UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Control of an autonomous robot using machine learning techniques implemented on a GPU device

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Sergi Mor Cases

In partial fulfillment
of the requirements for the master in
ELECTRONIC **ENGINEERING**

Advisor: Joan Manuel Moreno Arostegui
Barcelona, 20 October 2021

Contents

List of Figures

Listings

List of Tables

Revision history and approval record

Revision	Date	Purpose
0	06/09/2021	Document creation
1	07/10/2021	Document revision
2	20/10/2021	Final document

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Sergi Mor	sergimorcases@gmail.com
Joan Manuel Moreno	joan.manuel.moreno@upc.edu

Written by:		Reviewed and approved by:	
Date	06/10/2021	Date	21/10/2021
Name	Sergi Mor	Name	Joan Manuel Moreno
Position	Student	Position	Project Supervisor

Acronyms

AI Artificial intelligence
GPU Graphical Processing Unit
CPU Central Processing Unit
CNN Convolutional Neural Network
OI Open Interface
VPU Vision Processing Unit

Abstract

In a world where artificial intelligence is so consolidated in many fields and in different applications, there is a need to understand how it works and how we can improve these algorithms in order to optimize them. In this project we will focus on the autonomous control of a robot by face detection.
To implement this functionality, we have compared different Convolutional Neural Networks. After a detailed study of each of these models and comparing their results, it has been decided to use the FaceNet system for face detection algorithm. To improve the performance and the processing of the video frames, a GPU, Nvidia Jetson Nano, will be used. Finally, by means of the Jetson Nano board we will control the iRobot Roomba 600 robot through commands that will be sent through a serial port, which the robot will receive to manage its actuators and to be able to move following our face.

1 Introduction

This section will present the different requirements of the project, explain briefly where this project comes from and why; and the software and hardware used in the previous project.

1.1 Purpose of the project

For years, technology has been growing exponentially and with it the different technologies to make people's daily lives easier. As a result, many tedious and monotonous processes have been replaced by autonomous work carried out by machines, robots or computers. Previously, decisions about a process were made by a person or a group of people, but thanks to artificial intelligence algorithms, many of these decisions can be made by a computer. That is why nowadays autonomous decision making has become widespread in many fields, which is why it is growing.
The autonomous control of robots or vehicles is a widespread field where machine learning algorithms are applied, hence the interest in controlling a robot, a Roomba, by means of a microcontroller.
This is where this project has come from, being able to control a robot by means of machine learning algorithms using a GPU.

Theoretical objectives

- Study different autonomous learning algorithms for face recognition.

Practical objectives

- Implement face recognize algorithm.
- Control a robot by means of the face and be able to follow it.

1.2 Requirements

Control of an autonomous robot using machine learning techniques implemented on a GPU device.

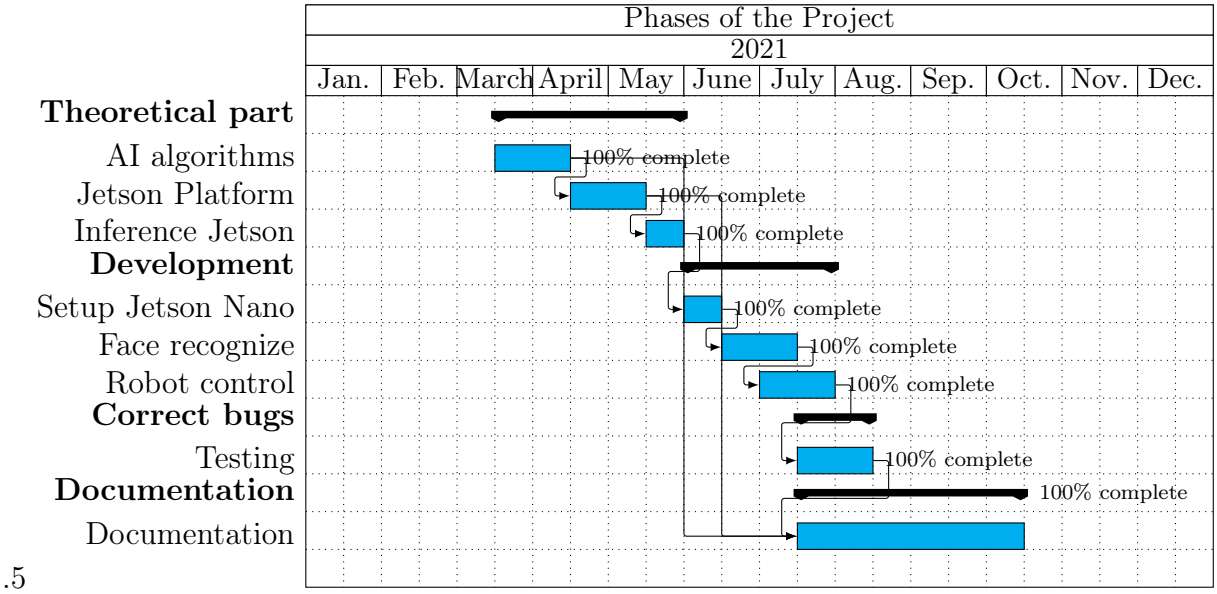
- Face recognize with a Nvidia Jetson Nano board using a Raspberry Pi Camera
- Control the robot depending on the position of the face in the image frame.

This project is the continuation of a project carried out in one of the classes of the Master of Electronic Engineering taught at the Polytechnic University of Catalonia (UPC).
The main objective of the laboratory assignment was to use a hardware module in order to accelerate the implementation of a machine learning algorithm for face recognition. This module was connected to a Raspberry Pi and by means of a camera module to be able to recognise the face.
The module for accelerating machine learning algorithms is an Intel Neural Compute Stick 2 (Intel NCS2). Combines the hardware-optimized performance of the latest Intel Movidius Myriad X VPU and Intel's OpenVINO Toolkit distribution to accelerate deep neural

network-based applications so this processor is used to optimise complex calculations for image processing[1].
This USB was connected to a Raspberry Pi 3, which collected live image frames via a camera and processed them with the Intel NCS2 module in order to control the Roomba 600 Roboto. This project aims to replace the Intel NCS2 module and the Raspberry Pi 3 with a single hardware module. For this purpose it was decided to use the Nvidia Jetson Nano board. This module is small, very powerful and has high computational capabilities that replaces the Intel module; it allows us to run multiple neural networks in parallel for image classification or object recognition among other applications.

1.3 Gantt Diagram

In the figure 1, by means of a Gantt diagram, we can see how we have planned the weeks for developing the project.



.5

Figure 1: Gantt diagram of the project

2 State of art

2.1 Introduction

When studying Machine Learning you will come across many different terms such as artificial intelligence, machine learning, neural network, and deep learning. But what do these terms actually mean and how do they relate to each other? Below we give a brief description of these terms and the figure 2 shows an overview of how they are related:

- Artificial Intelligence(AI): A field of computer science that aims to make computers achieve human-style intelligence. There are many approaches to reaching this goal, including machine learning and deep learning.
- Machine Learning: A set of related techniques in which computers are trained to perform a particular task rather than by explicitly programming them.
- Neural Network: A construct in Machine Learning inspired by the network of neurons (nerve cells) in the biological brain. Neural networks are a fundamental part of deep learning.
- Deep Learning: A subfield of machine learning that uses multi-layered neural networks. Often, "machine learning" and "deep learning" are used interchangeably.

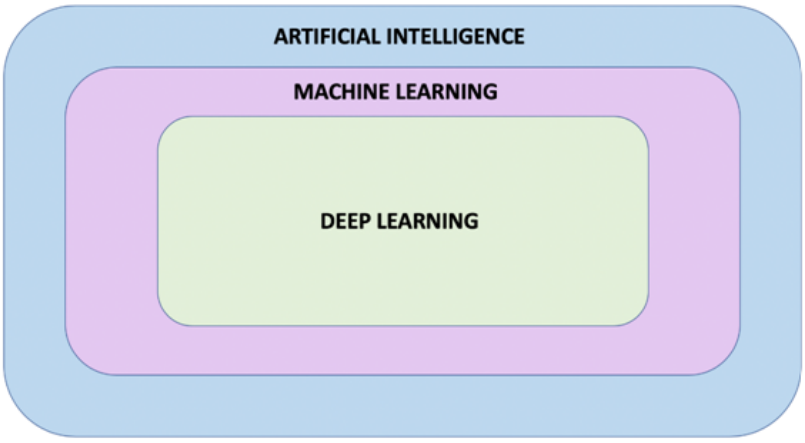


Figure 2: Global view Machine learning

Machine learning and deep learning also have many subfields, branches, and special techniques. A notable example of this diversity is the separation of Supervised Learning and Unsupervised Learning.
To over simplify, in supervised learning you know what you want to teach the computer, while unsupervised learning is about letting the computer figure out what can be learned. Supervised learning is the most common type of machine learning.

2.2 Convolutional Neural Network (CNN)

Before explaining the different architectures we have used in this project, we will explain what Convolutional Neural Networks are.
These algorithms are used for autonomous learning. This type of Neural Network with supervised learning contains different hidden layers that each of these layers is in charge of detecting different patterns as shown in the figure 3, making each layer more specific depending on what is left to be filtered or detected. These layers mimic the visual cortex of the eye that allows us to compare different features.

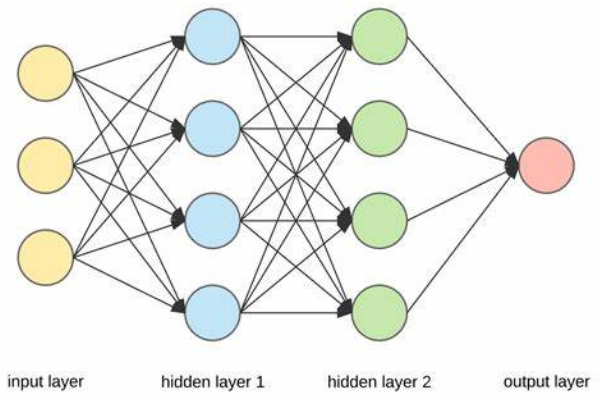


Figure 3: Convolutional neural network diagram

The function of this neural network is to recognise objects from an image or a video by inputting it with previously selected images of different objects, be they faces, cars or whatever we want it to learn to recognise. The image we want to recognise or identify is converted into a matrix of pixels. For example, FaceNet uses a 160x160x3 size face image as input and being the output a 128 dimensional float vector which can be quantified as each image is represented as a 128 dimensional byte vector.
The convolution process consists of taking groups of pixels and making the scalar product with the kernel, which consists of a 7x7x3 matrix in the case of the Zeiler & Fergus model that we will discuss later. In the model we have used, the activation function is the rectified linear (ReLU) unit as the no-linear activation [2]. This function returns 0 if the input values are negative and on the contrary it returns positive values as can be seen in the following figure 4. And the function is:

$$f(x) = max(0, x)$$

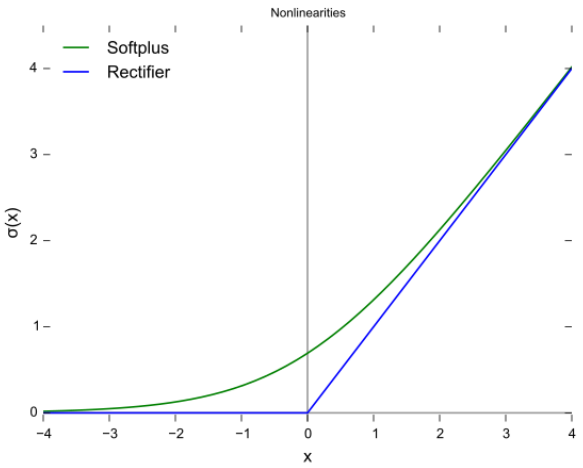


Figure 4: Rectified Linear Units graphic(ReLU)

The next layer is indispensable because we need it to be able to reduce the number of neurons, i.e. the number of calculations. Pooling layer is in charge of reducing the size of the data by combining the outputs of this first layer into a single neuron or more than one neuron. There are two different ways of pooling, L2-norm pooling and max pooling. On one side, we will use a 3x3 matrix normalising its value and on the other side, with a matrix, also 3x3, the largest value is taken. In this way we will reduce considerably the total number of neurons, depending on the model this process will be done more than once.
This process will be repeated several times because as we add layers we will be able to identify objects, people and not only geometric shapes. In the following figure 5 we can see an example with a general overview:

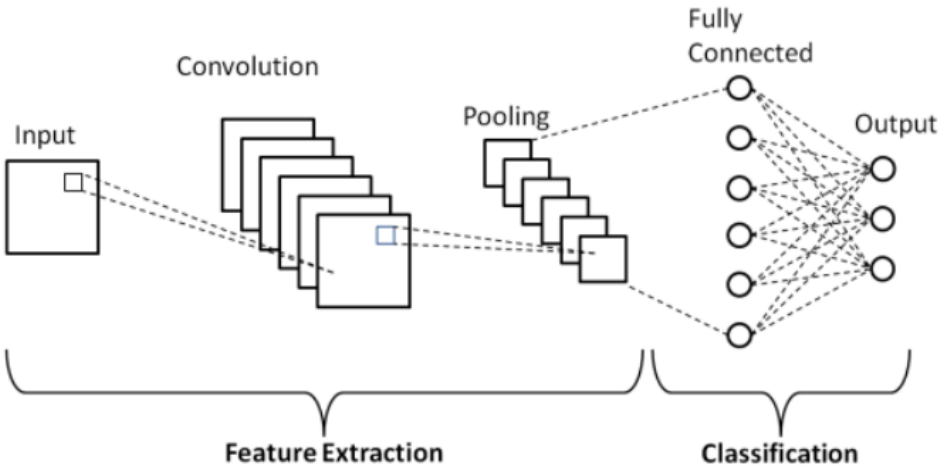


Figure 5: Architecture of a CNN [3]

2.3 Face recognize

Facial recognition is a technology for recognising faces and identifying people. In the last few years it has been used extensively for ID verification, public security and other purposes.

Facial recognition was first developed in the 1960s by Woody Bledsoe, Helen Chan and Charles Bisson. The system implemented by these mathematicians and scientists was not self-learning, but required a third party to locate common features in order to calculate the distance between common points, such as the eyes, mouth, nose, and mouth.

As the years have gone by, technology has evolved and with it the techniques for recognising faces. Nowadays there are several autonomous learning systems with a very high level of accuracy. One system with accuracy of 99.6 % was developed by Google and is called FaceNet: A Unified Embedding for Face Recognition and Clustering [2]. Another widely used classifier is the Haar Cascade Classifier, which recognises the front of the face. In the following we will explain these two models and their main features and why we have chosen one over the other.

2.3.1 Haar Cascade Classifier

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed in the paper, Rapid Object Detection using a Boosted Cascade of Simple Features [4].
This algorithm needs as many positive images as negative images; a positive image is an image of a face where different features are extracted, shown in the figure 6, these features are defined by the kernel. There are a total of three features: edge feature, line feature, four-rectangle feature.

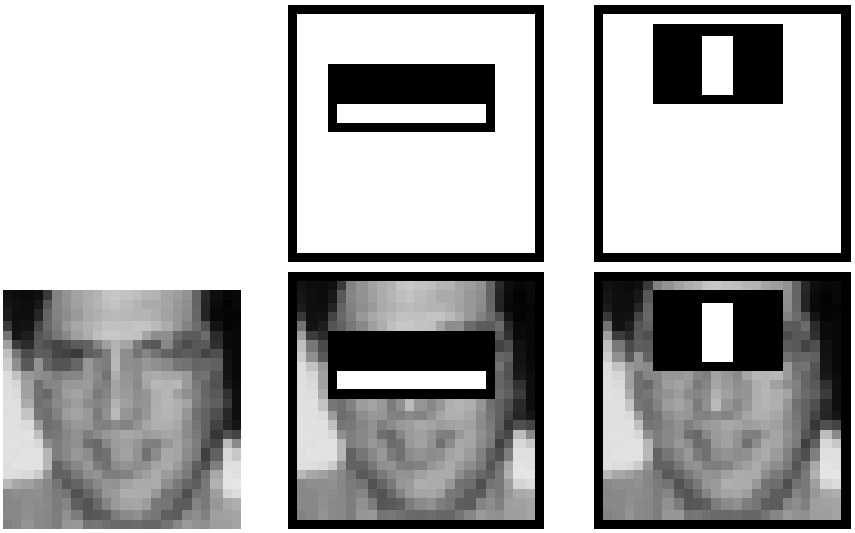


Figure 6: Haar Cascade Classifier features [4]

The value between two features of two rectangles is the difference between the sum of their respective regions, having the same size and the same shape. Once this computation is done, we will proceed to calculate a three-rectangle feature, which will be the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles [4].
For example with a window of 24x24 pixels, there are a total of 162336 features. This would result in a very slow process and would require a lot of calculations. For this purpose AdaBoost algorithm is used to select a small set of features and train the classifier.
To optimise these calculations, and because most of an image does not contain a face, we first check if a face is not a region of a face. To do this they introduce the concept of Cascade of Classifier, the features are grouped in different stages. According to the authors of the paper there are a total of 38 stages with over 6000 features [4].The following figure 7 shows the features used to detect the face in each one of the different stages.

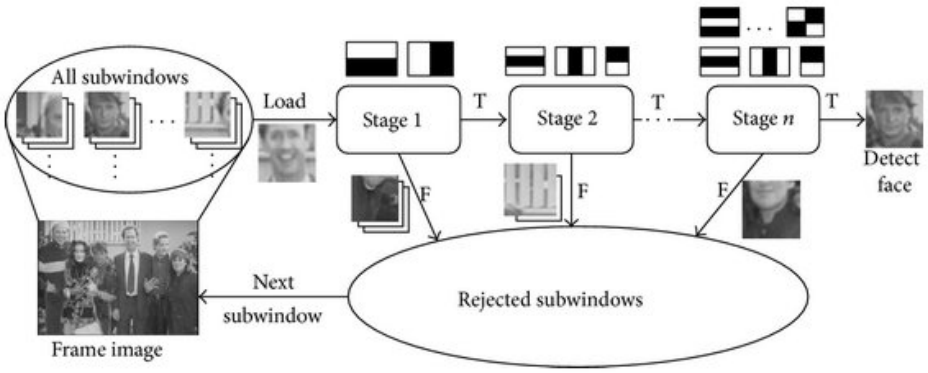


Figure 7: Haar Cascade Classifier architecture [5]

2.3.2 FaceNet

Previously, when introducing some deep learning concepts, reference has been made to this easy recognition system proposed by google in the paper entitled FaceNet: A Unified Embedding for Face Recognition and Clustering [2].
Two datasets have been used for this system: Labeled Face in the Wild (LFW) and Yotube Face Database.
In this paper it compare two different deep architectures but for our application we will treat them as black boxes and only compare the results of using one architecture and the other.
The network consists of a batch input layer and a deep CNN followed by L2 normalization, which results in the face embedding. This is followed by the triplet loss during training.See figure 8

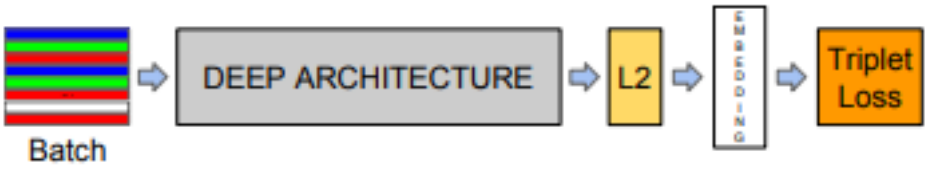


Figure 8: FaceNet architecture [2]

FaceNet proposes a solution using a mapping of face images in a Euclidean space such that the distances correspond to similarities in the face.
The Triplet loss function consists of comparing an entry either an image or a video (anchor) with a positive (true) or a negative (false) entry. On the one hand, the distance between the anchor and the positive input is minimised, while the distance between the anchor and the negative input is maximised. The motivation for using triplet loss tries to enforce a margin between each pair of faces from one person to all other faces. This allows not only face detection but also discrimination with other individuals, thanks to the embedding space generated which can then be used to identify people.
This embedding space is represented by $f(x) \in \mathbb{R}^d$ and it embeds an image x into d-dimensional Euclidean space. Generating all possible triplets, that are easily satisfied. The different triplets will not be used to train the model. For this we will have to select the hard tripets that will help to improve the neural network [2].

Algorithm 1 Loss minimized function

1: $L = \left\|x_a^i - x_p^i\right\|_2^2 + \alpha > \|x_a^i - x_n^i\|_2^2, \forall(x_a^i, x_p^i, x_p^n) \in \tau$

Where α is a margin that is enforced beetwen positive and negative pairs. τ is all the possible triplet in the training.

The neural networks studied in this paper are Zeiler & Fergus and NN2 inception. The two neural networks used in the study require a total of 1.6B FLOPS. FLOP is a measure to calculate the performance of a computer used in scientific computation [2].
In the table below we can see the different acuracy results between the two networks and different configurations.

Architecture	VAL
NN1 (Zeiler & Fergus 220x220)	87.9% ± 1.9
NN2 (Inception 224x224)	89.4% ± 1.6
NN3 (Inception 160x160)	88.3% ± 1.7
NN4 (Inception 96x96)	82.0% ± 2.3
NNS1 (mini Inception 224x224)	82.4% ± 2.4
NNS2 (tiny Inception 140x116)	51.9% ± 2.9

Table 1: Accuracy of different Networks

2.4 Architecture GPU

GPUs have become a very popular technology in the last decade. Having many different applications, such as video processing, gaming, cryptocurrency mining and artificial intelligence. Its use has become so widespread thanks to the great computing power, being able to paralyse different processes. GPUs were originally designed to render 3D graphics, but over the years their use has expanded along with their performance capabilities.

2.4.1 GPU VS.CPU

To understand the difference between a GPU and a CPU is to compare the way they process tasks. A CPU has different cores optimized for sequential serial processing, for example 8 cores. While a GPU has thousands of cores to process parallel workloads efficiently.
The CPU is a microprocessor designed to execute sequences of instructions, called thread, as fast as possible and it just can execute a few of tens in parallel. Its works is divided into four fundamental steps according to the Von Neumann architecture: fetch, decoding, execution and writeback.
On the other hand, GPU is dedicated to floating point operations, and is designed to reduce the workload of the central processor. High segmentation, with functional units, allows faster processing of pixels. In addition, its faster memory allows the management of intermediate results of operations in a more agile way.

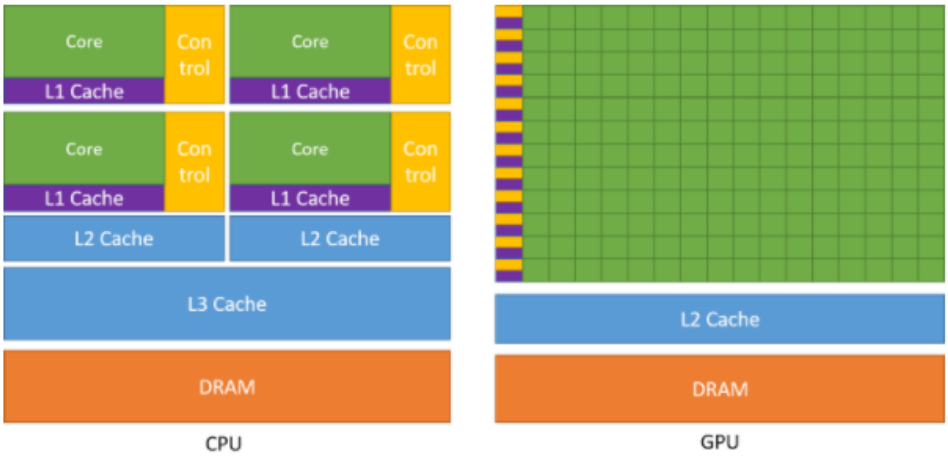


Figure 9: Differences between CPU and GPU components [6]

In conclusion, the main differences between a GPU and a CPU are in their design. The CPU design is a few very complex cores while the GPU has thousands of very simple cores. While a GPU has independent fundamental computing units, the inherent parallelism is one of the key aspects of a GPU and its difference from a CPU. Another important difference is that the GPU has more transistors for data processing instead of cache storage and flow control. See figure 9.

2.4.2 CUDA architecture

CUDA was first developed in 2006 [7], over the years it has been improved to cover different applications and publications making it easier to use.
The CUDA platform is made up of different software layers that give us access to different components of the platform. We can access each of these components through instructions and libraries for the execution of compute kernels. The compute kernels are precompiled routines for high-performance accelerators such as GPUs. These routines share CPU execution units with vertex shaders and pixel shaders (GPUs).
CUDA platform is designed to enable developers to use C++ as a high-level programming language.
Other programming languages can be used to program interfaces, libraries... such as FORTRAN, DirectCompute, Open ACC. In the following figure 10 we can see the supported languages.

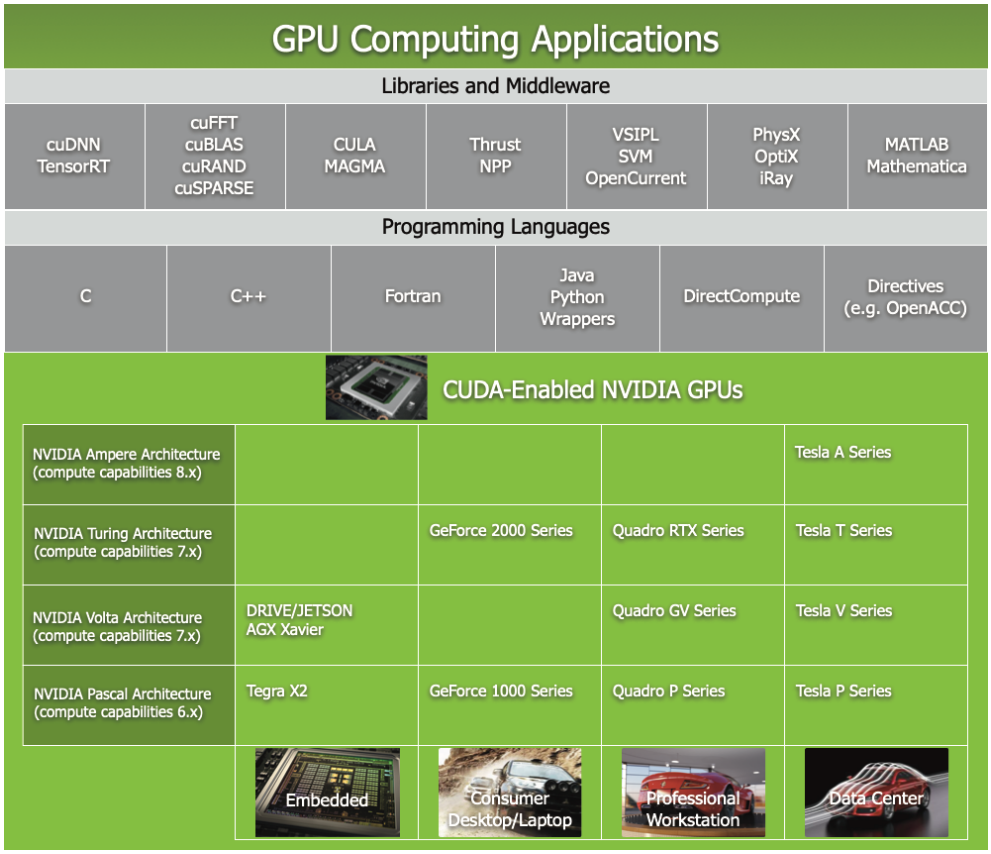


Figure 10: Languages used in the different components of CUDA [6]

2.4.3 Programming model

CUDA is a hardware and software platform that allows Nvidia GPUs to run programs written in a variety of programming languages, including C, C++, Fortran, and others. A CUDA program calls parallel functions known as kernels, which are executed on many parallel threads. These threads are organized by the programmer or compiler. These threads

are organized into thread blocks, which are then organized into a grid of thread blocks.

Each thread in a thread block executes a kernel instance. Each thread also includes a program counter, registers, private memory per thread, and input and output results.
A thread block is a collection of threads that are running at the same time and can cooperate by using synchronization and shared memory barriers.
Through synchronization and shared memory boundaries, they are able to collaborate with one another. Within its network, each thread block has a unique ID. If there are too many threads in each block, there will be fewer threads; if the number of threads per block is too large, there will be fewer blocks per multiprocessor running in parallel.
A multiprocessor can only physically handle a certain number of blocks and threads simultaneously. To get the most out of it, the number of threads per block on the GPU should be a multiple of the maximum number of threads supported by each multiprocessor, and big enough to allow the blocks to communicate with each other. Each multiprocessor must be able to support the blocks, and the blocks must be large enough to cover all available resources. It is strongly advised that the blocks must be a multiple of two rather than a little quantity. You should also aim to include a large number of blocks to cover latency delays, so that the multiprocessors continue to operate even if a block is waiting for whatever reason. If it is possible, this amount should be higher than the double of the total number of multiprocessors in the device. In the figure 12 we can see how the different multiprocessors are connected between each other.
A grid is a collection of thread blocks that all run the same kernel, read from global memory, write results to global memory, and synchronize across the kernel's dependent calls. Below the blocks are the warps, which are a new grouping of threads that make up the fundamental execution unit, to make program execution more concrete and optimized [8]. The following figure 11 shows how the threats are distributed by blocks.

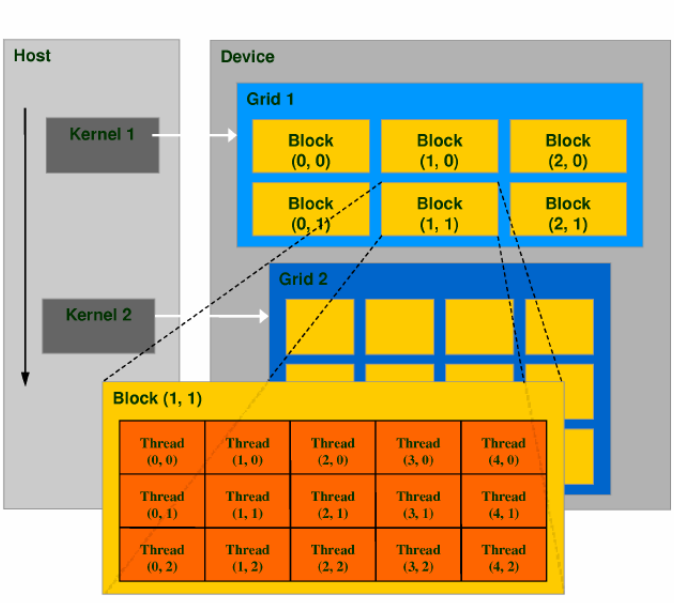


Figure 11: Thread organization in CUDA programming model [9]

The warps are currently made up of 32 threads, which implies that the same instruction will be performed by groups of threads of this size at the same time. When this is not possible due to the fact that different threads within the same warp have different instructions, each thread will execute its instruction in parallel with the others but in series with the rest of the threads with different instructions, greatly reducing the program's performance and parallelism.
Each thread has its own memory space in the CUDA parallel programming architecture, which is used for register dumps, function calls, and automated C vector variables. Register dumps, function calls, and automated vector variables are all supported in C. Each thread block also contains a shared memory area for communication between threads. A shared memory space is also used for inter-thread communication, data sharing, and result sharing for parallel computations in a block of threads. After global synchronization by the kernel, grids of thread blocks exchange results in the global memory space.
The system is divided into two parts from the perspective of the programmer: the primary processor (host), which is the CPU (in our project, Quad-core ARM A57), which will execute the program's main code; and the second component, which is made up of one or more devices, which are the GPUs. We just have one GPU on the board, thus the system will be considerably simpler.
Because GPU processors are significantly less powerful than CPU processors, both components of a program must be employed to obtain the best results. to get the most out of a program, which is directly impacted by the code optimization in both blocks.
A CUDA program is made up of many stages that can be run on the host or on the devices. Phases with little or no data-level parallelism are written in code that runs on the host processor, whereas phases with a lot of data-level parallelism are written in code that runs on the available devices.
Because the GPU is separate from the CPU and hence from the system's main memory, it will require explicit data transfers. To operate with in parallel functions, the system's main memory will require explicit data transfers, this results in a time delay that has a significant impact on the program's speed, albeit it is still less than the time it would take to run that section of the code sequentially on a CPU.
The host is in responsibility of reserving and freeing memory on each device, as well as performing memory transfers, during this procedure. The host is in charge of transferring memory between the two devices and determining when each device begins to execute each of the kernels that have been programmed is started by the device [8].

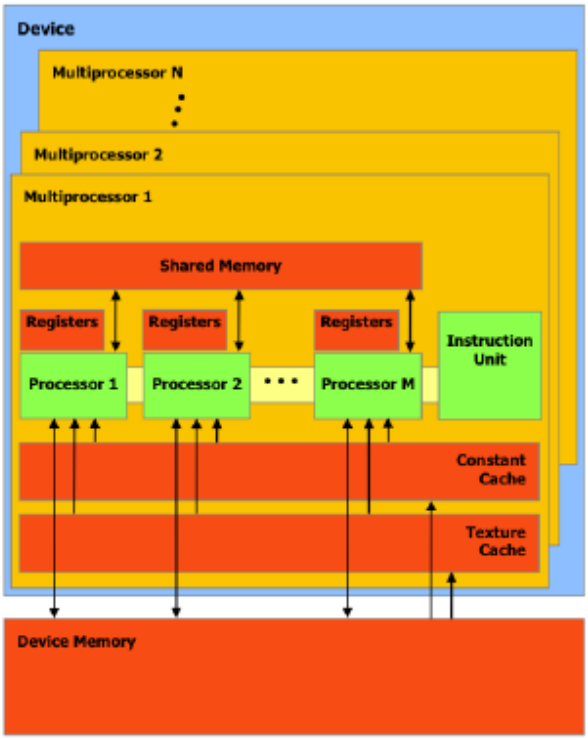


Figure 12: GPU multiprocessor components [7]

3 Methodology

3.1 Hardware

1. Jetson Nano Developer Kit
2. Computer with Internet Access and SD card port
3. MicroSD Memory Card (32GB UHS-I minimum)
4. Compatible 5V 4A Power Supply with 2.1mm DC barrel connector
5. USB cable (Micro-B to Type-A)
6. USB to serial port converter cable
7. Raspberry Pi Camera Module v2
8. iRobot Roomba 600

3.1.1 Jetson Nano Developer Kit

Jetson Nano, see figure 13, is an Nvidia-developed CPU with a compact size and a very powerful design for working with AI. Behind this board there is a large community, where you can find learning courses, forums, introduction videos and more. It is designed for learning about machine learning, or for professional use as its platform provides a set of frameworks to reduce complexity and effort for developers.
In order to start working with it. We have to download the latest OS image called Jetpack SDK, which includes the Linux Driver Package (L4T) with Linux operating system and CUDA-X accelerated libraries and APIs for Deep Learning, Computer Vision, Accelerated Computing and Multimedia. All the material, documentation, steps to follow to be able to use it are very well documented on their website [10].
For our project we have used JetPack 4.6, which includes the latest version of CUDA, cuDNN and TensorRT, which we will talk about later [11].

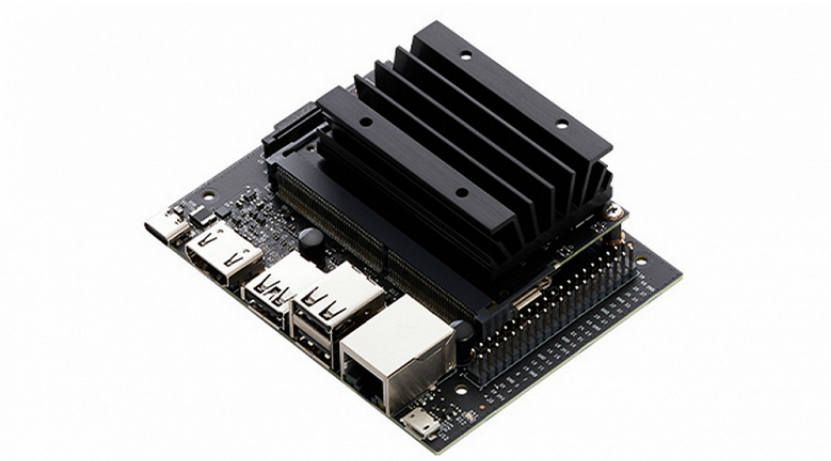


Figure 13: Jetsno nano developer kit [11]

Component	Characteristic
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I2C, I2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector
Portable battery 10000 mAh	

Table 2: Component and the characterstic

3.1.2 Camera

Figure 14: Raspberry Pi CSI Camera [13]

The CSI camera, see the figure 14, uses a module called the Sony IMX219 8-megapixel sensor, IMX219 is a 1/4 8MP MIPI CSI-2 image sensor [12]. This module offers high quality video, where we can take both photos and videos. Because it is a very popular component for Raspberry developers or hobbyists there is a large community behind it where we can find a large number of examples and projects for both beginners and more advanced levels. It supports 1080p30, 720p60 and VGA90. The camera can work for all Raspberry models and also this module is supported for Jetson Nano and Xavier NX. We have used the model BO191 with a field of view of 62.2°.

3.1.3 iRobot Roomba 600 [14]

The Roomba Open Interface (OI) is a software interface that, by means of commands, we can modify its behaviour and control the different sensors that it includes. These commands allow us to control the different actuators that it has, as well as activate its sensors or commands to clean. The figure 15 shows what the iRoobot Roomba 600 looks like.

Figure 15: iRobot Roomba 600 [14]

3.1.3.1 Communication port

It uses a Roomba's External Serial Port Mini-DIN Connector Pinout to be able to communicate and send commands with it through a microcontroller or a PC. Below we will explain its electrical characteristics as well as its pinout.

Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 – 5V Serial input to Roomba
4	TXD	0 – 5V Serial output from Roomba
5	BRC	Baud Rate Change
6	GND	Roomba battery ground
7	GND	Roomba battery ground

Table 3: Pinout iRobot Roomba 600

In order to communicate with our microcontroller, we will need a USB to 8 Pin Mini Din Male Serial Adapter cable.
This adapter contains an FTDI FT232RL which converts from USB to Serial UART interface with an output clock generator. This chip can handler USB protocol to a UART interface for 7 or 8 data bits, with 1 or 2 stop bits. The data transfer rate ranges from 300 baud to 3 megabaud.

Serial Port communication
In the following table 4 shows the configuration of the serial port.

Baud	115200
Data bit	8
Parity	None
Stop Bit	1
Flow Control	None

Table 4: Serial Port configuration

3.1.3.2 Roomba operation modes

The Roomba OI has four operating modes: Off, Passive, Safe and Full.
Off

- After power on, the roomba enterns in off mode waiting for the start command

Passive

- Request sensor data
- Receive sensor data
- No change in actuators (motors, speakers...)

Safe
Almost full control with a few exceptions, the Roomba will stop if some conditions happened:

- Detection of a cliff
- Detection of a wheel drop
- Charger plugged in and powered
- Detect an object

Full

- Full control over the Roomba

3.1.3.3 Commands used for controlling the Roomba

Start

Before sending any command to he OI, we must send this Start command.

CMD	D0	D1	D2	D3
0x80	-	-	-	-

Safe mode

This command initializes the OI and it puts into Safe Mode. So now we are able to control the OI and we must set the mode (Passive, Safe, Full)

CMD	D0	D1	D2	D3
0x83	-	-	-	-

Close communication

This command finishes the communication with the microcontroller. So the robot will no longer respond to commands.

CMD	D0	D1	D2	D3
0xAD	-	-	-	-

Drive

This command requires 4 bytes of data. The first two bytes refer to the speed of the wheels and are measured in millimetres per second (mm/s), the velocity will be the average between these two values; while the next two bytes are used to measure the radius at which we want to turn the robot and the longer radii make Roomba drive straighter, while the shorter radii make Roomba turn more. The radius is measured from the center of the turning circle to the center of Roomba.

This command is available in Safe or Full mode
The velocity ranges are: -500 to 500 mm/2
The turning radius ranges are: -2000 to 2000 mm/s
There are some special cases:
- Straight = 32768 or 32767 = 0x8000 or 0x7FFF
- Turn in clockwise move = -1 = 0xFFFF
- Turn in counter-clockwise move = 1 = 0x0001

Forward drive

This command controls the Roomba's drive in a fordware move.

CMD	D0	D1	D2	D3
0x89	0x00	0x00	0x30	0x00

D0	Velocity high byte (mm/s)
D1	Velocity low byte (mm/s)
D2	Radius high Byte (mm)
D3	Radius low Byte (mm)

Backward drive

This command controls the Roomba's drive in a backward move.

CMD	D0	D1	D2	D3
0x89	0xFF	0xD0	0x00	0x00

Counter-clockwise drive

This command controls the Roomba's drive in a counter-clockwise direction.

CMD	D0	D1	D2	D3
0x89	0x00	0x20	0x00	0x01

Clockwise drive

This command controls the Roomba's drive in a clockwise direction.

CMD	D0	D1	D2	D3
0x89	0x00	0x20	0xFF	0xFF

Stop drive

This command stops the Roomba.

CMD	D0	D1	D2	D3
0x89	0x00	0x00	0x00	0x00

3.2 Getting started with Nvidia Jetson Nano

In this section we will talk about the material we need for our project and the first steps to be able to use the Nvidia Jetson Nano. As well as the installation of the OS, the different steps to install the necessary libraries to improve the efficiency in the neural networks that we will use in the Jetson platform; improving performance and power efficiency using graphics optimisation, kernel fusion and FP16/INT8 precision.

3.2.1 Write image to microSD card

First of all we need to download the Jetson Nano Developer Kit SD Card Image. Once we have the image downloaded, we need to format the SD card and load the image. To format the card we have used the SD Card Formatter program. The next step is to load

the image to the SD card, for this we have used a program called Etcher. We select the desired image and the device we want to flash. Once flashed, we can take out the microSD and connect it to our microcontroller. Shown in figure 16.

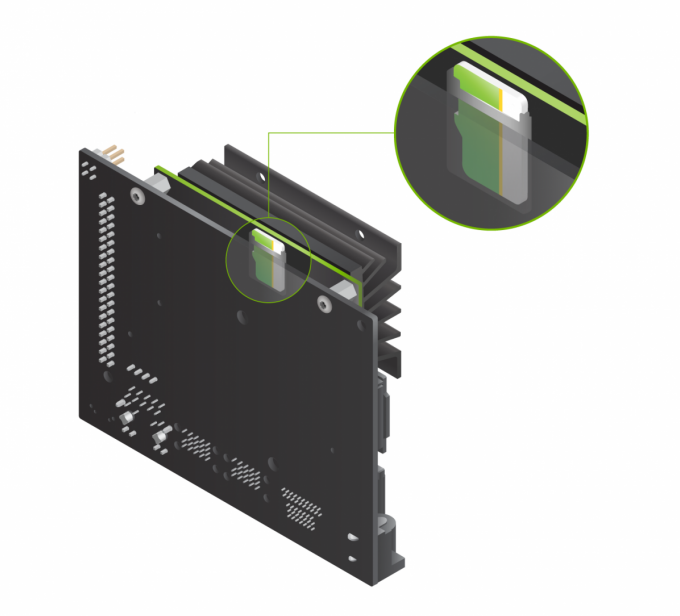


Figure 16: MicroSD insertion [11]

3.2.2 Setup and first boot

There are two ways to use the Nividia Jetson Nano, one is through a monitor, a mouse, a keyboard, ethernet cable to connect to the internet or a wifi adapter and DC power supply, which can be either through a micro-USB cable type C or through a power supply; and the other way is through a PC, a micro-USB cable and a power supply through the ssh connection.
To setup the alternate barrel power supply, use the following steps.

1. Insert the 2-pin jumper across the 2-pin connector, J48, located next to the MIPI CSI camera connector or behind the barrel power port (B01 version). This enables the DC barrel power supply.

2. Connect your DC barrel jack power supply (5V/4A). The Jetson Nano Developer Kit will power on and boot automatically. 3. A green LED next to the Micro-USB connector will light as soon as the developer kit powers on.

To configure the system and its correct installation we will start by connecting all the peripherals to the microncontroller and we will turn on the board.
When we turn on the microcontroller a green LED will light up, the first time we initialize it, we will have to make some initial configurations as well as define our user and password.

- Review and accept the NVIDIA End User License Agreements.
- Select the language, time zone...

- We must set our username and password
- Select APP partition size, a max size is suggested to use

3.3 Jetson Inference

As mentioned above, NVIDIA has a large community behind its platform that provides frameworks and libraries to make developers' work easier. This is what Hello AI World is all about, where NVIDIA provides us with a starting point to work with different demos and a set of deep learning inference in real time for object detection and image classification. This platform provides tutorials for deploying AI and computational vision for the Jetson Nano, Jetson AGX Javier, Jetson TX1 and TX2. This platform will provide us with the different tools needed as well as pretrained network models including ImageNet and DetectNet, which we will discuss later.

3.3.1 How to set up and build the project

As mentioned above, NVIDIA provides a TensorRT-accelerated deep learning networks project for object detection and semantic segmentation. This inferencing library can be supported in both Python and C++. It also comes with several pre-trained DNN models that are ready to use. There are a variety of pre-trained models depending on the application. For our project we are going to use the Facenet-120 model designed by Google Inc.

See the following command :

```
$ sudo apt-get update
$ sudo apt-get install git cmake libpython3-dev python3-numpy
$ git clone --recursive https://github.com/dusty-nv/jetson-
    inference
$ cd jetson-inference
$ mkdir build
$ cd build
$ cmake ../
```

When we call cmake automatically the tool will run the following figure 17 were we will able to chose which models we want to download, make sure that FaceNet is selected:

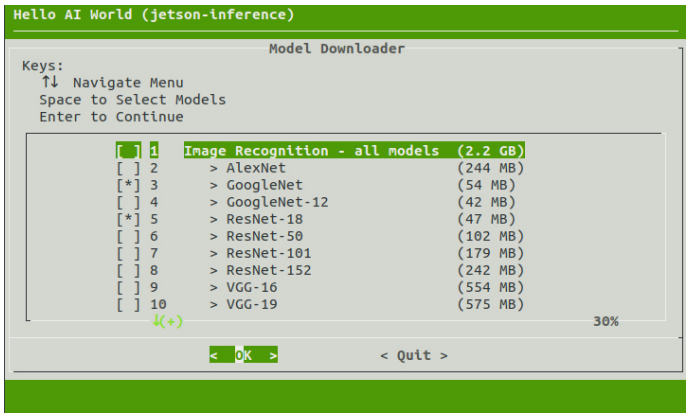


Figure 17: Model's installation

In the following figure 18 we have to chose which Pytorch version we want to install in our case we are using Python 2.7 so the first option will be selected.

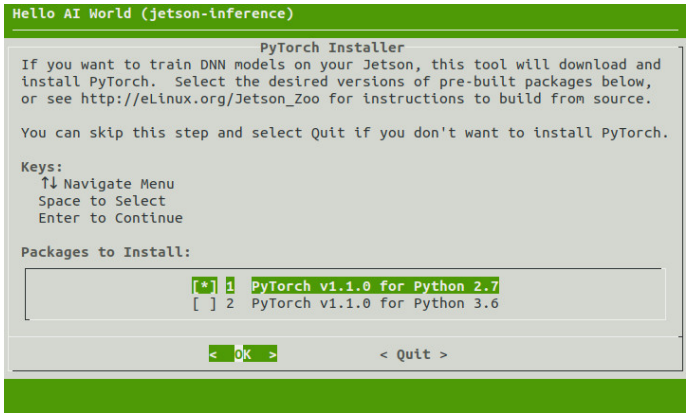


Figure 18: Pytorch installation version

During the development of the project, we found that the CSI camera rotated the image by 180°. This is a problem located in the libraries provided by NVIDIA. To solve this problem, before making the project, we have to change a parameter located in the source code of the project.In the file jetson-utils/camera/gstCamera.cpp change the line 139 for:

```
if( mOptions.flipMethod == videoOptions::FLIP_ROTATE_180 )
```

And now it is time to compile the project.

```
$ cd jetson-inference/build                    # omit if working
    directory is already build/ from above
$ make
$ sudo make install
$ sudo ldconfig
```

3.3.2 TensorRT

The repository we have downloaded above uses NVIDIA TensorRT to have an efficient implementation on the neural networks we will work with, improving their performance and an improvement in the management of the energy consumed for the Jetson platform. In this section we will talk about TensorRT, how to use it and the different applications it has.

3.3.2.1 What is TensorRT?

"The core of NVIDIA TensorRT is a C++ library that facilitates high-performance inference on NVIDIA graphics processing units (GPUs). It is designed to work in a complementary fashion with training frameworks such as TensorFlow, PyTorch, MXNet, and so on. It focuses specifically on running an already-trained network quickly and efficiently on a GPU." [18]

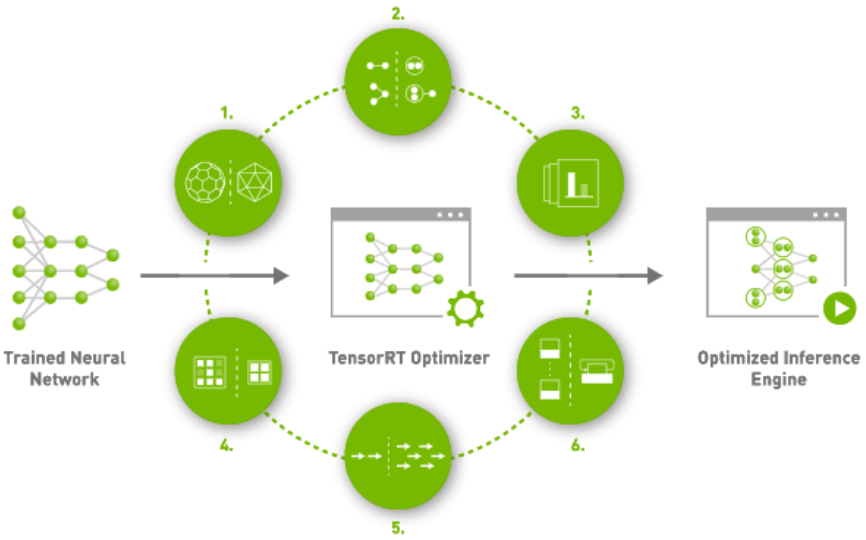


Figure 19: TensorRT [15]

TensorRT optimises trained DNN by combining different layers of it and optimising the kernel depending on the type of application we want. For example, if we are working on a system that needs power efficiency, the system will be optimised for low power consumption; or to improve latency or memory consumption.
When we talk about deep learning inference, we have to take into account five important factors for the software: Throughput, efficiency, latency, accuracy and memory usage.
So, TensorRT solves these problems that abstract away specific hardware details and an implementation that optimizes inference for high throughput, low latency, and a low device-memory footprint.
Many sectors can benefit such as autonomous vehicles, robotics, video analysis, automatic speech recognition. There are already a wide variety of companies using it. See the figure 19 to see the tensorRT overview.

3.3.2.2 Steps to use TensorRT The workflow to train and develop a neural network can be divided into 3 phases.

Phase 1:
In the first phase of the problem we have to define the inputs, the desired outputs as well as the loss function we will require. On the other hand, we need to have a good database, well labelled, with a large amount of data either to train the model or to test it. Once the data set is validated we will proceed to design the network structure and train the model. Finally we will validate the model and test it. During this process TensorRT is not being used.

Phase 2:
Once we have the trained model, we will create and validate the deployment solution.
We have to determine what application we want to give to this neural network, taking into account the design and the implementation. Furthermore, we have to define what priorities we have, as well as different things we have to take into account, for example.

- Do we have a single network or more than one? For our case we will only have one network, face detection.
- If we are going to use a GPU or CPU or a mix.
- What latency and throughput we need as well as the system outputs we are going to have.

After you define the architecture of your inference solution, by which you determine what your priorities are, you then build an inference engine from the saved network using TensorRT.
Once the network is parsed, we have to consider the optimization parameters as workspace size, precision, batch size. These options are fit in the TensorRT build steps, where you build an optimized inference engine based on your network. In the figure 20 it can be observe the different steps:

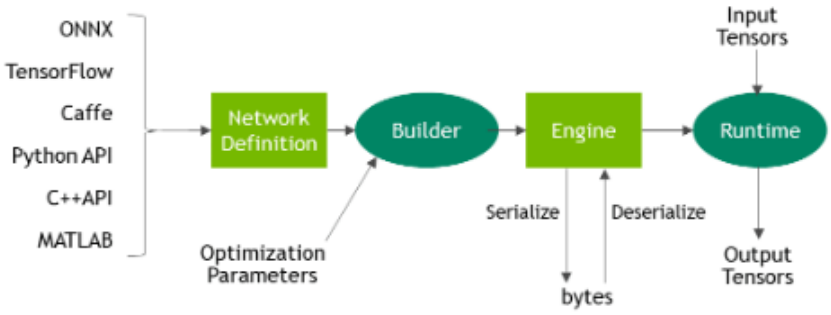


Figure 20: TensorRT diagram [18]

Finally, we have created an inference engine we will need to reproduce and validate the results of the model. And we have to write out the inference engine in a serialized format.

Phase 3:
The TensorRT library is linked to the deployment application; the application will first deserialize the model previously saved from the plan file into an inference engine.
Therefore, the capabilities provided by TensorRT are to adapt one or several previously designed and implemented models for different applications and optimise them depending on which parameters are most important to us, taking into account the system or hardware we want to use it on.

4 Results

In this section we will show the results obtained as well as the setup of our prototype. We will first discuss which model we have chosen for easy recognition, comparing the accuracy of each. And finally we will show the setup of our prototype.

4.1 Face detection model

As described above, for this project it has been decided to compare two models for face detection. On the one hand, the Haar Cascade Classifier has been studied and on the other hand, the FaceNet model described by Google engineers.

4.1.1 Haar Cascade Classifier

This classifier uses rectangular features to detect faces. It uses Adaboost training to choose which features are the best and discard those that are not necessary. It classifies them into weak classifiers and good classifiers. And finally, the cascade classifier consists of different stages that indicate whether the classifier has found a positive region or a negative region. To implement this model it is necessary to use OpenCV.

OpenCV is an open source library that is used in the aerial vision. This library is programmed in C++ and can be imported with the following command:

```
$ pip install opencv-python
```

Once the library is imported, to recognise the face we need to import the model. Intel provides us with a cascade file [16] This file consists of the following content:

- Weak classifiers containing different stages to create a more robust classifier. These internal nodes are used to create this model as reinforcement.
- The features necessary for face detection. These rectangles can be modified in the file and adjusted as desired.
- It contains the SVM parameters used to divide the information into positive or negative samples.

```
image = cv2.imread("person.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
face_detect = cv2.CascadeClassifier("cascade_face.xml")
faces = f_cascade.detectMultiScale(image, 1.3, 5)
```

Once we have passed the image through the Haar Cascade Classifier, the detectMultiScale `https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html` function will return the values of the rectangle containing the face or faces.

4.1.2 FaceNet

With the FaceNet model we generate a high quality mapping using two neural networks ZF-Net and Inception. We then use the Triplet loss method to be able to use these architectures. Reaching an accuracy of 98.87%. To use this model we need to install and set jetson-inference. This repository [17] provides us with the necessary libraries to use this pre-trained model, which is very intuitive to import. This inference library is intended to run on a Jetson platform. The previous section explains in detail how to download and import the necessary libraries to be able to use it.
The following will show how to import the model as well as the different values provided by the Detect function output.

```
net = jetson.inference.detectNet("facenet", threshold = 0.5)
detection = net.Detect(img , width, height)
```

The input parameters to the Detect method are the img, which in this case will be the live video with a resolution of 640x480.
Once the frame has been processed, it will return the different coordinates of the detected face forming a square. That is to say, it will return us a list with the parameters of the centre, the width and the height of the detected face.

4.1.3 Conclusion

After testing the two models, varying the camera position, the room light, the distance of the camera from the face, it was decided to use the Facenet model because it detected the face in situations that the Haar Cascade Classifier model did not. The Haar Cascade Classifier model did not detect the face when the camera was placed at an angle of 40° to the floor at a distance of 2m from the face while the Facenet model did detect the face.

4.2 Robot control

Once we have chosen the model to detect the face, it is time to manage how the robot should move. Roomba has to be all the time at a distance of approximately 2 meters either forward, backward, moving to the left or right.
Below we will list the requirements for the movement:

- If Roomba does not detect a face in the frame, it must start to rotate counterclockwise.
- If it detects a face in the frame and the distance is greater than 2 metres, it must pull forward. That is, when the Y position of the centre is greater than 270, we will use the command.fordward method to send it the command.
- If it detects a face in the frame and the distance is less than 2 metres, Y smaller than 210, we will use the command.backward method.

- If it detects a face in the frame and the centre X position is greater than 360, Roomba will rotate clockwise with the command.clockwise command.
- If it detects a face in the frame and the centre X position is less than 210, Roomba will start to rotate counterclockwise with the command.counterClkwise command.
- If it detects a face in the frame and none of the previous conditions is met it means that the face is in the center then the robot will stop moving.

In the following figure 21 we can see the final result of the prototype.

Figure 21: Prototype result

5 Budget

The following tables show the total cost of both hardware and time required for the development of the project.

5.1 Hardware cost

In this subsection are showed the price of the components used in the project.

Material	Units	Total price(€)
Nvidia Jetson Nano 4 GB	1	109
MicroSD 128 GB	1	20
Type C cable	1	8
Micro USB cable	1	5
Portable batery of 10000 mAh	1	20
Raspberry Pi Camera Module	1	38
iRobot Roomba 300		
Laptop	1	700

Table 5: Hardware cost

5.2 Cost hours worked

In this subsection are showed the price of the hours necessaries for develop, test and document the project.

Concept	hours(h)	Price(€)/hour(h)	Total price(€)
Preliminary study of the requirements	10	40	4000
System development	60	40	2000
Test	40	40	1600
Documentation	30	40	1200

Table 6: Cost total hours of the project

5.3 Total budget

The total price may vary depending on where the material is purchased, as well as the price per hour of the developer, which may vary.

Concept	Total price(€)
Hardware cost	1200
Software cost	0
Cost hours worked	7200
Total cost	8400

Table 7: Total cost of the project

6 Conclusions and future development:

6.1 Conclusions

In this work we have studied different techniques of automatic learning and deep learning to identify face in an image, a video or in real time. As we have presented above, we have studied two algorithms for face detection; on the one hand we found Haar cascade one of the most popular algorithms for object detection and on the other hand the FaceNet algorithm developed by Google Inc. After comparing the two models, we came to the conclusion that the algorithm developed by Google provides better results when detecting faces so it has been decided to use this one.
Once the model to be used in theory has been chosen, it is time to implement it in practice. For this, we will use a device developed by Nvidia, the Jetson Nano development module. It is a very useful tool to start learning about AI and robotics, with real-world projects with a large community behind them. That's why we decided to choose this device replacing the Raspberry PI 3 and the Inter NCS2. In this way with a single small, compact and powerful device we can get better computational results, which translates into better performance and power consumption. A developer from Nvidia, presents a guide to inference and realtim DNN vision library for the Jetson platform among others. The repository uses Nvidia TensorRT where we can find a variety of pre-trained models and applying them is very simple and intuitive. This repository provides all the necessary files to be able to use the FaceNet model. Once the model is imported and the face is successfully detected, it is time to control the robot.
To be able to communicate with the robot, we have used a serial port that depending on the coordinates of the center of the detected face, we will send commands to control the different motors and making it move forward, backward, rotate counterclockwise or clockwise.
Thanks to this work I have been introduced to the world of AI, allowing me to better understand the different development processes of a project related to machine learning. At the beginning it was a bit difficult because I had to learn many new concepts for me and I had to take different courses to learn about AI as well as about the Nvidia Jetson Nano device, a device I had never used before. After many tests, and installations of different frameworks I was able to successfully install the inference repository already thought to be used directly on the Jetson Nano device.

6.2 Future development

To improve the face detection algorithm, a future implementation of the project would be to use Pytorch or Tensorflow, these frameworks allow us to create and train models thanks to their APIs. Therefore, a future implementation would be to create and train an own model and not the use of a pre-trained model as it has been done in this work for it what is proposed is the following.

- First we collect our own data and generate a database to be able to train the desired model.
- Define the neural network

- Train and evaluate the model

Once these three tasks are done, we will have generated our data, implemented the desired model depending on the application we want to give it and finally train and evaluate it, being able to observe the outputs that the trained model will provide us. In this way, we can use the system not only to detect faces, but also to train it to detect objects, identify people, animals... In this way, we have a much greater control and allowing an instant interaction with the model and to be able to debug it in a simple way.

References

[1] Intel Software, Intel Neural Compute Stick 2 (Intel NCS2), (26 September 2021) https://software.intel.com/content/www/us/en/develop/hardware/neural-compute-stick.html

[2] F. Schroff, D. Kalenichenko, J.s Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015

[3] A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets - Scientific Figure on ResearchGate. (20 October 2021) https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-...convolutional-neural-network-CNN-architecture-26_fig1_336805909

[4] P. Viola, M.Jones. Rapid Object Detection using Boosted Cascade of Simple Feature. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001

[5] Haar Cascade Classifier, (26 September 2021) https://medium.datadriveninvestor.com/haar-cascade-classifiers-237c9193746b

[6] Nvidia Documentation, CUDA C++ Programming Guide, (24 September 2021) https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[7] Nvidia developers, CUDA, (25 September 2021) https://developer.nvidia.com/cuda

[8] Nvidia documentation, CUDA, (27 September 2021) https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

[9] Simulating active membrane systems using GPUs - Scientific Figure on ResearchGate, (26 September 2021) https://www.researchgate.net/figure/Thread-organization-in-CUDA-programming-...model_fig1_228987480

[10] Nvidia developers, Jetpack 4.6, (22 September 2021) https://developer.nvidia.com/embedded/jetpack

[11] Nvidia developers, Specifications Jetson Nvidia Nano, (27 September 2021) https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[12] Raspberry, Raspberry Accessories Documentation, (27 September 2021) https://www.raspberrypi.org/documentation/accessories/camera.html

[13] OpenHacks, Raspberry Pi Camera Module, (28 September 2021) https://www.openhacks.com/page/productos/id/840/title/Raspberry-Pi-Camera-Module#.YXA3Q9MzZQI

[14] iRobot, iRoboto Roomba 600 Open Interface (OI) Documentation, (10 August 2016)

https://www.irobotweb.com/~/media/MainSite/PDFs/About/STEM/Create/iRobot_Roomba_600_Open_Interface_Spec.pdf

[15] Nvidia Developers, NVIDIA TensorRT, (30 Semptember 2021)
https://developer.nvidia.com/tensorrt

[16] Intel Corporation, OpenCv, (2000), GitHub repository.
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

[17] Dustin Franklin, Jetson Inference, (2019), GitHub repository.
https://github.com/dusty-nv/jetson-inference

[18] Nvidia developers, Nvidia TensorRT documentation, (1 October 2021)
https://docs.nvidia.com/deeplearning/tensorrt/index.html

[19] Nvidia Developers, Release Notes v1.0, (17 October 2021)
https://developer.download.nvidia.com/embedded/L4T/r32-3-1_Release_v1.0/Jetson_Nano_Developer_Kit_User_Guide.pdf?WYw6hZZbZnJXw9yUnD3Q5G2Qhn-B-P1yCS54k-W9eupagu0nl6jprk6wjLtq7LtTx...7ZiJYAqh7gXVKStA0YL86_1Yyk2SRdXMzjGaNf_I8KvxlDwglzhw-sOp-IW5nplFy...WmD5w3T6aEE4tBPK5n5MwRcL70sg7bZOmENihU-MfKQ14iXHUC5PY

[20] Nvidia Developer, Nvidia forum, (20 October 2021)
https://forums.developer.nvidia.com/t/jetson-nano-physical-pinout-vs-gpio-list/123460

Appendices

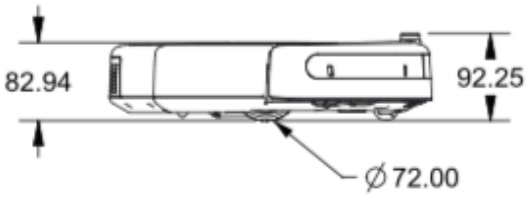


Figure 22: Right view Roomba[14]

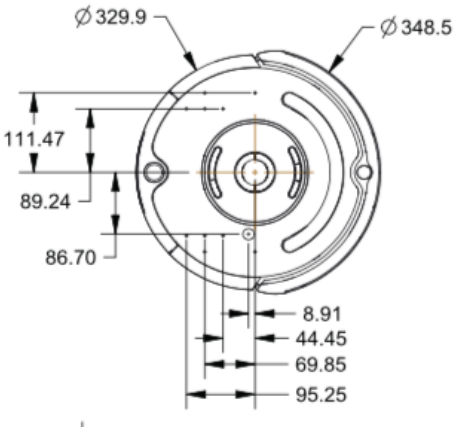


Figure 23: Top view Roomba [14]

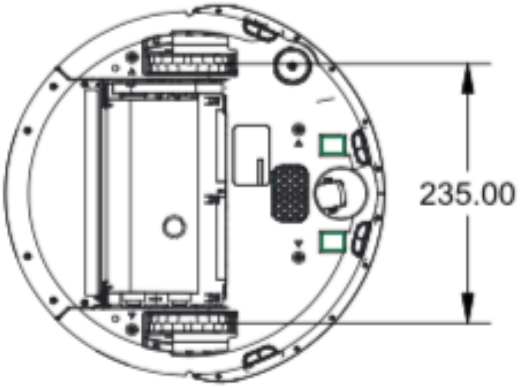


Figure 24: Bottom view Roomba [14]

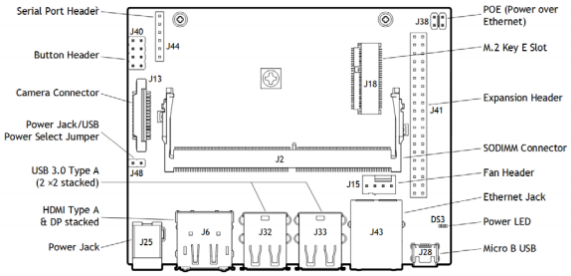


Figure 25: Top view Jetson Nano board [19]

Alt Function	Linux(BCM)	Board Label			Board Label	Linux(BCM)	Alt Function
DAP4_DOUT	78(21)	D21	40	39	GND		
DAP4_DIN	77(20)	D20	38	37	D26	12(26)	SPI2_MOSI
UART2_CTS	51(16)	D16	36	35	D19	76(19)	DAP4_FS
		GND	34	33	D13	38(13)	GPIO_PE6
LCD_BL_PWM	168(12)	D12	32	31	D6	200(6)	GPIO_PZ0
		GND	30	29	D5	149(5)	CAM_AF_EN
		D1/ID_SC	28	27	D0/ID_SD		
SPI1_CS1	20(7)	D7	26	25	GND		
SPI1_CS0	19(8)	D8	24	23	D11	18(11)	SPI1_SCK
SPI2_MISO	13(25)	D25	22	21	D9	17(9)	SPI1_MISO
		GND	20	19	D10	16(10)	SPI1_MOSI
SPI2_CS0	15(24)	D24	18	17	3.3V		
SPI2_CS1	232(23)	D23	16	15	D22	194(22)	LCD_TE
		GND	14	13	D27	14(27)	SPI2_SCK
DAP4_SCLK	79(18)	D18	12	11	D17	50(17)	UART2_RTS
		RXD/D15	10	9	GND		
		TXD/D14	8	7	D4	216(4)	AUDIO_MCLK
		GND	6	5	SCL/D3		
		5V	4	3	SDA/D2		
		5V	2	1	3.3V		

Figure 26: Pinout Jetson Nano board [20]