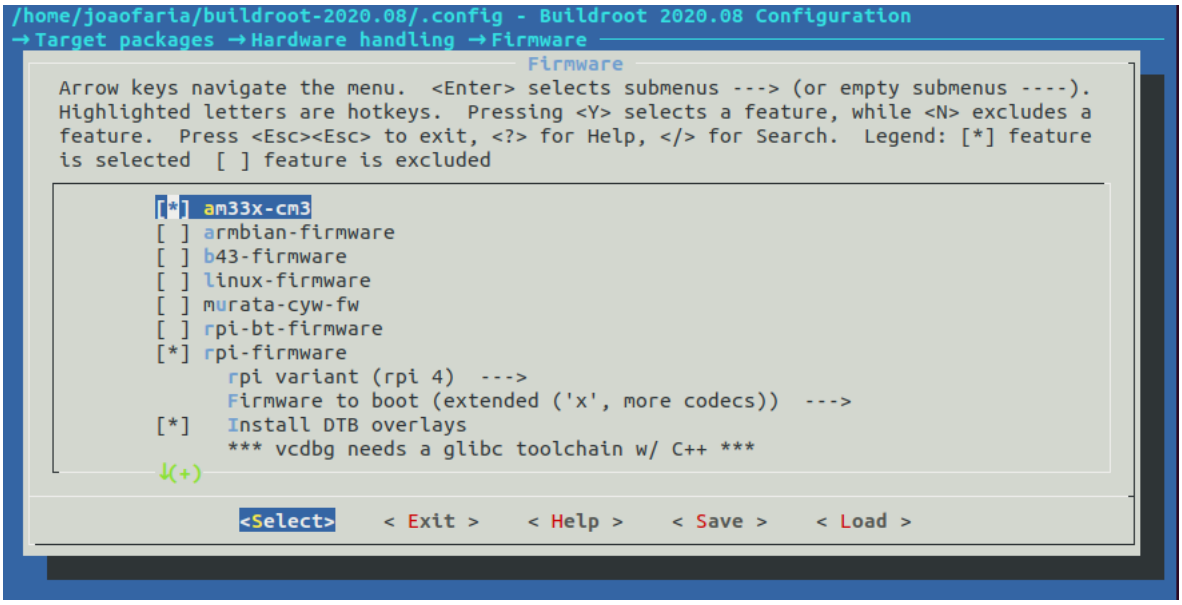


Figure 4.1: Rpi firmware

To configure the raspberry one needed to enable:

- **Camera Packages**
- **Interface Packages**
- **Sound Packages**
- **OpenCV Packages**
- **Python Packages**

4.1.1 Camera Packages

For the camera configs one should enable the gstreamer packages, lib4vl, and bcm2835.

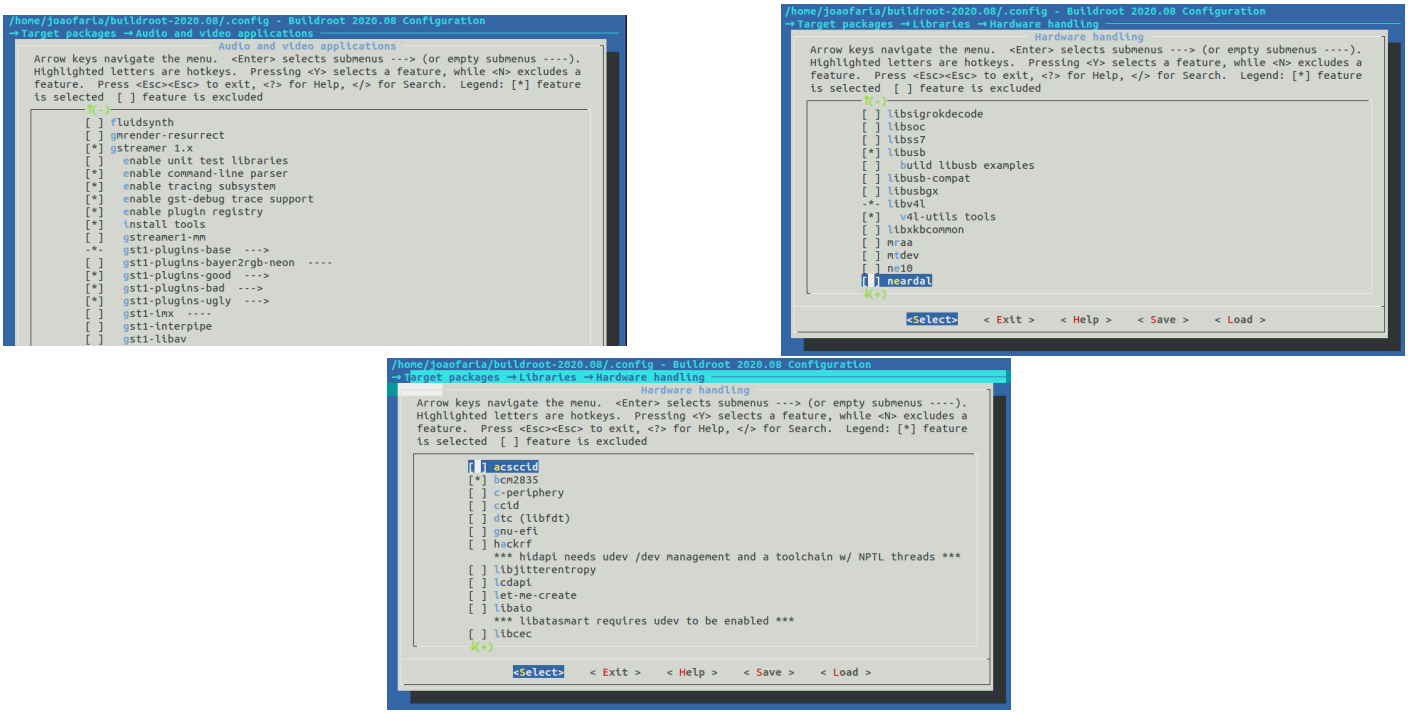


Figure 4.2: Camera packages

4.1.2 Interface Packages

As per the interface packages, the following qt packages need to be installed since one is working with a fusion of qt and python: pyqt.

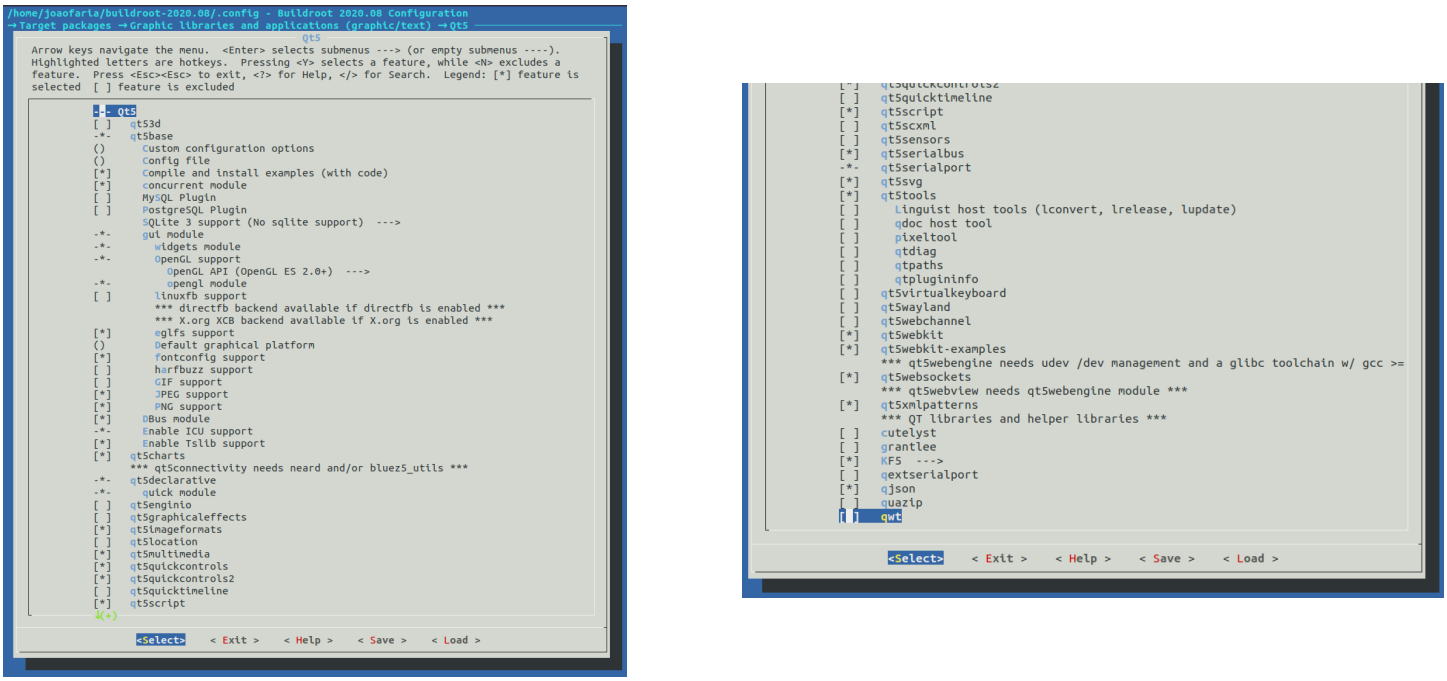


Figure 4.3: Interface packages

4.1.3 OpenCV Packages

The OpenCV packages are required to control the camera, and enable object detection and other functionalities, as defined below.

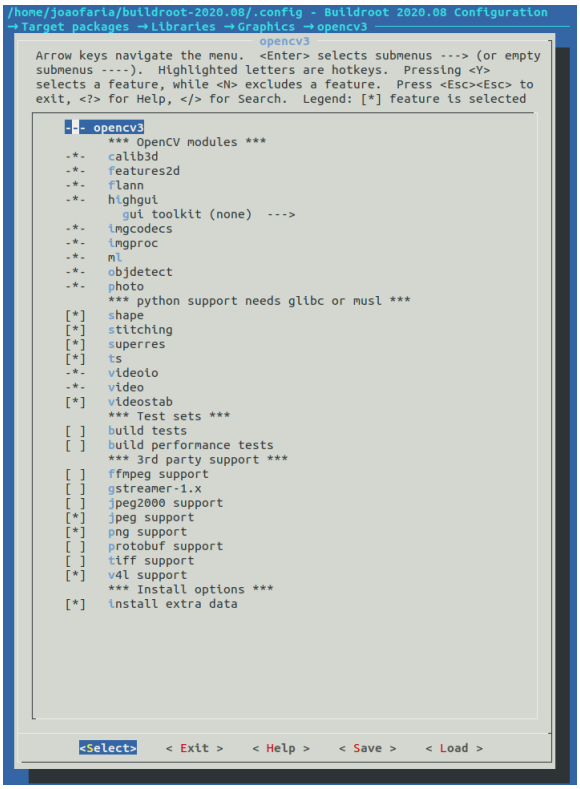


Figure 4.4: OpenCV packages

4.1.4 Sound Packages

For the sound, one enabled all the alsa configurations and the classic jack2 output.

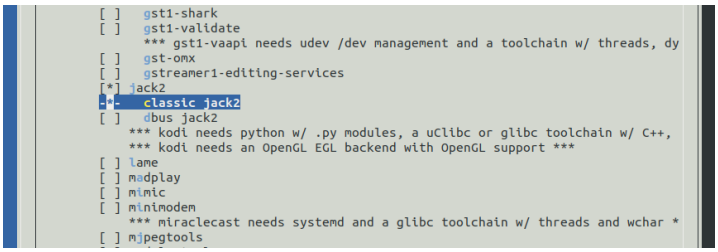


Figure 4.5: Sound packages

4.1.5 Python Packages

The python packages are needed since all the application code is in python and thus one enables pyqt and other important libraries defined below to make the project work.

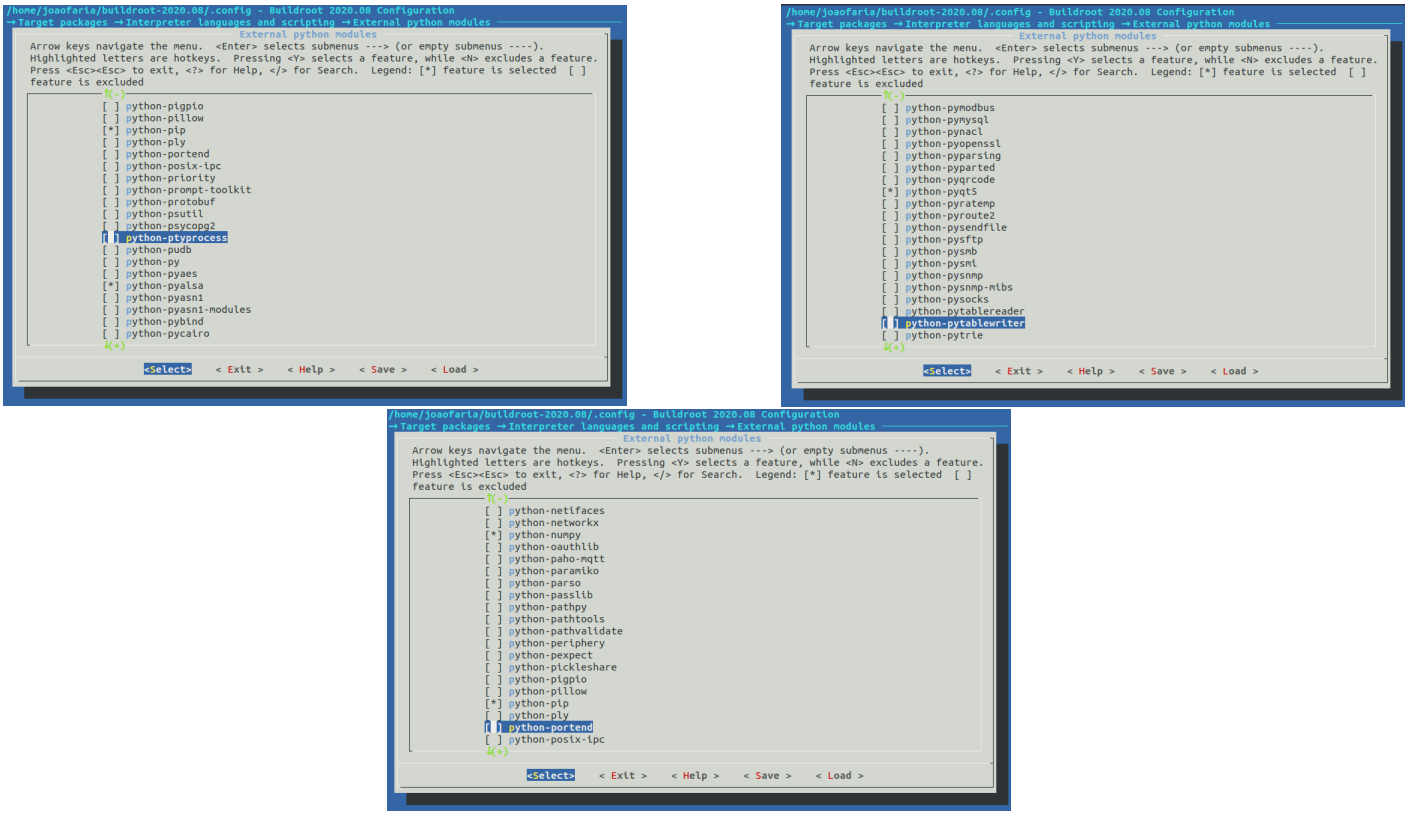


Figure 4.6: Python packages

4.1.6 Tensorflow Cross compiling

To run tensorflow on the raspberry to execute the application, one needed to install the following python wheel after installing all dependencies above and secure copy and install the python wheel below before secure copying the app which is the object file of app.py.

Listing 4.1: App and python wheel secure copy to rasp

```
scp tflite_runtime-2.5.0-cp38-cp38-linux_armv7l.whl root@10.42.0.209:/etc
//scp the python wheel

scp app.py root@10.42.0.209:/etc      //scp the application
```

Listing 4.2: Python wheel installation on raspberry

```
pip install tflite_runtime-2.5.0-cp38-cp38-linux_armv7l.whl
```

4.2 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. Notice the modbprobes for sound and video are from the bcm2835 library enabled on the prior section. Also, the main process which is the python application and the button driver and daemon.

```
# !/ bin / sh
# Sound
modprobe snd-bcm2835 &
# Video
modprobe snd-bcm2835-v4l2 &

# Device Drivers
insmod /root/buttons.ko &

# Daemon Process
./etc/mute_button &
# Main Process
./etc/App &
```

4.3 Machine Learning Neural Networks

4.3.1 Pre-training setup

One trained only the signs NN since the car detection model NN was already pre-trained and only needed to be integrated with the other submodules.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import cv2
from sklearn.model_selection import train_test_split
import pickle
import os
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
```

Listing 4.3: Imported Libraries

The above libraries are used to make sure every single part of the traning process is executable.

```
print("Num GPUs Available: ", len(tf.config.experimental.
    list_physical_devices('GPU')))
path = "myData"                     # folder with all the class folders
labelFile = 'labels.csv'            # file with all names of classes
batch_size_val=50                   # how many to process together
                                    # steps_per_epoch_val=2000
epochs_val=50                       # number of times the dataset
                                     # will be injected into the CNN
imageDimesions = (32,32,3)
testRatio = 0.2                     # if 1000 images split will 200
                                    # for testing
validationRatio = 0.2               # if 1000 images 20% of remaining 800
                                    # will be 160 for validation
```

Listing 4.4: Training Parameters

The above parameters define where to fetch the images, the labels, the number of time the dataset is fed back and forth into the CNN and data split between training, testing and validation phases.

```
count = 0
images = []
classNo = []
myList = os.listdir(path)                #Path of images
noOfClasses=len(myList)                  #Total classes detected
for x in range (0,len(myList)):          #Importing classes
    myPicList = os.listdir(path+"/"+str(count))
    for y in myPicList:
        curImg = cv2.imread(path+"/"+str(count)+"/"+y)
        images.append(curImg)
        classNo.append(count)
    print(count, end = " ")
    count +=1
```

```
print(" ")
images = np.array(images)                #Image data storage
classNo = np.array(classNo)              #Class data storage
```

Listing 4.5: Image Importing

The above code snippet is used to iterate through all images and append them along with their corresponding labels to the image and class arrays.

```
Split Data
X_train, X_test, y_train, y_test = train_test_split(images, classNo,
    test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train,
    y_train, test_size=validationRatio)
# X_train = ARRAY OF IMAGES TO TRAIN
# y_train = CORRESPONDING CLASS ID
```

Listing 4.6: Data Split

There is an array for storing images of the foremetioned training, validation and testing phases.

```
print("Data Shapes")
print("Train",end = "");print(X_train.shape,y_train.shape)
print("Validation",end = "");print(X_validation.shape,y_validation.shape)
print("Test",end = "");print(X_test.shape,y_test.shape)
assert(X_train.shape[0]==y_train.shape[0]), "The number of images in not
    equal to the number of lables in training set"
assert(X_validation.shape[0]==y_validation.shape[0]), "The number of
    images in not equal to the number of lables in validation set"
assert(X_test.shape[0]==y_test.shape[0]), "The number of images in not
    equal to the number of lables in test set"
assert(X_train.shape[1:]==(imageDimesions)),"The dimesions of the
    Training images are wrong "
assert(X_validation.shape[1:]==(imageDimesions)),"The dimesionas of the
    Validation images are wrong "
assert(X_test.shape[1:]==(imageDimesions)),"The dimesions of the Test
    images are wrong"
```

Listing 4.7: Image Label Assertion

To properly train the NN, the number of images should match the number of labels.

```
data=pd.read_csv(labelFile)
print("data shape ",data.shape,type(data))
```

Listing 4.8: Label Reading

Labels are stored in a csv file format

4.3.2 Image pre-processing

Before the training phase, one should pre-process dataset images. Firstly they were fetched with the following dimensions: **32x32x3**. The **32x32** are number of pixels in width and height and the **3** is the number of color channels of the RGB color system.

```
def grayscale(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    return img
def equalize(img):
    img =cv2.equalizeHist(img)
    return img
def preprocessing(img):
    img = grayscale(img)      # CONVERT TO GRAYSCALE
    img = equalize(img)       # STANDARDIZE THE LIGHTING IN AN IMAGE
    img = img-128/128         # TO NORMALIZE VALUES BETWEEN -1 AND 1
                              # INSTEAD OF 0 TO 255
    return img

X_train=np.array(list(map(preprocessing,X_train)))
X_validation=np.array(list(map(preprocessing,X_validation)))
X_test=np.array(list(map(preprocessing,X_test)))
```

Listing 4.9: Image Pre-processing

After that step, those same images are converted to **black and white** and its contrast is calibrated through an equalization histogram. Finally are normalized, that is, its pixels share a value between -1 and 1 instead of between 0 and 255.

```
dataGen= ImageDataGenerator(width_shift_range=0.1,   # 0.1 = 10%     IF
    MORE THAN 1 E.G 10 THEN IT REFFERS TO NO. OF  PIXELS EG 10 PIXELS
                            height_shift_range=0.1,
                            zoom_range=0.2,          # 0.2 MEANS CAN GO
    FROM 0.8 TO 1.2
                            shear_range=0.1,         # MAGNITUDE OF SHEAR
     ANGLE
                            rotation_range=10)       # DEGREES
dataGen.fit(X_train)
batches= dataGen.flow(X_train,y_train,batch_size=20) # REQUESTING DATA
    GENERATOR                                                 # TO
    GENERATE IMAGES  BATCH
      # SIZE = NO. OF IMAGES
             # CREATED EACH TIME ITS
                    # CALLED
X_batch,y_batch = next(batches)
                                                      # TO SHOW AUGMENTED
    IMAGE                  SAMPLES
fig,axs=plt.subplots(1,15,figsize=(20,5))
fig.tight_layout()
for i in range(15):
    axs[i].imshow(X_batch[i].reshape(imageDimesions[0],imageDimesions[1])
    )
    axs[i].axis('off')
plt.show()
y_train = to_categorical(y_train,noOfClasses)
y_validation = to_categorical(y_validation,noOfClasses)
y_test = to_categorical(y_test,noOfClasses)
```

Listing 4.10: Image Augmentation

Image augmentation is a technique used to improve the model predictions where some dataset images are picked and applied a rotationary transformation or blurry.

4.3.3 Overfitting handling

```
def myModel():
    no_Of_Filters=60
    size_of_Filter=(5,5) # THIS IS THE KERNEL THAT MOVE AROUND THE IMAGE TO GET THE FEATURES.
                         # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN USING 32 32 IMAGE
    size_of_Filter2=(3,3)
    size_of_pool=(2,2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MORE, TO REDUCE OVERFITTING
    no_Of_Nodes = 500   # NO. OF NODES IN HIDDEN LAYERS
    model= Sequential()
    model.add((Conv2D(no_Of_Filters,size_of_Filter,input_shape=(imageDimesions[0],imageDimesions[1],1),activation='relu')))  # ADDING MORE CONVOLUTION LAYERS = LESS FEATURES BUT CAN CAUSE ACCURACY TO INCREASE
    model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu')))
    model.add(Dropout(0.5))
    model.add(MaxPooling2D(pool_size=size_of_pool)) # DOES NOT EFFECT THE DEPTH/NO OF FILTERS
    model.add((Conv2D(no_Of_Filters//2, size_of_Filter2,activation='relu')))
    model.add(Dropout(0.5))
    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(no_Of_Nodes,activation='relu'))
    model.add(Dropout(0.5)) # INPUTS NODES TO DROP WITH EACH UPDATE 1 ALL 0 NONE
    model.add(Dense(noOfClasses,activation='softmax')) # OUTPUT LAYER
    # COMPILE MODEL
    model.compile(Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
    return model
```

Listing 4.11: Model Composition

To handle **overfit**, a condition where the model has an extremely high accuracy detecting the correct images in training but fails to generalize predictions on new images, some techniques were used such as:

- **increase the dataset images** of the categories to detect;
- add **dropout layers**, that is, to turn on or off randomly some neurons;
- **reduce model complexity**, removing layers of neurons;

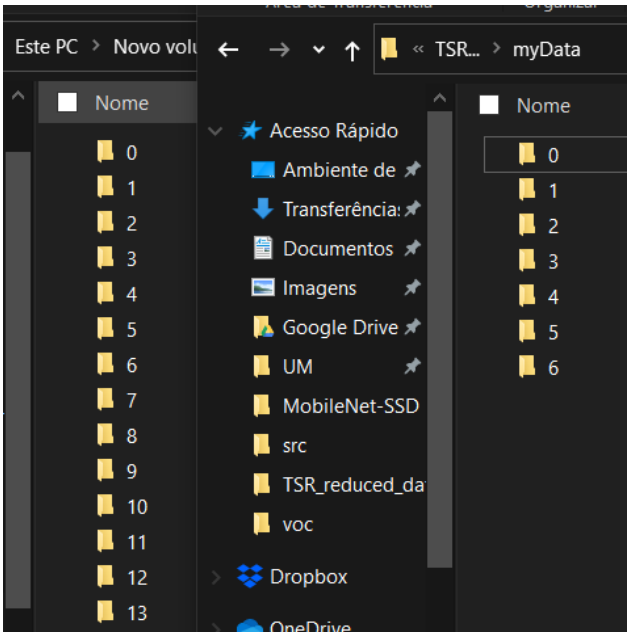
4.3.4 Training

To detect traffic signs, one **trained** a model in the laptop with the intended **GTSRB** with a dataset comprised of 43 classes of signs and then applied **transfer learning** by reducing the dataset to only 7 classes of signs to improve framerate (and to prove the concept) 4.7a.

```
model = myModel()
print(model.summary())
history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=
    batch_size_val),epochs=epochs_val,validation_data=(X_validation,
    y_validation),shuffle=1)
```

Listing 4.12: Train function

The pictures bellow depict how the model is composed 4.7b, the image per class distribution 4.7c of the traffic signs, as well as some output examples of image **pre-processing** techniques were performed to improve training, 4.7d. Moreover, in 4.7e and 4.7f is represented the progress over the training and validation phases and also a prove that the system correctly predicts the sign portrayed.

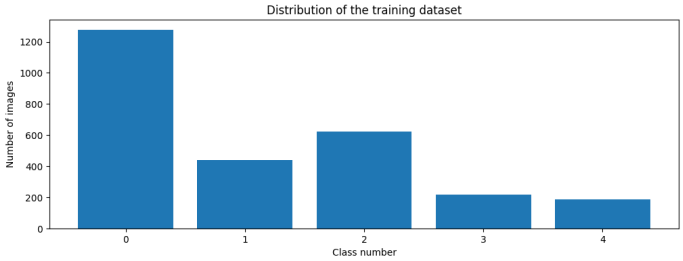


(a) Transfer Learning: Dataset reduction

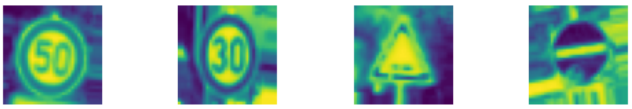
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 60)	1560
conv2d_1 (Conv2D)	(None, 24, 24, 60)	90060
max_pooling2d (MaxPooling2D)	(None, 12, 12, 60)	0
conv2d_2 (Conv2D)	(None, 10, 10, 30)	16230
conv2d_3 (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d_1 (MaxPooling2	(None, 4, 4, 30)	0
dropout (Dropout)	(None, 4, 4, 30)	0
flatten (Flatten)	(None, 480)	0
dense (Dense)	(None, 500)	240500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 7)	3507

Total params: 359,987
Trainable params: 359,987
Non-trainable params: 0

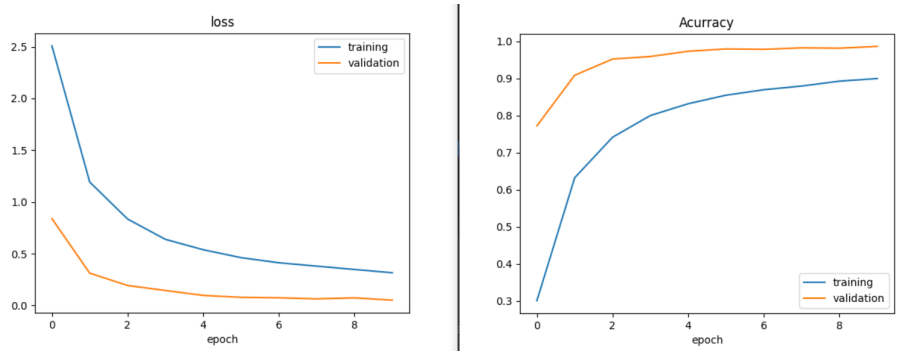
(b) Model Architecture



(c) Images per class distribution



(d) Image augmentation



(e) Training and Validation Graphs: Loss and Accuracy

(f) Stop sign classification

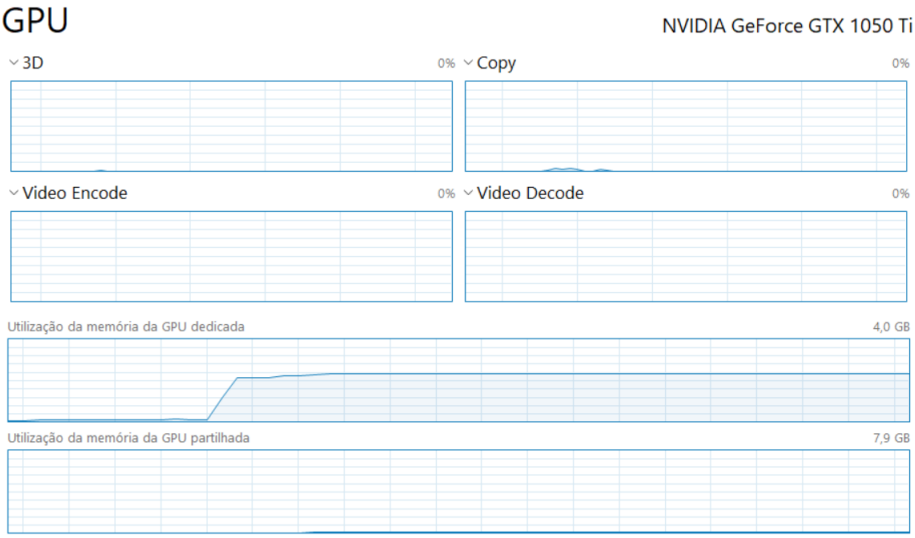


Figure 4.8: GPU Usage

The above picture 4.8 shows the GPU usage when the algorithm is training. Training with GPU can be 75x faster than with the CPU.

4.4 Camera

To use camera one needs to have on the /dev folder the video0 driver and perform the below commands for image or video capture.

```
raspistill -o image.jpg //capture image

raspisvid -o image.jpg  //capture video
```

Listing 4.13: Camera play commands

4.5 Sound

Initially the sound was reproduced but could not be heard. So one had to perform the following configurations.

4.5.1 Raspberry Setup Configurations

```
pcm.!default {
        type hw
        card 1
}

ctl.!default {
```

```
        type hw
        card 1
```

Listing 4.14: Sound Output Jack Enable

The project uses the Google API text to speech to turn the product more user-friendly.

4.5.2 Setup Google API text to speech

```
#!/usr/bin/env python
# Script Written by - Mikhail Kulin 2020 www.kulin.co
# Copyright 2018 Google Inc. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
    .
# See the License for the specific language governing permissions and
# limitations under the License.

"""Google Cloud Text-To-Speech API sample application .
Example usage:
python quickstart.py
"""

import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="/home/joaofaria/Desktop/
    ttsjsonkey/tts test-d9303a475d04.json"
from google.cloud import texttospeech

# Instantiates a client

client = texttospeech.TextToSpeechClient()

# Set the text input to be synthesized
synthesis_input = texttospeech.SynthesisInput(text="Yield sign")

# Build the voice request, select the language code ("en-US") and the
    ssml
# voice gender ("female")
voice = texttospeech.VoiceSelectionParams(
language_code='en-US',
name='en-US-Wavenet-C',
ssml_gender=texttospeech.SsmlVoiceGender.FEMALE)

# Select the type of audio file you want returned
audio_config = texttospeech.AudioConfig(
audio_encoding=texttospeech.AudioEncoding.MP3)

```

```
# Perform the text-to-speech request on the text input with the selected
# voice parameters and audio file type
response = client.synthesize_speech(
input=synthesis_input, voice=voice, audio_config=audio_config
)

# The response's audio_content is binary.
with open('output.mp3', 'wb') as out:
# Write the response to the output file.
  out.write(response.audio_content)
print('Audio content written to file "output.mp3"')
```

Listing 4.15: Text To Speech Script

Before reproducing the sound, one had to convert its codec to wav form and before that, to convert from mp3 to wav format, otherwise white noise will be reproduced.

4.5.3 FFMPEG Conversion

```
ffmpeg -i output.mp3 output.wav
```

Listing 4.16: Mp3 to wav conversion

```
ffmpeg -i output.wav -c:a wav output.wav
```

Listing 4.17: Codec conversion to wav

This is the command used to play the sound.

```
aplay -D output.wav
```

Listing 4.18: Sound play command

4.6 Touchscreen

For the 3.5 inch touchscreen one installed its driver with the below commands.

```
sudo rm -rf LCD-show
git clone https://github.com/goodtft/LCD-show.git
chmod -R 755 LCD-show
cd LCD-show/
sudo ./LCD35-show
```

Listing 4.19: 3.5 Rpi Touchscreen driver installation command

4.6.1 Deployment

In order to deploy the same libraries were installed on the raspberry as well as python 3.6 and tensorflow 2.3 version. To run on the raspberry both models the following steps were followed:

- load model
- setup picamera resolutions

- define threshold, so that only high percentage predictions are not discarded
- define the classes that the model is able to predict
- read images on a loop in real-time

4.7 Application

The application was totally build in python, using pyqt.

4.7.1 Qthreads as an wrapper for pthreads

Qthreads are pthreads wrappers. So one used them to work in the pyqt environment.

4.7.2 Image Capture Thread

There is an thread for capturing images.

```
class capture_img_thread(QThread):
    #capture image
    done_signal = pyqtSignal(str)
    def __init__(self):
        #self.Cam = PiCamera()
        #self.Cam.resolution =(640,480)
        self.img0 = cv.VideoCapture(0)
        self.img0.set(3, frameWidth)
        self.img0.set(4, frameHeight)
        self.img0.set(10, brightness)
        QThread.__init__(self)
    def run(self):
        print("Entrada: Tirar foto")
        success, cap2 = self.img0.read()
        q_signs.put(cap2)
        q_cars.put(cap2)
        self.done_signal.emit('Foto')
        print("Saida: Tirar foto")
```

Listing 4.20: Capture Image Thread

4.7.3 Receive From STM Thread

This is the thread responsible for receiving commands from STM and showing these values to the user.

```
class receive_from_stm_thread(QThread):
    #read imputs from stm
    def __init__(self):
        QThread.__init__(self)
    def run(self):
        global ref
        global setpoint
```

```
        global actual_lux
        global actual_hum
        global actual_temp
        while True:
            while usb_serial.inWaiting()>0:
                readed=usb_serial.read(usb_serial.inWaiting())
                if b'RT' in readed:
                    if readed[2] >47 and readed[2] <58 and readed[3] >47
    and readed[3] <58:
                        setpoint = chr(readed[2])+chr(readed[3])
                        win.setpoint_val.setNum(int(setpoint))
                elif b'T' in readed:
                    if readed[1] >47 and readed[1] <58 and readed[2] >47
    and readed[2] <58:
                        actual_temp = chr(readed[2])+chr(readed[3])
                        win.actual_val.setNum(int(actual_temp))
                if b'RL' in readed:
                    if len(readed)>5:
                        ref = 100
                    elif len(readed)==5:
                        ref = chr(readed[2])+chr(readed[3])
                    elif len(readed)<5:
                        ref = chr(readed[2])
                    win.ref_light_slider.setValue(int(ref))
                elif b'ML' in readed:
                    mode = chr(readed[2])
                    if int(mode) ==1:
                        win.auto_btn.setStyleSheet("border:2px solid rgb
    (25,25,25)")
                    else:
                        win.auto_btn.setStyleSheet("border:1px solid rgb
    (200,200,200)")
                elif b'L' in readed:
                    if len(readed)>4:
                        actual_lux = 100
                    elif len(readed)==4:
                        actual_lux = chr(readed[1])+chr(readed[2])
                    elif len(readed)<4:
                        actual_lux = chr(readed[2])
                elif b'H' in readed:
                    if len(readed)>4:
                        actual_hum = 100
                    elif len(readed)==4:
                        actual_hum = chr(readed[1])+chr(readed[2])
                    elif len(readed)<4:
                        actual_hum = chr(readed[1])
                        win.humidity_val.setNum(int(actual_hum))
            QThread.msleep(50)
```

Listing 4.21: Receive From STM Thread

4.7.4 Detect Cars Thread

This is the thread that checks the existence of a vehicle in the proximity of the car through the captured image.

```
class detect_cars_thread(QThread):
    #Obstacle Detection
    done_signal = pyqtSignal(str)
    def __init__(self):
        self.CLASSES = ["background", "aeroplane", "bicycle", "bird", "
    boat",
            "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
            "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
            "sofa", "train", "tvmonitor"]
        #self.timer.singleShot(5000, lambda: win.CollisionWarningLabel.
    setPixmap(self.pix))
        QThread.__init__(self)
    def run(self):
        print("Entrada: Verificar se existe carro")
        #self.timer = QTimer()
        img = q_cars.get()
        rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        (H, W) = img.shape[:2]
        # convert the frame to a blob and pass the blob through the
        # network and obtain the detections
        blob = cv.dnn.blobFromImage(img, size=(300, 300), ddepth=cv.CV_8U
    )
        net.setInput(blob, scalefactor=1.0 / 127.5, mean=[127.5, 127.5,
    127.5])
        detections = net.forward()
        # loop over the detections
        for i in np.arange(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated
        # with the prediction
            confidence = detections[0, 0, i, 2]
        # filter out weak detections by ensuring the ‘confidence‘
        # is greater than the minimum confidence
            if confidence > 0.7:
                idx = int(detections[0, 0, i, 1])
                if self.CLASSES[idx] != "car":
                    #print (self.CLASSES[idx])
                    self.done_signal.emit('NOP')
                    break
                box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
                (startX, startY, endX, endY) = box.astype("int")
                    #find screen dimensions
                if ((endX - startX)>0.3*W)and((0.3*W)<((endX+startX)/2)
    <(0.7*W)):
                    print("COLLISION WARNING")
                    pix = QPixmap('../app_python/signs/
    collision_avoidance.png')
                    win.CollisionWarningLabel.setPixmap(pix)
                    win.collision_label.setText('Collision Warning')
                    os.system(f'aplay ../app_python/signs/sound.wav')
```

```
                    self.done_signal.emit('Obstacle')
        else:
            self.done_signal.emit('NOP')
        print("Saida: Verificar se existe carros")
```

Listing 4.22: Detect Cars Thread

4.7.5 Detect Signals Thread

This is the thread that checks for the existence of a signal and its identification.

```
class detect_signals_thread(QThread):
    #Signal Detection
    done_signal = pyqtSignal(str)
    def __init__(self):
        self.Threshold = 0.95
        QThread.__init__(self)
    def run(self):
        print("Entrada: Verificar se existe sinais")
        # READ IMAGE
        global model
        img0 = q_signs.get()
        print("Dps retirar queue")
        img = np.asarray(img0)
        img = cv.resize(img, (32, 32))
        img = preprocessing(img)
        img = img.reshape(1, 32, 32, 1)
        # PREDICT IMAGE
        predictions = model.predict(img)
        classIndex = model.predict_classes(img)
        probabilityValue = np.amax(predictions)
        print("Dps inferencia")
        if probabilityValue > self.Threshold:
            print("predicted")
            print(probabilityValue)
            signs_path = '../app_python/signs/' + str(classIndex)[1] + '/
    img.png'
            audio_path = f'aplay ../app_python/signs/' + str(classIndex)
    [1] + '/sound.wav'
            win.Sign_label.setText(label_array[int(str(classIndex)[1])])
            pix = QPixmap(signs_path)
            win.Sign_show.setPixmap(pix)
            os.system(audio_path)
            self.done_signal.emit('Sign')
        else:
            self.done_signal.emit('NOP')
        print("Saida: Verificar se existe sinais")
```

Listing 4.23: Detect Signals Thread

4.7.6 Main and UI Thread

This is the class of the graphical interface and its functions.

```
class Ui(QtWidgets.QMainWindow):
    def __init__(self):
        super(Ui,self).__init__()
        uic.loadUi('mainwindow.ui',self)
        self.more_btn.clicked.connect(self.btn_more_pressed)
        self.less_btn.clicked.connect(self.btn_less_pressed)
        self.auto_btn.clicked.connect(self.btn_auto_pressed)
        self.ref_light_slider.valueChanged.connect(self.
    slider_ref_changed)
        self.setWindowFlag(QtCore.Qt.FramelessWindowHint)
        self.show()
    def btn_more_pressed(self):
        global setpoint
        setpoint +=1
        if setpoint >= 27:
            setpoint =27
        self.setpoint_val.setNum(setpoint)
        usb_serial.write(str.encode('T'+str(setpoint)+'\r\n'))
    def btn_less_pressed(self):
        global setpoint
        setpoint -=1
        if setpoint <= 15:
            setpoint =15
        self.setpoint_val.setNum(setpoint)
        usb_serial.write(str.encode('T'+str(setpoint)+'\r\n'))
    def btn_auto_pressed(self):
        global mode
        if mode ==1:
            mode=0
            self.auto_btn.setStyleSheet("border:1px solid rgb
    (200,200,200)")
        else:
            mode=1
            self.auto_btn.setStyleSheet("border:2px solid rgb(25,25,25)")
        usb_serial.write(str.encode('LM'+str(mode)+'\r\n'))
    def slider_ref_changed(self,value):
        global ref
        ref=value
        usb_serial.write(str.encode('L'+str(ref)+'\r\n'))
```

Listing 4.24: User Interface Class

This is the main task where you can see the task startup sequence.

```
if __name__ == '__main__':

    #Application Start
    app = QApplication(sys.argv)
    win = Ui()

    #Queues Initialization
    q_signs = queue.Queue()
    q_cars = queue.Queue()

    #Camera Capture Task
```

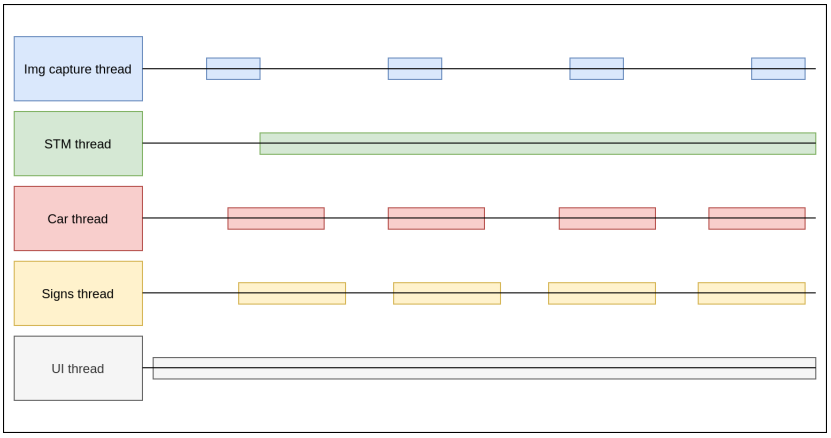


Figure 4.9: Thread Timeline

```
    img_capture = capture_img_thread()
    img_capture.done_signal.connect(process_done_signal)

    #Receive from STM Task
    receive_from_stm = receive_from_stm_thread()

    #Detect vehicles Task
    detect_vehicles = detect_cars_thread()
    detect_vehicles.done_signal.connect(process_done_signal)

    #Detect Transit Signals Task
    detect_signals = detect_signals_thread()
    detect_signals.done_signal.connect(process_done_signal)

    #Tasks initialization
    img_capture.start()
    receive_from_stm_thread.start()

    while True:
        app.exec_()

    cv.destroyAllWindows()
    sys.exit()
```

Listing 4.25: Main Thread

4.8 Device Driver

For the device driver, one used a button for **muting/unmuting**.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
```

```
5  #include <linux/cdev.h>
6  #include <linux/device.h>
7  #include <linux/slab.h>
8  #include <linux/uaccess.h>
9  #include <linux/ioctl.h>
10 #include <linux/sched/signal.h>
11 #include <linux/proc_fs.h>
12 #include <linux/fcntl.h>
13 #include <linux/types.h>
14
15 #include <linux/gpio.h>
16 #include <linux/interrupt.h>
17
18 #include "utils.h"
19
20 #define DEVICE_NAME "button"
21 #define CLASS_NAME "buttonClass"
22 #define REG_CURRENT_TASK _IOW('a','a',int32_t*)
23 #define SIGBUTTON 44
24 #define IOCTL_PID 1
25 #define GPIO_INPUT 0
26
27 MODULE_LICENSE("GPL");
28
29 static struct kernel_siginfo info;
30 static pid_t pid;
31 static struct task_struct *task = NULL;
32
33 static dev_t dev;
34 static struct class *dev_class = NULL;
35 static struct cdev c_dev;  // Character device structure
36
37 struct GpioRegisters *s_pGpioRegisters;
38
39 static unsigned int GpioPin = 20;
40 static unsigned int irqNumber;
41 static int __init button_driver_init(void);
42 static void __exit button_driver_exit(void);
43 static int button_open(struct inode *inode, struct file *file);
44 static int button_close(struct inode *inode, struct file *file);
45 static ssize_t button_read(struct file *filp, char __user *buf, size_t
      len,loff_t * off);
46 static ssize_t button_write(struct file *filp, const char *buf, size_t
      len, loff_t * off);
47 static long button_ioctl(struct file *file, unsigned int cmd, unsigned
      long arg);
48 static int toggle = 1;
49 static int toggle_ant = 0;
50 static int aux = 0;
51 static irqreturn_t irq_handler(int irq,void *dev_id)
52 {
53   int value = gpio_get_value(GpioPin);
54
```

```
  if(value)
{
    aux = toggle;
    toggle = toggle_ant;
    toggle_ant = aux;
}
      info.si_signo = SIGBUTTON;
      info.si_code = SI_QUEUE;
      info.si_int = toggle;
  printk(KERN_INFO "PIN -> %d\n",value);
  task = pid_task(find_pid_ns(pid, &init_pid_ns), PIDTYPE_PID);


      if (task != NULL)
      {
          if(send_sig_info(SIGBUTTON, &info, task) < 0)
          {
              printk(KERN_INFO "Unable to send signal\n");
          }
          else
          {
            printk(KERN_INFO "Signal sent\n");
          }
      }
  return IRQ_HANDLED;
}

static int button_open(struct inode* inode, struct file *file)
{
  printk(KERN_INFO "Device File Opened\n");
  return 0;
}

static int button_close(struct inode *inode, struct file * file)
{
  printk(KERN_INFO "Device File Closed\n");
  return 0;
}

static ssize_t button_write(struct file *filp, const char __user *buf,
    size_t len, loff_t *off)
{
  printk(KERN_INFO "Write function\n");
  return 0;
}

static ssize_t button_read(struct file *filp, char __user *buf, size_t
    len, loff_t *off)
{
  printk(KERN_INFO "Read function\n");
  return 0;
}

```

```
106 static long button_ioctl(struct file *file, unsigned int cmd, unsigned
    long arg)
107 {
108   if(cmd == IOCTL_PID)
109   {
110   if (copy_from_user(&pid, (pid_t*)arg, sizeof(pid_t)))
111   {
112     printk(KERN_INFO "copy_from_user failed\n");
113     return -1;
114   }
115   printk(KERN_INFO "PID->%d\n", pid);
116   }
117   else
118   {
119     printk(KERN_INFO "ioctl failed\n");
120   }
121   return 0;
122 }
123
124
125 static struct file_operations fops =
126 {
127   .owner = THIS_MODULE,
128   .write = button_write,
129   .read = button_read,
130   .release = button_close,
131   .open = button_open,
132   .unlocked_ioctl = button_ioctl,
133 };
134
135
136 static int __init button_driver_init(void)
137 {
138   if ((alloc_chrdev_region(&dev, 0, 1, DEVICE_NAME)) < 0)
139   {
140     printk(KERN_INFO "Cannot allocate major number\n");
141           return -1;
142   }
143
144   /*Creating cdev structure*/
145   cdev_init(&c_dev, &fops);
146
147       /*Adding character device to the system*/
148       if((cdev_add(&c_dev,dev,1)) < 0)
149       {
150           printk(KERN_INFO "Cannot add the device to the system\n");
151     unregister_chrdev_region(dev,1);
152   }
153
154   /*Creating struct class*/
155     if((dev_class = class_create(THIS_MODULE, CLASS_NAME)) == NULL)
156     {
157           printk(KERN_INFO "Cannot create the struct class\n");
```

```
158             unregister_chrdev_region(dev,1);
159     }
160
161       /*Creating device*/
162         if((device_create(dev_class, NULL, dev, NULL, DEVICE_NAME)) == NULL
        )
163         {
164             printk(KERN_INFO "Cannot create the Device\n");
165             class_destroy(dev_class);
166             unregister_chrdev_region(dev,1);
167         }
168
169         irqNumber = gpio_to_irq(GpioPin);
170
171         //IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING
172
173         if (request_irq(irqNumber, irq_handler, IRQF_TRIGGER_RISING,
        DEVICE_NAME, (void *)(irq_handler)))
174         {
175             printk(KERN_INFO "Cannot register IRQ\n");
176             free_irq(irqNumber,(void *)(irq_handler));
177             class_destroy(dev_class);
178             unregister_chrdev_region(dev,1);
179           }
180
181     s_pGpioRegisters = (struct GpioRegisters *)ioremap_nocache(GPIO_BASE,
        sizeof(struct GpioRegisters));
182     SetGPIOFunction(s_pGpioRegisters, GpioPin, GPIO_INPUT);
183     return 0;
184 }
185
186 static void __exit button_driver_exit(void)
187 {
188     SetGPIOFunction(s_pGpioRegisters, GpioPin, GPIO_INPUT);
189     iounmap(s_pGpioRegisters);
190     free_irq(irqNumber,(void *)(irq_handler));
191     device_destroy(dev_class, dev);
192     class_destroy(dev_class);
193     cdev_del(&c_dev);
194     unregister_chrdev_region(dev, 1);
195     printk(KERN_INFO "Device Driver Remove\n");
196 }
197
198 module_init(button_driver_init);
199 module_exit(button_driver_exit);
```

Listing 4.26: Button driver

As one can see in the code above, besides the init() and exit() functions, only the ioctl() is implemented to associate button signal SIGBUTTON (44) emitted when it is pressed with the daemon identified by its pid, below explained in detail. When the button is pressed, the irqhandler() registered in the ioctl() receives the button state (PIN 20 set has output GPIO) and it emits a signal with an integer value 0 if previously it was

sent an 1 with the pressing of the button, and vice-versa.

4.9 Button Daemon

The daemon button is created, running in background so that the user can mute/unmute sound system whilst interacting with the application, the main process. When the button is pressed, the daemon in the user space catches the signal and toggles the mute functionality.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>


#define REG_CURRENT_TASK _IOW('a','a',int32_t*)
#define SIGBUTTON 44
#define IOCTL_PID 1

void sig_handler(int n, siginfo_t *info, void *unused, int toggle);

int main(int argc, char *argv)
{
  pid_t pid, sid;
  int len, fd;
  const time_t timebuf;

  pid = fork(); // create a new process

  if (pid < 0) { // on error exit
    perror("fork");
  exit(EXIT_FAILURE);
  }

  if (pid > 0){
    printf("Deamon PID: %d\n", pid);
    exit(EXIT_SUCCESS); // parent process (exit)
  }
  sid = setsid(); // create a new session

  if (sid < 0) { // on error exit
    perror("setsid");
    exit(EXIT_FAILURE);
```

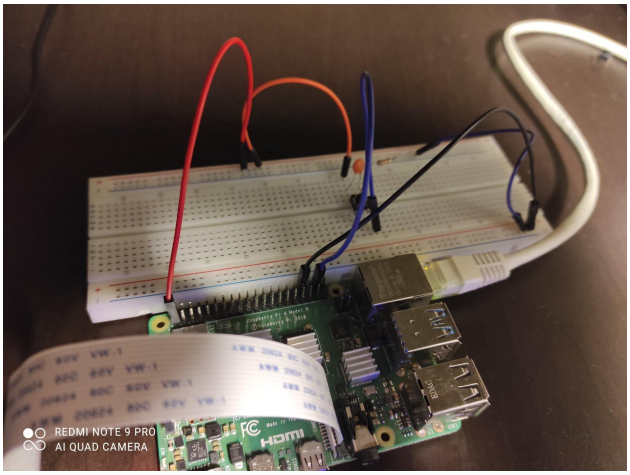
```
  }
  // make '/' the root directory
  if (chdir("/") < 0) { // on error exit
    perror("chdir");
    exit(EXIT_FAILURE);
  }
  umask(0);
  close(STDIN_FILENO);  // close standard input file descriptor
  close(STDOUT_FILENO); // close standard output file descriptor
  close(STDERR_FILENO); // close standard error file descriptor

  static int button;
  struct sigaction act;
  char buffer[20];

      pid = getpid();
      printf("PID->%d\n",pid);

  button = open("/dev/button",O_RDWR);
  if(button < 0)
  {
    printf("/dev/button not found\n");
    return -1;
  }

  if (ioctl(button, IOCTL_PID, &pid))
  {
          printf("Failed\n");
          close(button);
          return -1;
      }

      sigemptyset (&act.sa_mask);
      act.sa_flags = SA_SIGINFO;
      act.sa_sigaction = sig_handler;
      sigaction (SIGBUTTON, &act, NULL);

  while(1);

  close(button);
  return 0;

}

void sig_handler(int n, siginfo_t *info, void *unused, int toggle)
{
  printf("SIGBUTTON received\n");
  printf("Info received -> %d\n", info->si_int);
  if(info->si_int)
  {
    system("amixer set Headphone,0 mute");
    //system("aplay output.wav");
    //printf("muted\n");
```

```
94    }
95    else
96    {
97       system("amixer set Headphone,0 unmute");
98       //system("aplay output.wav");
99       //printf("unmuted\n");
100   }
101
102 }
```

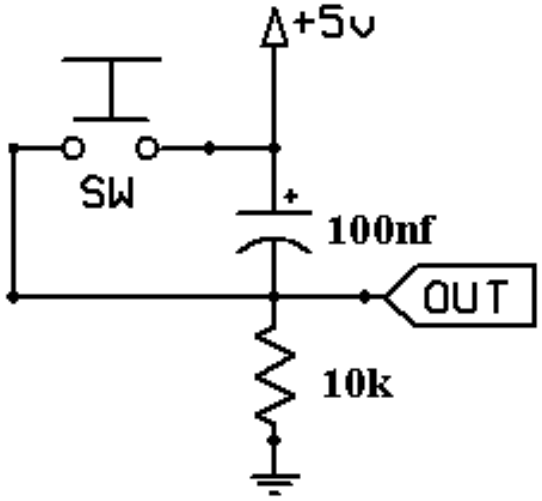
Listing 4.27: Button daemon

4.10 Button Connections

The push button is connected in a pull down configuration as it is shown in the pictures below. Pin 20 is reads the button state. It has a logic 0 value when it is not press, otherwise it goes to 1 (3.3V) whilst being pressed.



(a) Button with hardware debounce:low pass filter



(b) Button Schematic

4.11 User Interface

The graphical interface was developed in QT Designer, allowing for easier placement and interaction with its components. This tool generates an HTML file with a description of the different components and all their characteristics. This file is readed by the application to obtain the characteristics of the graphical interface for later viewing.

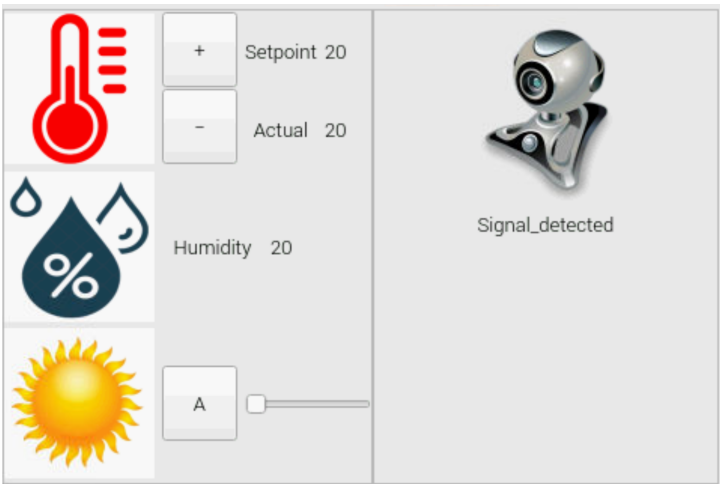


Figure 4.11: Graphical User Interface

4.12 Structure

The structure is mounted on a plastic surface that would be placed on the top of the car.

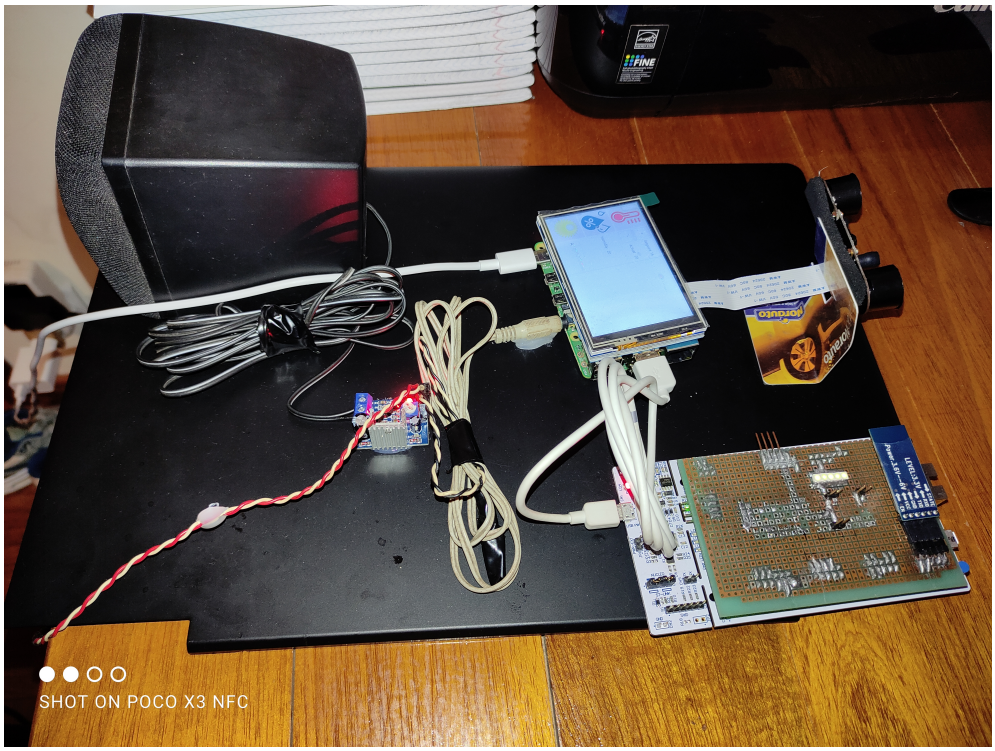


Figure 4.12: Structure

4.13 Budget

In this section is presented the budget to this project.

Quantity	Product	Price Unit (€)
1	Raspberry Pi 4B	43
1	Display 3.5"	20
1	Rasp Camera	25
1	Step-Down Module	1
1	Materials to build Model	30
Total		120

Table 4.1: Final Budget

Chapter 5

Verification

In this phase, one see if the software meets the specifications defined in the design stage.

5.1 Unit Tests

In this section, the results of the unit tests are presented. Keep in mind the following tests were performed with the raspbian operative system since one opted for allocate the time implementing and integrating everything instead of repeatedly configure the buildroot raspberry image. However, in the buildroot, it was possible to cross-compile Qt, reproduce sound, and capture video.

5.1.1 Raspberry Pi Camera

Camera	Expected Result	Real Result
Turn on Camera	Camera Opens	Camera Opens
Take a picture	Camera takes a picture	Camera takes a picture

Table 5.1: Raspberry Pi test

5.1.2 Speakers

Speakers	Expected Result	Real Result
Power the speakers and turn it on	Speakers are turned on	Speakers are turned on
Play some sound	Speakers reproduce sound	Speakers reproduce sound

Table 5.2: Speakers test

5.1.3 LCD

Speakers	Expected Result	Real Result
Connect it to Raspberry	Display some image on	Display Desktop Image

Table 5.3: LCD test

5.2 Integrated Tests

The above tests are simpler to execute while the below ones are more advanced and are only performed at the end of the developing of the system.

Intermediate	Expected Result	Real Result
Turn off motor if distance too close	Motor stops	Motor stops
Display a sign on touchscreen	Sign image alert is displayed on screen	Sign image alert is displayed on screen
Display a proximity warning on touchscreen	Warning message is displayed on screen	Warning message is displayed on screen
Reproduce sign alert through speakers	Sign audio alert is reproduced	Sign audio alert is reproduced
Reproduce proximity warning through speakers	Proximity warning audio alert is reproduced	Proximity warning audio alert is reproduced

Table 5.4: Intermediate tests

Final	Expected Result	Real Result
Detect a traffic sign	A traffic sign is detected in near real time	A traffic sign is detected
Recognize a traffic sign	The respective traffic sign is recognized	Signal recognized
Detect a car nearby in front	Car in front detected	Car in front detected
Display proper sign in real time	Sign image displayed in real time	Sign image displayed
Display warning when vehicle close in real time	Warning reproduced in near real time	Warning reproduced

Table 5.5: Final tests

Chapter 6

Conclusion

The execution of this project allowed us to deepen the knowledge acquired during the teaching activity. We believe that the verdict was successful, despite the difficulties and problems that it was not possible to solve.

6.1 Final Considerations

This project allowed the study of several themes from the choice of electronic equipment, through the configuration of the Operating System and the various necessary packages, as well as the use of neural networks associated with the vision system. As the code was developed, several problems arose, making project execution challenges. The method that helped in the organization of this project was the top-down approach. This methodology allows seeing the problem from top to bottom, detailing the different parts as the project is developed.

6.2 Future Work

As an improvement, we should train a neuronal network with a better capability to distinguish signals and also with a greater number of signals. Implement the mute button. Solve problems related to Buildroot dependencies.

6.3 Unit Tests

As a whole, there are some tests that are needed to make sure the over all of the system will work perfectly when every module is combined. These tests are composed by intermediate ones and global ones.

6.4 Gantt Diagram

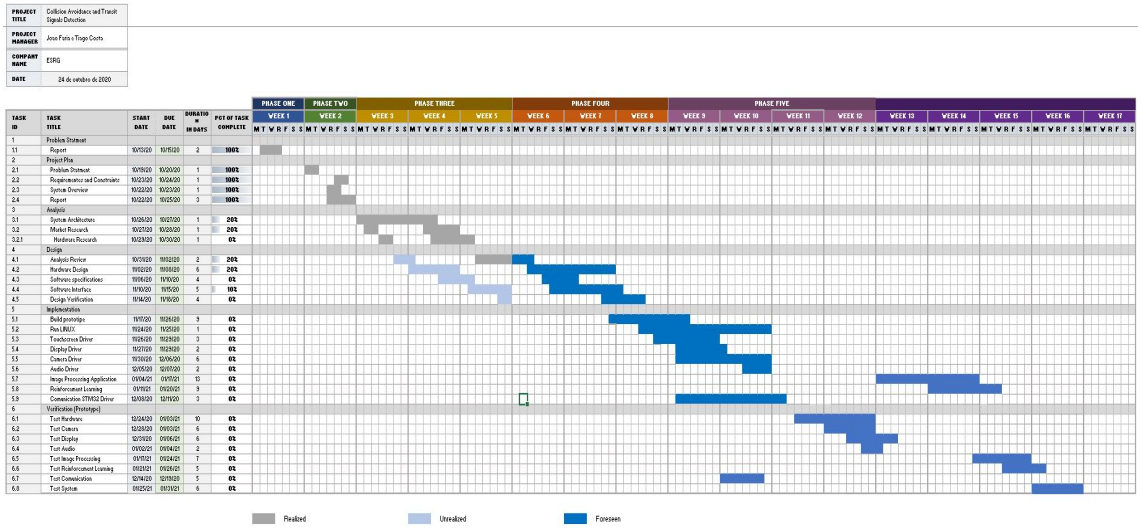


Figure 6.1: Gantt Diagram

Bibliography

[1] https://www.mordorintelligence.com/industry-reports/global-traffic-signal-recognition-market-industry

[2] https://www.youtube.com/watch?v=rfLn3m0dnEQ

[3] https://www.tensorflow.org/lite/api_docs

[4] https://www.youtube.com/watch?v=npZ-8Nj1YwY&t=627s&ab_channel=EdjeElectronics

[5] https://www.buildroot.org/

[6] https://medium.com/@leung.benson/usb-type-c-s-configuration-channel-31e08047677d

[7] https://www.totalphase.com/support/articles/200349256-USB-Backgrounds1.1.2

[8] https://www.youtube.com/watch?v=kad9wOKEQc8&ab_channel=ArasanChipSystems
https://repositorio.inesctec.pt/bitstream/123456789/3588/1/P-009-Q12.pdf
https://https://repositorio.inesctec.pt/bitstream/123456789/3588/1/P-009-Q12.pdf
https://repositorio.inesctec.pt/bitstream/123456789/3588/1/P-009-Q12.pdf

https://lmb.informatik.uni-freiburg.de/people/bahlmann/data/ba_zh_ra_pe_ko_iv2005
https://lmb.informatik.uni-freiburg.de/people/bahlmann/data/ba_zh_ra_pe_ko_iv2005

[9] https://en.wikipedia.org/wiki/Artificial_neural_network

[10] https://www.researchgate.net/publication/326974599_Evaluation_of_Deep_Neural_Networks_for_traffic_sign_detection_systems

[11] https://www.researchgate.net/publication/326974599_Evaluation_of_Deep_Neural_Networks_for_traffic_sign_detection_systems
https://repositorio.inesctec.pt/bitstream/123456789/3588/1/P-009-Q12.pdf
https://https://repositorio.inesctec.pt/bitstream/123456789/3588/1/P-009-Q12.pdf
https://repositorio.inesctec.pt/bitstream/123456789/3588/1/P-009-Q12.pdf
https://lmb.informatik.uni-freiburg.de/people/bahlmann/data/ba_zh_ra_pe_ko_iv2005
https://lmb.informatik.uni-freiburg.de/people/bahlmann/data/ba_zh_ra_pe_ko_iv2005