University of study of Trento

Department of Industrial Engineering

Master of Science in Mechatronics Engineering

Embedded Systems

Enhancing UAV capabilities with machine learning on board

Supervisor

Prof. Brunelli Davide

Candidate

Francesco Argentieri

ID 183892

Academic Year 2018/2019

Abstract

Contents

List of Figures

List of Tables

Chapter 1

Hardware

This chapter describes the hardware used to realize out the project. Where the use of the Raspberry Pi single board computer designed to have high performance both in education and science coupled with 8 mega-pixel camera module CMOS sensor was an almost conditioned choice. Shows that scientific and engineering–grade imagery can be produced with the Raspberry Pi 3 and its V2.1 camera module. It also presents a brief introduction to thermography and its physical principles before introducing Lepton 2.5 thermal camera module, made by the company FLIR which is currently one of the top manufactures of thermal camera solutions. The Coral Dev Board is a single-board computer that contains an Edge TPU coprocessor. It's ideal for prototyping new projects that demand fast on-device inferencing for machine learning models. The Coral Dev Board is ideal when you need to perform fast machine learning (ML) inferencing in a small form factor.

1.1 Raspberry Pi 3

The Raspberry Pi is a high-performance single-board computer designed to experiment and solve real-world problems. This small computer supports a camera module that uses a Sony IMX219 8 mega-pixel CMOS sensor.

1.1.1 Specification

As of early 2016, over 8 million Raspberry Pi's had been sold, making it one of the most popular single-board computers on the market.[1]
The Raspberry Pi credit-card-sized computer supports several accessories, including a camera module containing the Sony IMX219 sensor. This computer and camera configuration is of particular interest since it can provide raw-data format imagery that can

be used for a multitude of applications, including computer vision, biophotonics, medical testing, remote sensing, astronomy, improved image quality, high dynamic range (HDR) imaging, and security monitoring. The Raspberry Pi 3 is the third generation single board Raspberry Pi computer and became available to consumers in February 2016.
Some of the more significant Raspberry Pi attributes, including interfaces, are described in Table 1.1. The Raspberry Pi Foundation provides several operating systems for the Raspberry Pi 3, including Raspbian and a Debian-based Linux distribution, as well as third-party Ubuntu, Windows 10 IOT Core, RISC OS, and specialized distributions for download.[2]

Table 1.1: Raspberry Pi 3 Specification.

CPU 1.2 GHz 64-bit ARM Cortex-A53
1 GB of RAM LPDDR2 (900 MHz)
Wireless N and Blue-tooth 4.1 communication
Four USB ports
HDMI interface
Ethernet port
MicroSD card slot
40 GPIO pins
Camera interface
Composite video audio jack

1.2 Raspberry Pi, Camera Board V2

The camera is based on the Sony IMX219 silicon CMOS back-lit sensor and produces 8 mega-pixel images that are 3280 × 2464 pixels in size.
The IMX219 sensor operates in the visible spectral range from 400 to 700 nm).[3]
Sensor specifications are detailed in Table 1.2.

1.2.1 Specification

The V2 camera module operates at a fixed focal length (3.04 mm) and single f-number (F2.0) typically focused from the near-field to infinity. Images can be captured at ISO settings between 100 and 800 in manually set increments of 100 and camera exposure times between 9 µs and 6 s using a rolling shutter. Some of the more significant camera specifications are shown in Table 1.3. In addition to still photos, the Raspberry Pi Sony IMX219 sensor supports a cropped 1080p format at 30 frames per second (fps) and full-frame imaging video at up to 15 fps, but not in raw-data format. The entire camera board is small 25 mm × 25 mm × 9 mm and weighing about 3 g. It connects directly to

Table 1.2: Sony IMX219 sensor chip specifications.

Sensor parameter	**Specification**
Image sensor type	Back-lit CMOS
Image size	Diagonal 4.60 mm (type 1/4.0)
Number of active pixels	3280 (H) × 2464 (V) ∼ 8.08 mega-pixels
Chip size	5.095 mm (H) × 4.930 mm(V) (w/ Scribe)
Unit cell size (pixel)	1.12 µm (H) × 1.12 µm(V)
Substrate material	Silicon
Bit depth	10-bit A/D converter on chip
Data output	CSI2 serial data output (selection of 4 lane/ 2 lane)
Communication	2-wire serial communication circuit on chip
Max full-frame frame rate	30 frames/s
Pixel rate	280 mega-pixel/s (all-pixels mode)
Data rate	Max. 755 Mbps/lane (at 4 lane), 912 Mbps / lane(at 2 lane)

the Raspberry Pi 3 through a 15 pin mobile industry processor interface (MIPI) camera serial interface and is shown alongside a Raspberry Pi 3 in Figure 1.1.[1, 4]

Table 1.3: Raspberry Pi camera Specification.

Camera parameter	**Specification**
Lens focal length	3.04 mm
f-number	2.0
Instantaneous field of view	0.368 mrad
Full-frame field of view	59.17 °(H) × 58.3 ° (V)

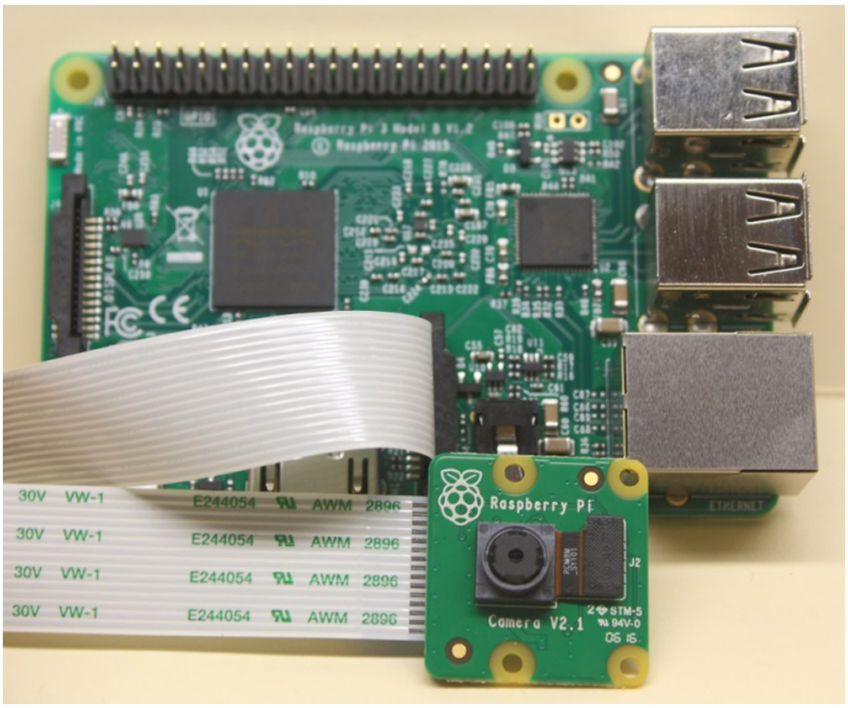


Figure 1.1: Raspberry Pi 3 and camera module V2.1.

1.3 Theory

Electromagnetic radiation is all around (and within, and throughout) us and is comprised of everything from gamma radiation on the high frequency end to radio waves on the low frequency end. While most imaging sensors detect radiation in the visible spectrum (wavelengths from 380 to 700 nm), long wave infra-red sensors detect radiation from 900 to 14000 nm. This is known as the infra-red spectrum, and it accounts for most of the thermal radiation emitted by objects near room temperature.

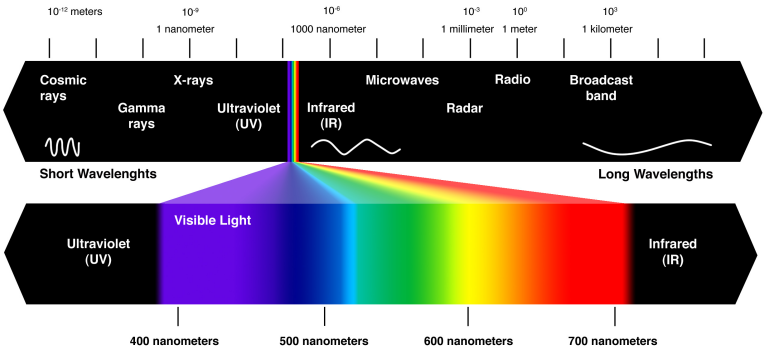


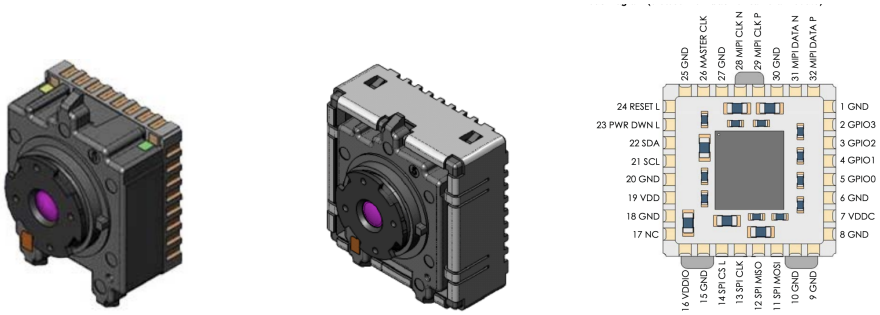
Figure 1.2: Composition of the spectrum of electromagnetic waves.
Source: https://socratic.org

1.4 Thermal Camera module Lepton 2.5

The thermal camera module we use in this project its made by FLIR, we will go through its technical specification, modes of capture and communication protocol for both video frame transfer and camera control. Information about the camera are extracted from official documentation.

1.4.1 Specification

Lepton is an infra-red camera system that integrates a fixed-focus lens assembly, an 80×60 long-wave infra-red (LWIR) micro-bolometer sensor array, and signal-processing electronics. Easy to integrate and operate, Lepton is intended for mobile devices as well as any application requiring very small footprint, very low power, and instant-on operation. Lepton can be operated in its default mode or configured into other modes through a command and control interface (CCI). The effective frame rate of the camera is only 8.6 Hz, however for our needs this is not a problem. The camera only requires low voltage supply and has small power consumption of around 140 mW.



(a) *with and without socket.* **(b)** *Pinout Diagram.*

Figure 1.3: Lepton Camera sketch.
Source: [5]

See table 1.4 for more specifications. For better manipulation with the camera module we use a breakout board (figure 1.4) with a housing for the Lepton camera module. The breakout board provides better physical accessibility, improves heat dissipation and increases input voltage supply range to 3–5 V, as it has its own regulated power supply. This power supply provides the camera module with three necessary voltages: 1.2, 2.8 and 2.5–3.1 V. The breakout board also supplies the camera with master clock signal.[6] The camera uses two interfaces for communication:

- SPI for transferring video frames from the camera to a SPI master device.
- I^2C for receiving control commands from the I^2C master device.

Table 1.4: Thermal camera specification

	FLIR Lepton 2.5	
Resolution (h x w)	80 × 60	pixels
Spectral Range	8 to 14	µm
Horizontal Field of View	51	°
Thermal Sensitivity	< 50	mK
Frame Rate	< 9	Hz
Control Interface	I^2C	
Video Interface	SPI	
Promised Time to Image	< 0.5	s
Integral Shutter	yes	
Radiometry	14-bit pixel value	
Operating Power	∼150	mW

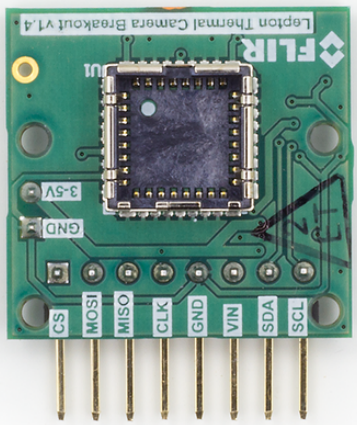


Figure 1.4: Breakout board.
Source: https://groupgets.com/manufacturers/getlab/products/flir-lepton-breakout-board-v1-4

1.4.2 System Architecture

The lens assembly focuses infrared radiation from the scene onto an 80 × 60 array of thermal detectors with 17-micron pitch. Each detector element is a vanadium-oxide (VOx) microbolometer whose temperature fluctuates in response to incident flux. The change in temperature causes a proportional change in each microbolometer's resistance. VOx provides a high temperature coefficient of resistance (TCR) and low $1/f$ noise, resulting in excellent thermal sensitivity and stable uniformity. The microbolometer array is grown monolithically on top of a readout integrated circuit (ROIC) to comprise

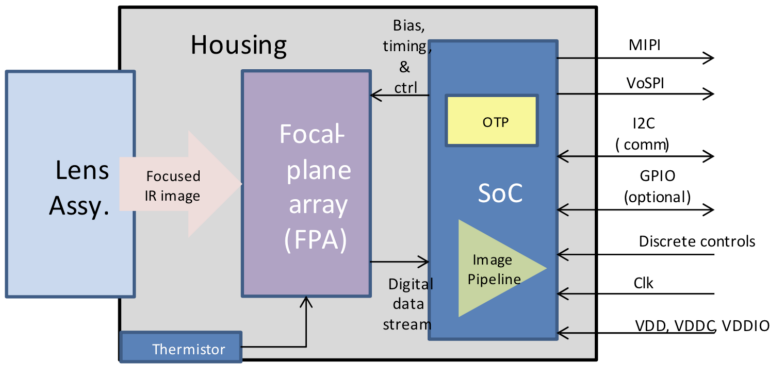


Figure 1.5: Lepton Architecture.
Source: [5]

the complete focal plane array (FPA). Once per frame, the ROIC senses the resistance of each detector by applying a bias voltage and integrating the resulting current for a finite period of time called the integration period. The serial stream from the FPA is received by a system on a chip (SoC) device, which provides signal processing and output formatting.

1.4.3 Video Pipeline

The video pipeline includes non-uniformity correction (NUC), defect replacement, spatial and temporal filtering, automatic gain correction (AGC), and colourization.

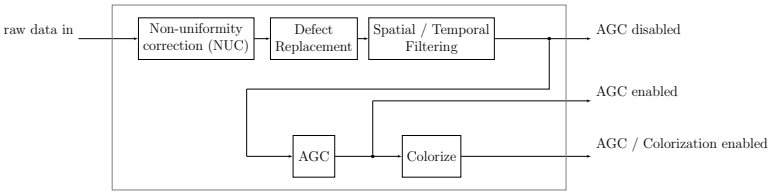


Figure 1.6: Lepton video pipeline block diagram.
Source: [5]

The non-uniformity correction (NUC) block applies correction terms to ensure that the camera produces a uniform output for each pixel when imaging a uniform thermal scene. Factory-calibrated terms are applied to compensate for temperature effects, pixel response variations, and lens-illumination roll-off. To compensate for temporal drift, the NUC block also applies an offset term that can be periodically updated at runtime via a process called flat-field correction (FFC). The FFC process is further described in FFC States, (1.4.5).

The defect-replacement block substitutes for any pixels identified as defective during factory calibration or during runtime. The replacement algorithm assesses the values of neighboring pixels and calculates an optimum replacement value. The typical number of defective pixels is ≤ 1.

Temporal Filtering the image pipeline includes a number of sophisticated image filters designed to enhance signal-to-noise ratio (SNR) by eliminating temporal noise and residual non-uniformity. The filtering suite includes a scene-based non-uniformity correction (SBNUC) algorithm which relies on motion within the scene to isolate fixed pattern noise (FPN) from image content.

The AGC algorithm for converting the full-resolution (14-bit) thermal image into a contrast-enhanced image suitable for display is a histogram-based non-linear mapping function. See (1.4.6).

The colorize block takes the contrast-enhanced thermal image as input and generates a 24-bit RGB color output.

1.4.4 Power States

Lepton currently provides five power states. As depicted in the state diagram shown in Figure 6, most of the transitions among the power states are the result of explicit action from the host. The automatic transition to and from the overtemp state is an exception. In the figure, transitions that require specific host-side action are shown in bold. Automatic transitions are not bolded.

- Off: When no voltage is applied, Lepton is in the off state. In the off state, no camera functions are available.
- Uninitialized: In the uninitialized state, all voltage forms are applied, but Lepton has not yet been booted and is in an indeterminate state. It is not recommended to leave Lepton in this state as power is not optimized; it should instead be booted to the on-state (and then transitioned back to standby if imaging is not required).
- On: In the on state, all functions and interfaces are fully available.
- Standby: In the standby state, all voltage forms are applied, but power consumption is approximately 4 mW. In the standby state, no functions are available, but it is possible to transition to the on state via the start-up sequence. The shutdown sequence is the recommended transition back to the standby state. It is also possible

to transition between standby and on states via software commands, as further defined in the software IDD.

- Overtemp: The overtemp state is automatically entered when the Lepton senses that its temperature has exceeded approximately 80 °C. Upon entering the overtemp state, Lepton enables a "*shutdown imminent*" status bit in the telemetry line and starts a 10 s counter. If the temperature of the Lepton falls below 80 °C before the counter times out, the "*shutdown imminent*" bit is cleared and the system transitions back to the on state. If the counter does time out, Lepton automatically transitions to the standby state.

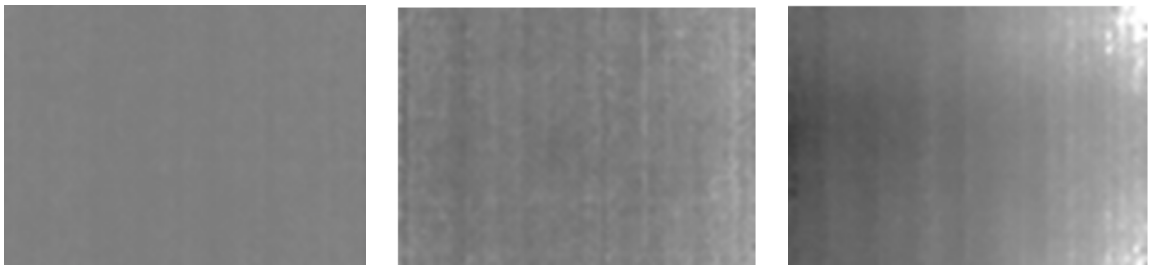
1.4.5 FFC States

Lepton is factory calibrated to produce an output image that is highly uniform, such as shown in Figure 1.7a, when viewing a uniform-temperature scene. However, drift effects over long periods of time degrade uniformity, resulting in imagery which appears more grainy (Figure 1.7b) and/or blotchy (Figure 1.7c). Operation over a wide temperature range (for example, powering on at −10 °C and heating to 65 °C) will also have a detrimental effect on image quality. For scenarios in which there is ample scene movement, such as most handheld applications, Lepton is capable of automatically compensating for drift effects using an internal algorithm called scene-based non-uniformity correction (scene-based NUC or SBNUC). However, for use cases in which the scene is essentially stationary, such as fixed-mount applications, scene-based NUC is less effective. In those applications, it is recommended to periodically perform a flat-field correction (FFC). FFC is a process whereby the NUC terms applied by the camera's signal processing engine are automatically recalibrated to produce the most optimal image quality. The sensor is briefly exposed to a uniform thermal scene, and the camera updates the NUC terms to ensure uniform output. The entire FFC process takes less than a second.
The current FFC state is provided through the telemetry line. There are three FFC states, as illustrated in Figure 1.8:

1. FFC not commanded (default): In this state, Lepton applies by default a set of factory-generated FFC terms.

2. FFC in progress: Lepton enters this state when FFC is commanded. The default FFC duration is nominally 23 frames.

3. FFC complete: Lepton automatically enters this state whenever FFC is completed. Lepton also provides an "FFC desired" flag in the telemetry line. The "FFC desired" flag is asserted at start-up, when a specified period (default = 3 minutes) has elapsed



(a) *Highly uniform image.* **(b)** *Grainy image (high-spatial frequency noise).* **(c)** *Blotchy image (low-spatial frequency noise).*

Figure 1.7: Examples of Good Uniformity, Graininess, and Blotchiness
Source: [5]

since the last FFC, or when the sensor temperature has changed by a specified value (default = 3 °C) since the last FFC. The "FFC desired" flag is intended to indicate to the host to command an FFC at the next possible opportunity.

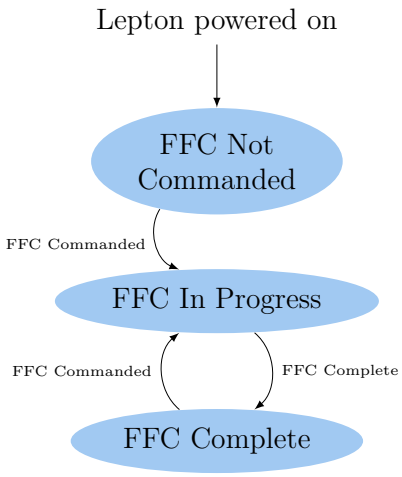


Figure 1.8: FFC States.

1.4.6 AGC Modes

There are two AGC modes:

- AGC disabled (default)
- AGC enabled

AGC is a process whereby the large dynamic range of the infrared sensor is collapsed to a range more appropriate for a display system. For Lepton, this is a 14-bit to 8-bit conversion. In its most simplistic form, AGC can be a linear mapping from 14-bit

to 8-bit; however, a simple linear AGC is generally incapable of providing pleasing imagery in all imaging conditions. For example, when a scene includes both cold and hot regions (for example, a hot object in front of a cold background as illustrated in 1.10, linear AGC can produce an output image in which most pixels are mapped to either full black or full white with very little use of the gray shades (8-bit values) in between. Because of this limitation of linear AGC, a more sophisticated algorithm is preferred. Similar to most AGC algorithms that optimize the use of gray shades, Lepton's is histogram-based. Essentially a histogram counts the number of pixels in each frame that have a given 14-bit value. Figure 1.9 the concept for a 3x3 pixel area.

Classic histogram equalization uses the cumulative histogram as a mapping function

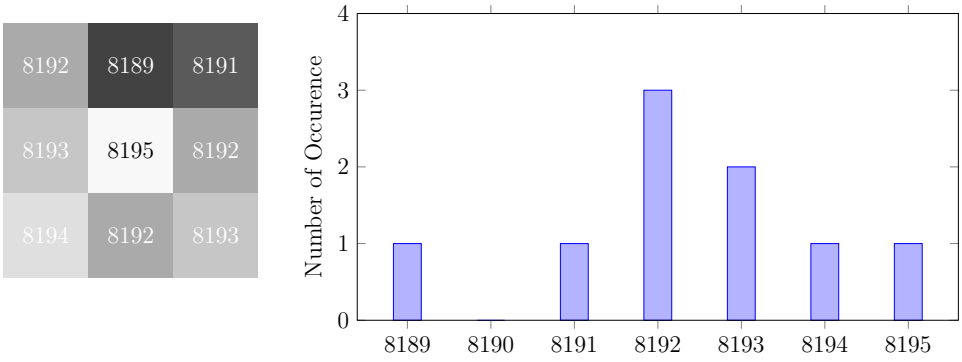


Figure 1.9: Illustration of a Histogram for a 3×3 Pixel Area.
Source: [5]

between 14-bit and 8-bit. The intent is to devote the most gray shades to those portions of the input range occupied by the most pixels. For example, an image consisting of 60% sky devotes 60% of the available gray shades to the sky, leaving only 40% for the remainder of the image. By comparison, linear AGC "wastes" gray shades when there are gaps in the histogram, whereas classic histogram equalization allocates no gray shades to the gaps. This behavior is in principle an efficient use of the available gray shades, but there are a few drawbacks:

- The resulting contrast between an object and a much colder (or hotter) background can be rendered poor by the fact the algorithm "collapses" the separation between such that the object is only one step gray shade above the background. This phenomenon is illustrated in 1.10.

- Too much emphasis can be placed on background clutter, particularly when a mostly isothermal background comprises a large fraction of the total image area. This is also illustrated in 1.10. The Lepton AGC algorithm is a modified version of classic histogram equalization that mitigates these shortcomings. One such modification

is a parameter called “clip limit high”. It clips the maximum population of any single bin, limiting the influence of heavily populated bins on the mapping function. Another parameter utilized by the Lepton algorithm is called “clip limit low”. It adds a constant value to every non-zero bin in the histogram, resulting in additional contrast between portions of the histogram separated by gaps. Figure 1.10 is an example showing the benefit of the Lepton clip parameters.

(a) *Linear AGC.*

(b) *Classic Histogram Equalization.*

(c) *Lepton's Variant of HistogramEqualization.*

Figure 1.10: Comparison of Linear AGC and Classic–Lepton variant of Histogram Equalization
Source: [5]

A high value of clip limit high results in a mapping more like classic histogram equalization, whereas a low value results in mapping more like linear AGC. For clip limit low, the opposite is true: a high value results in a mapping more like linear AGC, whereas a low value results in a mapping more like classic histogram equalization. The default values of both parameters produce a good compromise between the two; however, because optimum AGC is highly subjective and often application dependent, customers are encouraged to experiment to find settings most appropriate for the target application. By default, the histogram used to generate Lepton's 14-bit to 8-bit mapping function is collected from the full array. In some applications, it is desirable to have the AGC algorithm ignore a portion of the scene when collecting the histogram. For example, in some applications it may be beneficial to optimize the display to a region of interest (ROI) in the central portion of the image. When the AGC ROI is set to a subset of the full image, any scene content located outside of the ROI is not included in the histogram and therefore does not affect the mapping function (note: this does not mean the portion outside of the ROI is not displayed or that AGC is not applied there, only that those portions outside the AGC ROI do not influence the mapping function).[5]

1.5 Interface

The Raspberry Pi seen in section (1.1) has bi-directional I/O pins, which you can use to drive LEDs, spin motors, or read button presses. The board offers its GPIO over a standard male header on the board. Over the years the header has expanded from 26 pins to 40 pins while maintaining the original pinout. There are at least two, different numbering schemes you may encounter when referencing pin numbers:

1. Broadcom chip-specific pin numbers
2. `P1` physical pin numbers.

Here's a figure 1.11 showing all 26 pins on the `P1` header, including any special function they may have, and their dual numbers.

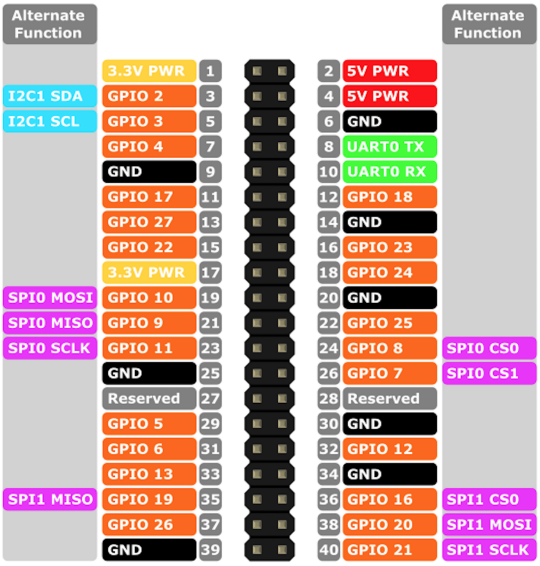


Figure 1.11: Raspberry Pi 3 Pin Mappings.
Source: Microsoft Windows Dev Center

The camera uses two interfaces for communication:

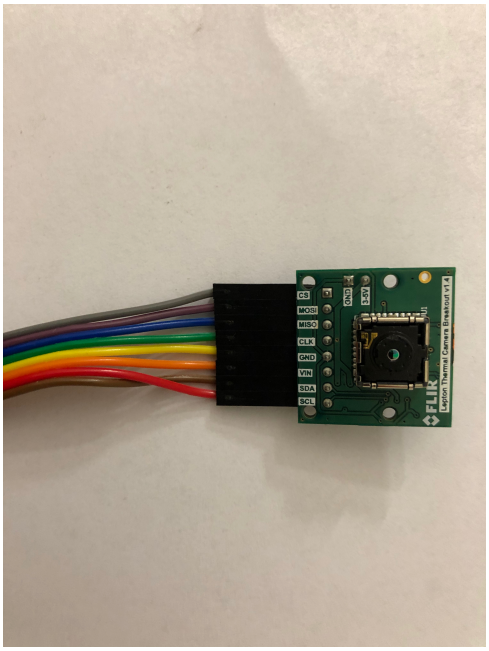
- **SPI** for transferring video frames from the camera to a SPI[1] master device.
- **I^{2}C** for receiving control commands from the I^{2}C[2] master device.

The Raspberry Pi's CPU has enough processing power to maintain smooth operation without delays, which turned out to be crucial for maintaining synchronization with the camera module when transferring video frames. Below is reported the connection scheme used between the FLIR breakout and the Raspberry Pi 's GPIO according to the figures (1.12) and the table (1.5).

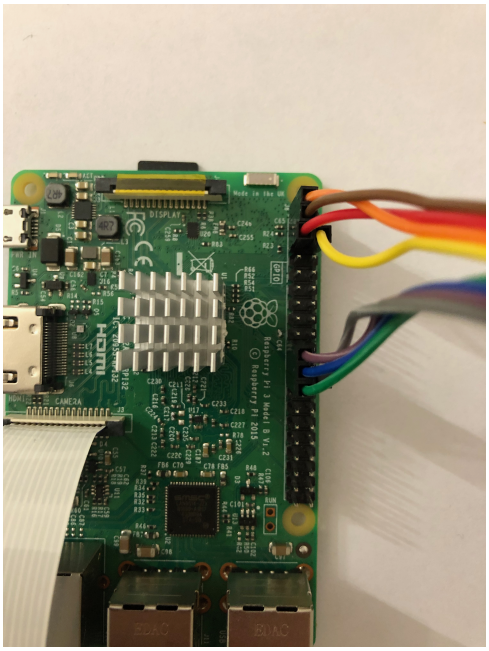
[1]SPI – Serial peripheral interface bus
[2]I2C (Inter-integrated circuit)

Raspberry GPIO	Breakout board	Alternative function	PIN
+3.3V	VIN	SPI0	1
SDA	SDA	SPI0	3
SCL	SCL	SPI0	5
GND	GND	SPI0	6
MOSI	MOSI	GND	19
MISO	MISO	3.3V	21
SCLK	CLK	I2C1	23
CE0	CS	I2C1	24

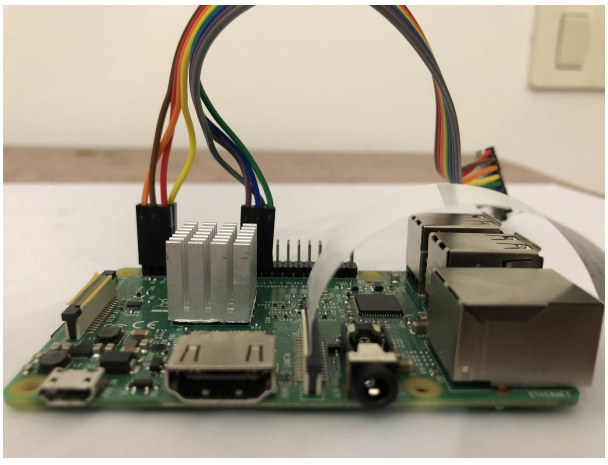
Table 1.5: Schematic connection GPIO



(a) *Breakout board detail.*



(b) *Top view.*



(c) *Frontal view.*

Figure 1.12: Realization of the connection between the Raspberry Pi 3 GPIO and Break out board

1.6 Coral Dev-Board

Nowadays some manufacturers replace the GPU for a TPU, a Tensor Processing Unit. Driven by the esponetial interest for deep learning in different field such as reserch, industries, robotics. The Coral Dev Board is a single-board computer that contains an Edge TPU coprocessor. It's ideal for prototyping new projects that demand fast on-device inferencing for machine learning models. Coral Dev Board cannot train a neural network because the TPU is specific designed to work with special pre-compiled model and to be small and energy efficient.

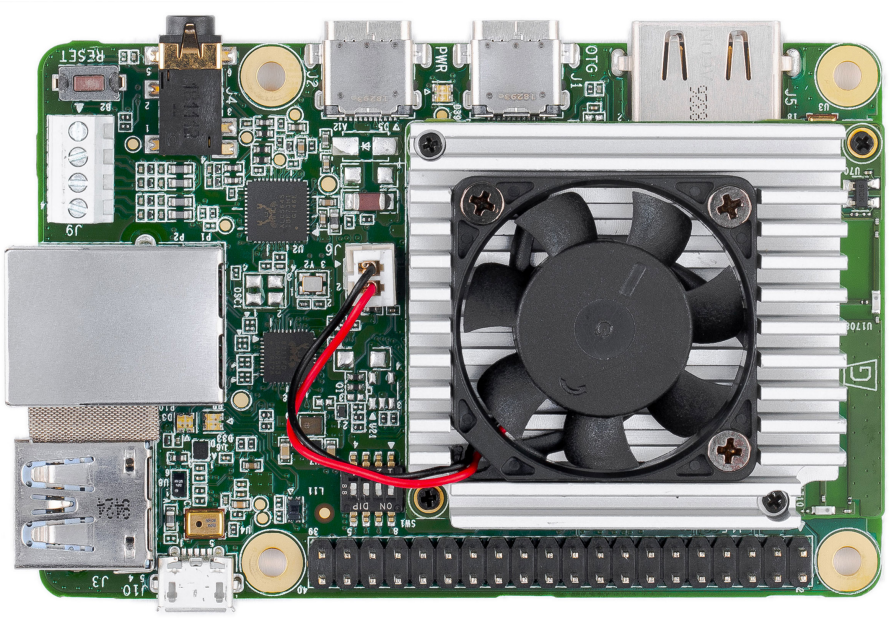


Figure 1.13: Coral Dev Board.
Source: https://coral.ai/docs/dev-board/get-started/

1.6.1 Overview

The Coral Dev Board is a single-board computer that's ideal when you need to perform fast machine learning (ML) inferencing in a small form factor. You can use the Dev Board to prototype your embedded system and then scale to production using the on-board Coral System-on-Module (SoM) combined with your custom PCB hardware. The SoM provides a fully-integrated system, including NXP's iMX 8M system-on-chip (SoC), eMMC memory, LPDDR4 RAM, Wi-Fi, and Bluetooth, but its unique power comes from Google's Edge TPU coprocessor. The Edge TPU is a small ASIC designed by Google that provides high performance ML inferencing with a low power cost. For example, it can execute state-of-the-art mobile vision models such as MobileNet v2 at almost 400 FPS, in a power efficient manner. The baseboard provides all the peripheral connections you

need to prototype a project, including USB 2.0/3.0 ports, DSI display interface, CSI-2 camera interface, Ethernet port, speaker terminals, and a 40-pin I/O header.
Key benefits of the Dev Board:

- High-speed and low-power ML inferencing (4 TOPS @ 2 W)
- A complete Linux system (running Mendel, a Debian derivative)
- Prototyping and evaluation board for the small Coral SoM (40 x 48 mm)

Edge TPU System-on-Module (SoM)
NXP i.MX 8M SoC (Quad-core Arm Cortex-A53, plus Cortex-M4F)
Google Edge TPU ML accelerator coprocessor
Cryptographic coprocessor
Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5 GHz)
Bluetooth 4.2
8 GB eMMC
1 GB LPDDR4
USB connections
USB Type-C power port (5 V DC)
USB 3.0 Type-C OTG port
USB 3.0 Type-A host port
USB 2.0 Micro-B serial console port
Audio connections
3.5 mm audio jack (CTIA compliant)
Digital PDM microphone (x2)
2.54 mm 4-pin terminal for stereo speakers
Video connections
HDMI 2.0a (full size)
39-pin FFC connector for MIPI DSI display (4-lane)
24-pin FFC connector for MIPI CSI-2 camera (4-lane)
MicroSD card slot
Gigabit Ethernet port
40-pin GPIO expansion header
Supports Mendel Linux (derivative of Debian)

Table 1.6: Coral Dev Board Features.

The Google Coral with a special TPU chip performing all tensor calculations works with special pre-compiled TensorFlow Lite networks. If the topology of the neural network and its required operations can be described in TensorFlow it may work well on the Google Coral.

1.6.2 8 bits integers

An alternative to reduce compute time is the use of integers 8-bit instead float 32-bit. The difference in the use of amount of memory allocated is reduced. The advantage derive from the Neural Network 's accuracy insensibility of number represented, while maintaining the same precision. On the basis of this we can deduce that will save memory, furthermore an cut-off of number of transistor in the chip because is not more necessary take into account floating point. Consequently the this permit to achieve processor powerful, small and energy efficient.

1.7 TPU explained in depth

The Google Coral has a TPU on board which speeds up the tensor calculations enormously. These tensor calculations are used in deep learning and neural networks.
The Google Coral Dev board use an ASIC made by the Google team called the Edge TPU. It is a much lighter version of the well-known TPUs used in Google's datacenter. It also consumes very little power, so it is ideal for small embedded systems. Nevertheless, the similarities in applied technology are significant.[7]

1.7.1 Neural Node

Neural networks used in deep learning consists of many neural nodes. They are all connected together in a defined way. The way these nodes are wired is called the topology of the network. This topology determines the function the network performs. See this list for a selection of several types of deep learning networks. Each node has always three basic components. A multiplier multiplies all the inputs with their respective so-called weight, the synapses. An adder who accumulates all the individual multiplications. And an activation function that shapes the output given the addition. A schematic view below (1.14).

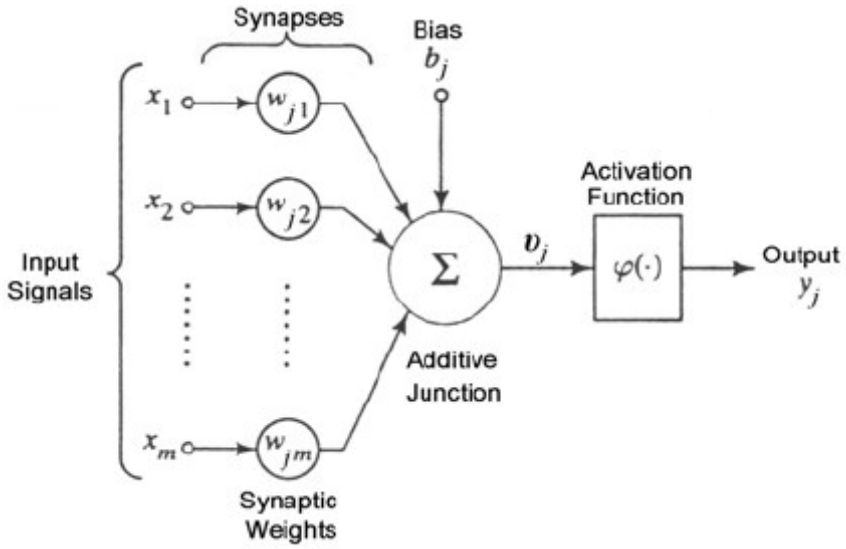


Figure 1.14: Artificial Neuron models and its parts.
Source: Adapted from Haykin (1994)

The formula can be written as follows (1.1).

$$y_i = \phi \left(\sum_{i=1}^{n} w_{ij} \cdot x_i \right) \tag{1.1}$$

Deep neural network is consist of millions of neural nodes distribute over many layers; despite the simplicity of the operation, only on multiplication and one addition, require long compute time to obtain a result. keep in mind that for training a network requires many epochs. In other words, the main problem is time. The algorithms is well suited for parallel execution. Although distributing the algorithm over the processes and threads can be an option actually provides disappoint results for the presence of bottlenecks due to architecture general purpose. On the other hand use of Graphical Processing Unit, the GPU on video card designed for efficient manipulation memory. Their highly parallel structure make them more efficient respect CPU especially for algorithms that process large blocks of data in parallel. The best choice is the use of the Tensor Processing Unit, the TPU. This device has been specially designed for the above neural node algorithm.

1.7.2 The adder

Generally a strategy adopted when software is tool slow is modify the hardware to reach the better achievements. The structure inside TPU is realized by three main components derived from neural node, then keeping in mind the scheme in figure (1.14): the **multiplier**, the **adder** and the **activation function** must be included in the hardware. Start with analysing the diagram of the 4-bit adder realized in figure (1.15). Where: `A` and `B` are the

inputs. If the output overflows `C4` the carry out is set. `C0` is the carry-in of a previous phase.[7] Signals `A` and `B` propagate through the circuit and generate the result `A + B`.

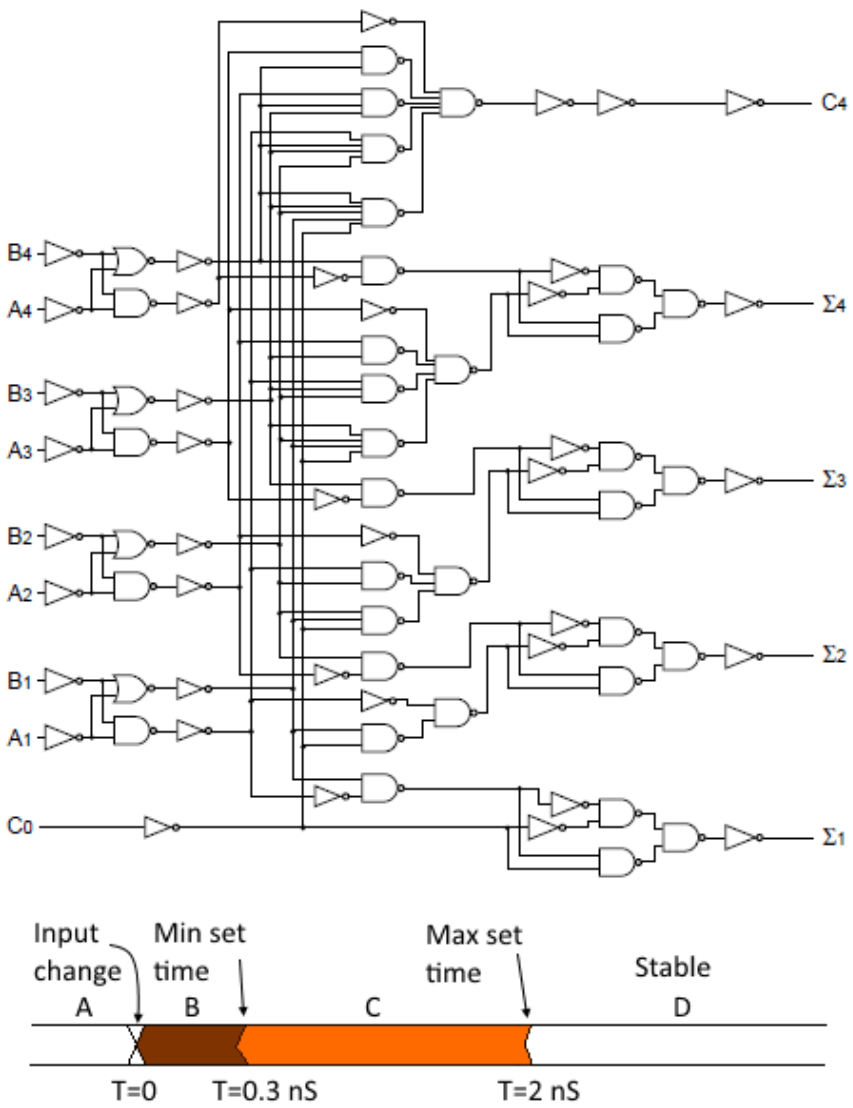


Figure 1.15: 4-bit adder
Source: Q-engineering

Changing one of them alters almost immediately the output. This happens extremely very fast, within a few nanoseconds. This propagation time depends on the number of digital ports whose output changes. The propagation time is therefore not fixed, but lies between two limits, a minimum and a maximum time, see diagram at the bottom of the drawing (1.15). By the way, al mentioned times are illustrative and have no relationship to any device.[7]

1.7.3 Pipeline

Consider the example where: the propagation time of one adder is 2 ns, therefore the maximum clock rate can be 500 MHz. Taking into account that a neural node can have many inputs, also hundreds, that must be accumulated this affect the propagation time that increase dramatically. This implies that the last adder in the chain must be attend all intermediate result before its output becomes stable. If we consider a chain of 250 input each of one with 2 ns delay, we obtain total time of 500 ns waiting.
Consequently clock results extremely slow 2 MHz, then solution adopted is structured pipeline where between each adder is inserted a memory element that keep keeps the result stable for the next adder. As show in figure (1.16).

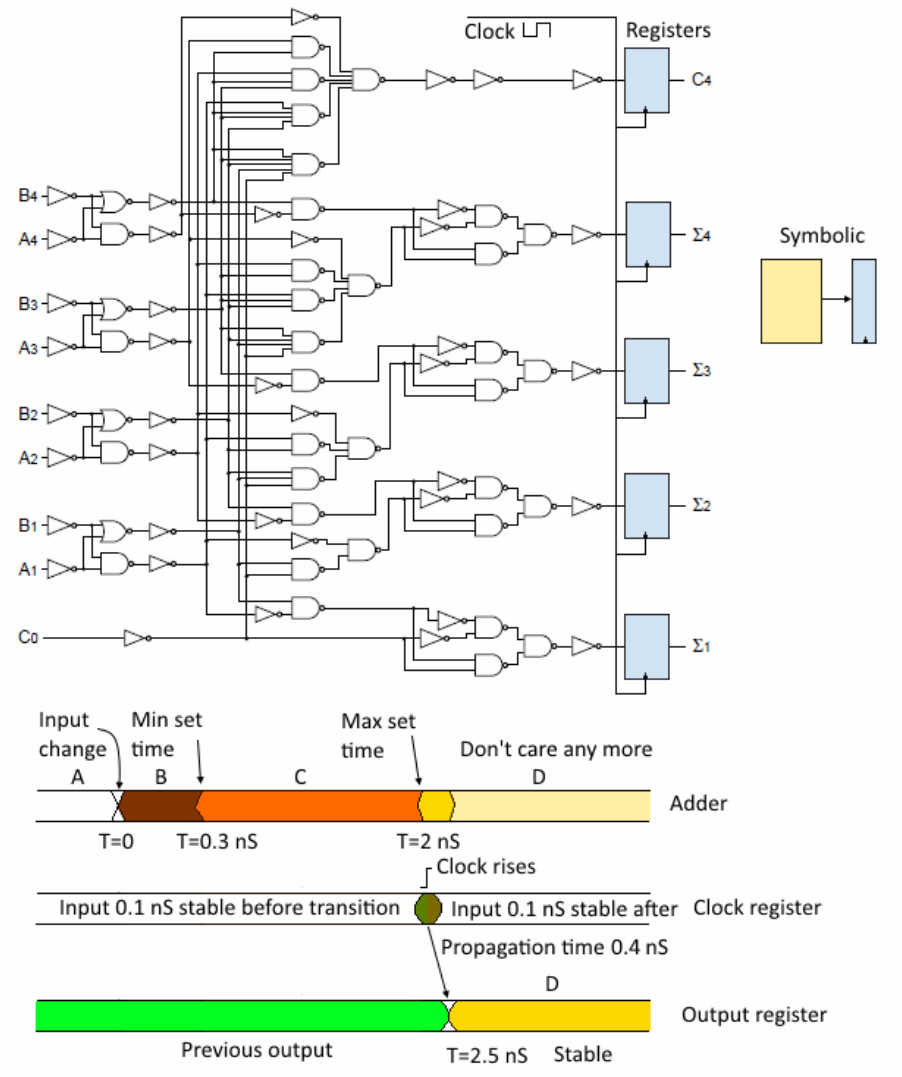
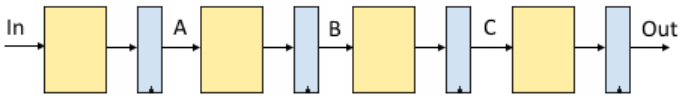


Figure 1.16: 4 bit adder with register
Source: Q-engineering

The output of the registers is updated at the rising edge of the clock signal. That is the only time the output can change. When the clock signal is high or low, the inputs cannot manipulate the output, then it remains stable. Just before the clock rises, the input must be stable for a minimum time, also just after the rise time. In contrast to the previous example, now consider a delay of 0.1 ns and adding the register's propagation time of (0.4 ns), the total propagation reach for a new stable signal is now 2.5ns, which results increasing up to clock speed of 400 MHz. Observable in figure (1.17). Taking into



Clk	In	A	B	C	Out
⌐¬					
0		?	?	?	?
1			?	?	?
2				?	?
3					?
4					
5					
6					

Figure 1.17: Digital pipeline.
Source: Q-engineering

account the table in figure 1.17, each color represents a value. After four clock cycles, the value at the input has propagated through the network and appears at the output. Because the registers are updated simultaneously a new input is accepted every 2.5 ns. The propagation time has been restored. The time it takes to travel through the whole pipeline is called latency time and is 10 ns in the above diagram (4×2.5 ns). Every digital component is built on large pipelines which guarantee the required speed.[7]

1.7.4 The mul-add cell

The input signal of every neural node has effect on final result. This gives a simple multiplication for an input mediate by a weight value. The operation of multiplication can be implemented with same logic of an adder, then are necessary more gates. Remembering that the neural node has multiplied his value for weight value. Thus the representation assume the scheme in figure (1.18).

Are necessary many important clarifications about the input registers `X1` and `X2` displayed in figure (1.19). They generate a delay line in the *mul-add* chain, thus can permit to correct synchronize. Considering the second cell *mul-add* who sum the output from

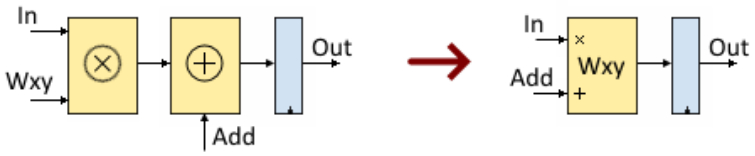


Figure 1.18: Mul-Add register.
Source: Q-engineering

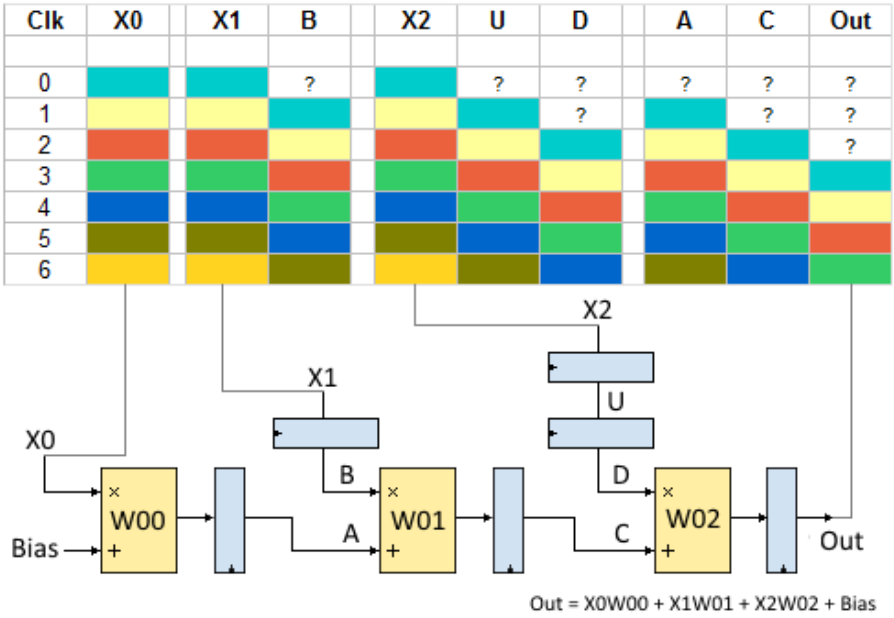


Figure 1.19: Schematic for a three input neuron.
Source: Q-engineering

previous cell (signal `A`) retarded by one clock cycle. Consequently also the multiplication between the value in register `X1` and weight value must be delayed the same amount of time. The schema reported in figure (1.19) shows for each colour that represent the vector of input $[X0, X1, X2]$ while the input `A` and `B`, represented by lines straight, maintains always same colour. This means that they appear simultaneous, i.e. are synchronised. The same must be valid for input `C` and `D`.

1.7.5 Systolic array

Starting from structure, just examined, is quite easy extend it to other neurons, taking into account that every single input is connected with all neurons in the next layer, as shows in figure (1.20). The adoption of this solution, called systolic array, permits to have an increment of speed in parallel calculation. Since the propagation time remains unchanged for an spread of systolic array in depth or width, while only the latency time

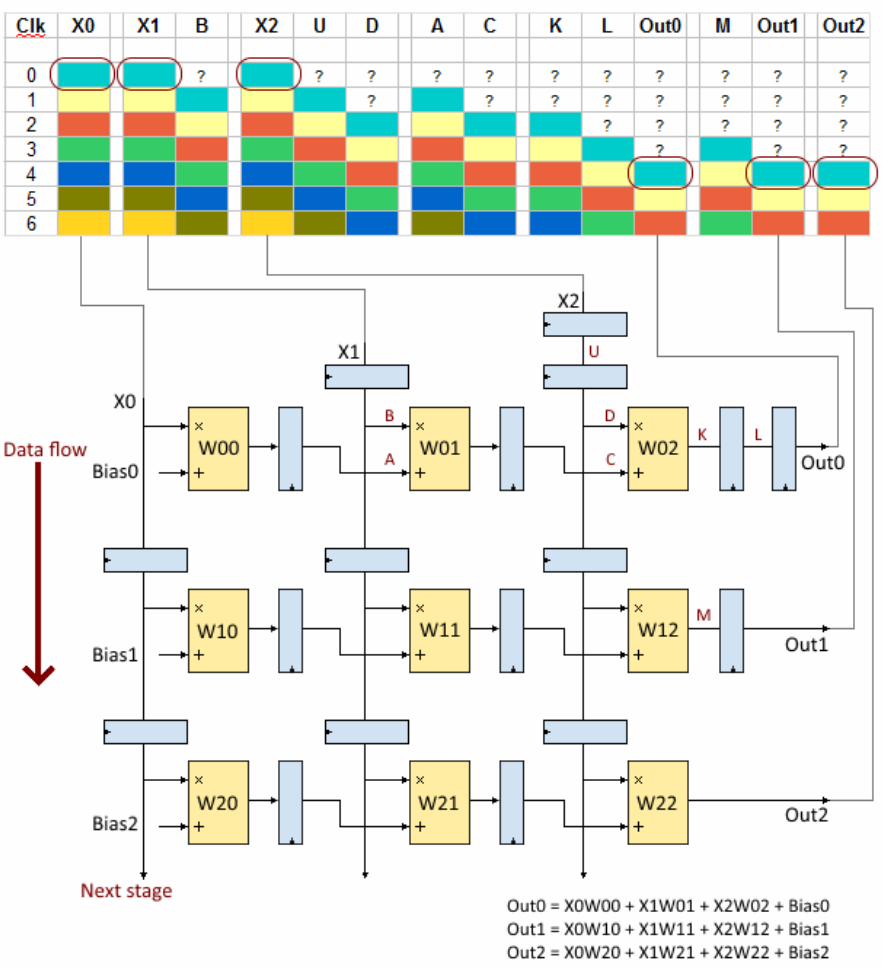


Figure 1.20: Systolic Array Edge TPU.
Source: Q-engineering

grows up this represent the solution widely used for neural network hardware.

1.7.6 Activation unit

Once the output is available, it is sent to the activation unit. This module within the Edge TPU applies the activation function to the output. It is hardwired. In other words, you cannot alter the function, it works like a ROM. Probably it is a ReLU function as it is nowadays the most used activation function and it is very easy to implement in hardware.[7]

Chapter 2

Software

This chapter describes the software developed for the hardware described in the chapter 1. The choice to use the Qt framework guarantees the portability and cross-platform support, as well as the graphical interface and high data performance the feature of compiled languages. The software is divided into two programs capable of communicating with each other: the first works on Raspberry Pi and allows the user to interface easily with the Flir thermal camera and with the Raspicam camera. It also acts as a TCP server for sending the images shot to connected devices. The second part is a client program that receives the data sent by the main device, and allows the analysis of the image through machine learning on dedicated hardware.

2.1 Signals & Slots

In the project extensive use was made of the Signal and Slot mechanism, this to allow communication between the objects in the code. Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. Signals and slots are made possible by Qt's meta-object system.[8]

2.1.1 Introduction

In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another.

Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to a function, so if you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function.

The processing function then calls the callback when appropriate. While successful frameworks using this method do exist, callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments.[8]

2.1.2 Signals and Slots

In Qt, we have an alternative to the callback technique: We use signals and slots. A signal is emitted when a particular event occurs. Qt's widgets have many predefined signals, but we can always subclass widgets to add our own signals to them. A slot is a function that is called in response to a particular signal. Qt's widgets have many pre-defined slots, but it is common practice to subclass widgets and add your own slots so that you can handle the signals that you are interested in.

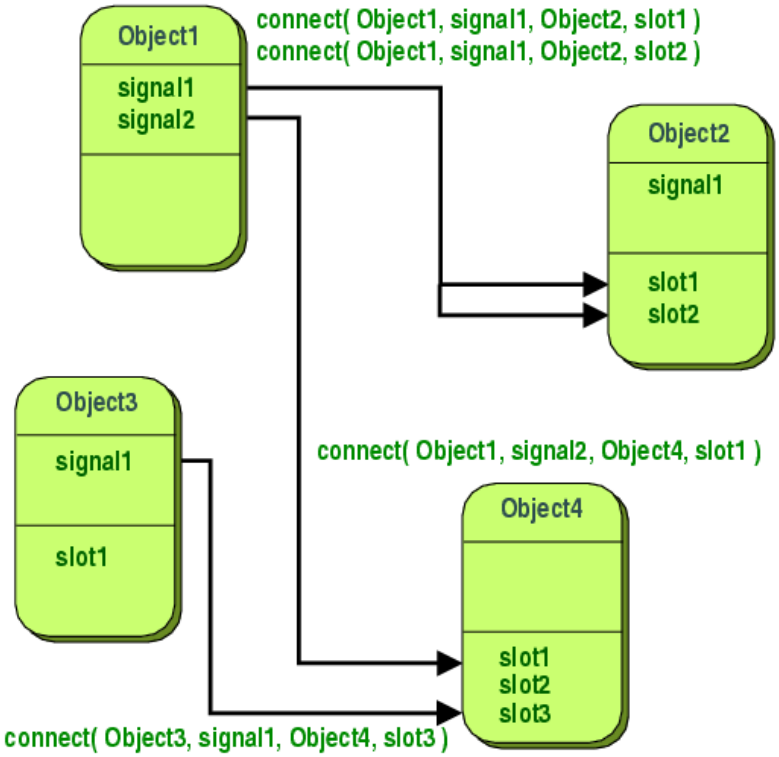


Figure 2.1: Signal and Slot scheme.
Source: https://doc.qt.io/qt-5/signalsandslots.html

The signals and slots mechanism is type safe: The signature of a signal must match the signature of the receiving slot. (In fact a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.) Since the signatures are compatible, the compiler can help us detect type mismatches when using the function pointer-based syntax. The string-based **SIGNAL** and **SLOT** syntax will detect type mismatches at

runtime. Signals and slots are loosely coupled: A class which emits a signal neither knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensures that if you connect a signal to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. They are completely type safe.

All classes that inherit from `QObject` or one of its subclasses (e.g., `QWidget`) can contain signals and slots. Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation, and ensures that the object can be used as a software component.

Slots can be used for receiving signals, but they are also normal member functions. Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be created with Qt.

You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need. It is even possible to connect a signal directly to another signal. (This will emit the second signal immediately whenever the first is emitted.)

Together, signals and slots make up a powerful component programming mechanism.[8]

2.1.3 Signals

Signals are emitted by an object when its internal state has changed in some way that might be interesting to the object's client or owner. Signals are public access functions and can be emitted from anywhere, but we recommend to only emit them from the class that defines the signal and its subclasses.

When a signal is emitted, the slots connected to it are usually executed immediately, just like a normal function call. When this happens, the signals and slots mechanism is totally independent of any GUI event loop. Execution of the code following the emit statement will occur once all slots have returned. The situation is slightly different when using queued connections; in such a case, the code following the `emit` keyword will continue immediately, and the slots will be executed later.

If several slots are connected to one signal, the slots will be executed one after the other, in the order they have been connected, when the signal is emitted.

Signals are automatically generated by the *moc* and must not be implemented in the `.cpp` file. They can never have return types (i.e. use void).[8]

2.1.4 Slots

A slot is called when a signal connected to it is emitted. Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.

Since slots are normal member functions, they follow the normal C++ rules when called directly. However, as slots, they can be invoked by any component, regardless of its access level, via a signal-slot connection. This means that a signal emitted from an instance of an arbitrary class can cause a private slot to be invoked in an instance of an unrelated class.

Compared to callbacks, signals and slots are slightly slower because of the increased flexibility they provide, although the difference for real applications is insignificant. In general, emitting a signal that is connected to some slots, is approximately ten times slower than calling the receivers directly, with non-virtual function calls. This is the overhead required to locate the connection object, to safely iterate over all connections (i.e. checking that subsequent receivers have not been destroyed during the emission), and to marshall any parameters in a generic fashion. While ten non-virtual function calls may sound like a lot, it's much less overhead than any `new` or `delete` operation, for example. As soon as you perform a string, vector or list operation that behind the scene requires `new` or `delete`, the signals and slots overhead is only responsible for a very small proportion of the complete function call costs. The same is true whenever you do a system call in a slot; or indirectly call more than ten functions. The simplicity and flexibility of the signals and slots mechanism is well worth the overhead, which your users won't even notice.[8]

2.2 Interface

As anticipated, the software run on the Raspberry Pi is made following the paradigm of object-oriented programming. Made in C++/Qt making extensive use of the proprietary classes of the framework and the Standard Template Library (STL). This program has some dependencies with regard to the thermal camera drivers supplied by the manufacturer and are 32-bit, since the Raspberry Pi 3, described in 1.1, is equipped with a 32-bit ARM Cortex-A53 processor. These allow total control of the camera. The second dependency is the Raspicam library which allows the interface with the RGB camera allowing the image acquisition. Continuing a user interface has been created through the use of widgets, this is divided into three areas. The first area shows the video streams acquired separately in two labels, the third larger label shows the two streams mixed through filters made available as default the overlay filter is used. It was preferred not to perform a

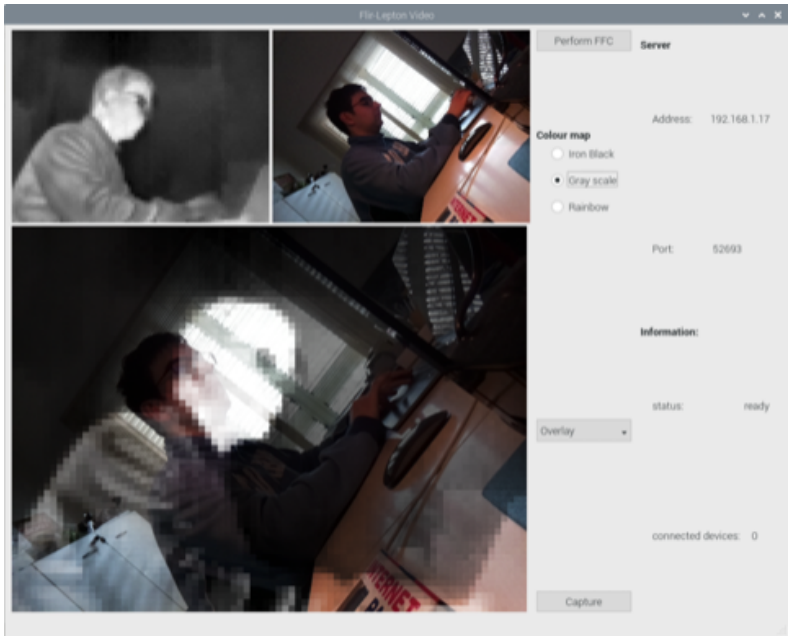
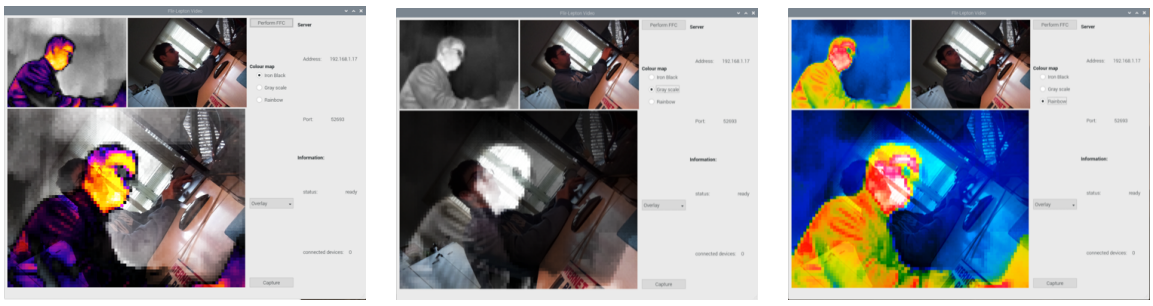


Figure 2.2: User interface run on Raspberry Pi 3b.

match of the images with pixel-by-pixel recalculation for two reasons: the first due to the high difference in size of the sensors as that of the thermal camera is 80×60 pixels while the Raspicam is 3280×2464 pixels. Furthermore motivation is due to not aggravate the computational load on the CPU. The second area provides some controls for the user, in fact it is possible to save thermal images on files during the acquisition. you can change the heat map applied to the image on the fly by choosing from three different possibilities, in the figure you can observe the result. Finally, it is possible to modify the mixing filter, also on the fly, of the video streams to obtain different effects to improve visibility or to increase details. The last section of the user interface shows the information relating to the TCP socket used for sending the video stream to other devices.



(a) *lava.* **(b)** *grayscale.* **(c)** *rainbow.*

Figure 2.3: Result colourization applied to thermal camera data.

2.2.1 Software Analysis

The user interface is run on the main thread, in order to maximize the performance of the executed code the image acquisition operation is performed in the secondary thread. Thanks to the use of the signal and slot system of Qt, introduced before in (2.1), it is possible to update the labels without blocks typical of multi-threaded programming. The difficulty of concurrent programming usually consists in synchronizing the access to resources by different threads that act in competition on the same resources. Having two or more threads accessing the same data simultaneously can lead to unexpected and unwanted results. In fact, without the application of particular programming techniques, it is not possible to predict in a deterministic way, at the time of execution, when that specific thread will be executed: their progression depends on the priorities decided by the scheduler of the operating system and not by the programmer. In fact, multiple threads can access the same variable and modify its content or value. Therefore, synchronization techniques such as mutual exclusion are used to solve the problem. As a result, ideally a thread should execute code as independent of the rest of the program as possible. Furthermore, errors in synchronization between threads are often very difficult to detect because their occurrence essentially depends on the environment in which the program is run. The synchronization of one thread with another is normally necessary to allow them to communicate with each other and to return the results of a function to the main process; it is normally done through *mutex*[9]. Analysing the code of the thread that takes care of acquiring the images we can see that it proceeds without mutex[1], but uses the signal and slot system as mentioned before.

```
36  void LeptonThread::run() {
37    m_ir_image = QImage(80, 60, QImage::Format_RGB888);
38    leptonSPI_OpenPort(0);
39    usleep(LeptonLoadTime);
40    while (true) {
41      int resets = 0;
42      for (int j = 0; j < PACKETS_PER_FRAME; j++) {
43        read(spi_cs0_fd, result + sizeof(uint8_t) * PACKET_SIZE * j,
44             sizeof(uint8_t) * PACKET_SIZE);
45        int packetNumber = result[j * PACKET_SIZE + 1];
46        if (packetNumber != j) {
47          j = -1;
48          resets += 1;
49          usleep(LeptonResetTime);
50          if (resets == MaxResetsPerSegment) {
51            leptonSPI_ClosePort(0);
52            usleep(LeptonRebootTime);
53            leptonSPI_OpenPort(0);
```

[1]The mutex class is a synchronization primitive that can be used to protect shared data from being simultaneously accessed by multiple threads.

```
54          }
55        }
56      }
57      m_frameBuffer = (uint16_t *)result;
58      int row, column;
59      uint16_t value;
60      uint16_t minValue = 65535;
61      uint16_t maxValue = 0;
62      for (int i = 0; i < FRAME_SIZE_UINT16; i++) {
63        if (i % PACKET_SIZE_UINT16 < 2) {
64          continue;
65        }
66        auto temp = result[i * 2];
67        result[i * 2] = result[i * 2 + 1];
68        result[i * 2 + 1] = temp;
69        value = m_frameBuffer[i];
70        if (value > maxValue) {
71          maxValue = value;
72        }
73        if (value < minValue) {
74          minValue = value;
75        }
76        column = i % PACKET_SIZE_UINT16 - 2;
77        row = i / PACKET_SIZE_UINT16;
78      }
79      float diff = static_cast<float>(maxValue - minValue);
80      float scale = 255 / diff;
81      QRgb color;
82      for (int i = 0; i < FRAME_SIZE_UINT16; i++) {
83        if (i % PACKET_SIZE_UINT16 < 2) {
84          continue;
85        }
86        value = (m_frameBuffer[i] - minValue) * scale;
87        color = qRgb(this->colorMap[3 * value], this->colorMap[3 * value + 1],
88                     this->colorMap[3 * value + 2]);
89        column = (i % PACKET_SIZE_UINT16) - 2;
90        row = i / PACKET_SIZE_UINT16;
91        m_ir_image.setPixel(column, row, color);
92      }
93      QImage colour_image = cam->getImageRGB();
94      emit updateImage(m_ir_image);
95      emit updateCam(colour_image);
96      recalculateResult(m_ir_image.scaled(640, 480, Qt::KeepAspectRatio),
97                        colour_image.scaled(640, 480, Qt::KeepAspectRatio));
98    }
99    leptonSPI_ClosePort(0);
100 }
```

Listing 1: Infinite loop thread cameras.

Going to analyse in detail the infinite loop, reported in the listing (1), executed in a thread other than the main one that manages only the main interface, it can be observed that:

- (line 37–56) When the function starts, an instance of the `QImage` type object is instantiated to contain the image acquired during the cycle. The parameters of the frame size are provided and the color space this allows to increase the speed of the cycle as it will not be cyclically cancelled and reallocated the same, but will be reused. The communication with the thermal camera is opened via the SPI port if the cycle is not started or if the communication is interrupted the communication is closed.

- (line 66–78) The main cycle is to acquire data from the saved camera registers, as the order is MSB[2], reversed in according to LSB[3]. The values are then scaled to obtain a consistent representation of the hot and cold areas. Then the color map is applied to color the raw data obtained from the scaling process described above.

- (line 79–97) Finally, signals are output for the coloured thermal image, for the RGB image acquired by the Raspicam library and passed through the slot and the last resulting from mixing of previous two depend on effect of selected in UI interface.

As can be seen, the cycle proceeds without mutuals or blocking conditions, in fact, as previously described, it is possible to change the color map or the mixing tool on the fly.

[2]MSB can also stand for "most significant byte". *Big-endian processor*: When data is loaded into a multi-byte register, the first byte (with the lowest address) is the most significant byte of the data.[10]

[3]LSB can also stand for "least significant byte". *Little-endian processor*: When data is loaded into a multi-byte register, the first byte (with the lowest address) is the least significant byte of the data.[10]

2.3 Communication systems

In digital data communications, wiring together two or more devices is one of the first steps in establishing a network. As well as this hardware requirement, software must also be addressed. The Open System Interconnection (OSI) model proposed by the International Organization for Standardization (ISO) is a standard way to structure communication software that is applicable to any network. The model has been standardized by ISO and International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) which is the organization coordinating standards for telecommunications. The communication is based on low-level message passing between the communicating systems.

- A wants to communicate with B;
- A builds a message addressing B;
- A executes a call to the communication module to send the message to B.

Of course, A and B need to speak the same language, i.e., they need to agree on the meaning of the bits being sent. The protocols define the rules for communication. When data is exchanged through a computer network, the system rules are called a network protocol. A protocol must define the syntax, semantics, and timing of communication (i.e. how, what and when); the specified behaviour is typically independent of how it is to be implemented. Syntax: refers to the structure or the format of the data. Semantics: the way in which the bit patterns are interpreted. Timing: specify when the data can be sent and how fast it will be. Another term is synchronization. The communication between two nodes of a network occurs by sending messages. a message is broken down into a sequence of packets, and each packet is transmitted individually. The structure includes some control bits at the beginning of the message called *header* and at the end called *footer*. the control bits can contain information such as: the sending node of the packet, the recipient node of the package, package length information, information that allows you to verify the correctness of the package. [11]

2.3.1 Architecture

Client/server architectures are based on the functional division of IT applications into two categories. Few computers act as servers, run a particular program that allows the computer to receive requests and send replies. As shows in figure (2.4).
The clients, on the other hand, run a program that allows them to send requests and receive replies. Client/server applications are two-level architectures, that is, the first

level is the client and the second level is the server. The interaction protocols between client and server are quite simple. among the advantages of client/server architectures we have the simplicity of construction and the possibility of having easy-to-use clients. Disadvantages include the risk of overloading the computer acting as a server and the communication channel with which the server is connected to the network.[11]

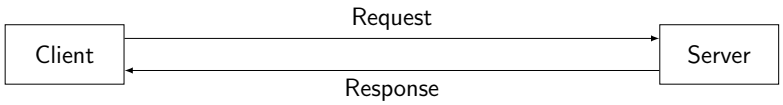


Figure 2.4: Two level client/server architecture

In the Internet protocol suite there is also the Transmission Control Protocol (TCP). It is the complement of the Internet Protocol, yielding the well known TCP/IP. TCP provides reliable, ordered, and error-checked delivery of the messages between applications on an IP network. The most used internet applications, i.e. the World Wide Web, e-mail, file transfer, rely on TCP.

2.3.2 Server implementation

In our case, it implements `QTcpSocket` network communication. So, there is a server service in main program shows in figure (2.5). While there is a client application deepened later in section (2.4). Server application has `QTcpSocket` and it listens to some port, in our case 52693. Client has `QTcpSocket`, but has not connected to the server yet:

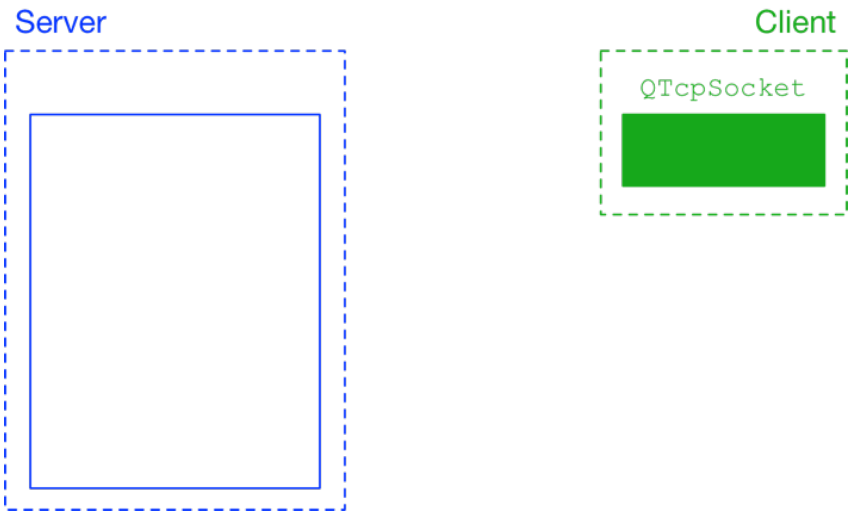


Figure 2.5: Interface client server `QTcpSocket`.

When client connects to server, a `QTcpSocket` is created on the server's side, through which server and client can talk to each other and start send message, shows in figure (2.6). In particular, we observe the function reported in (2) implemented in server application

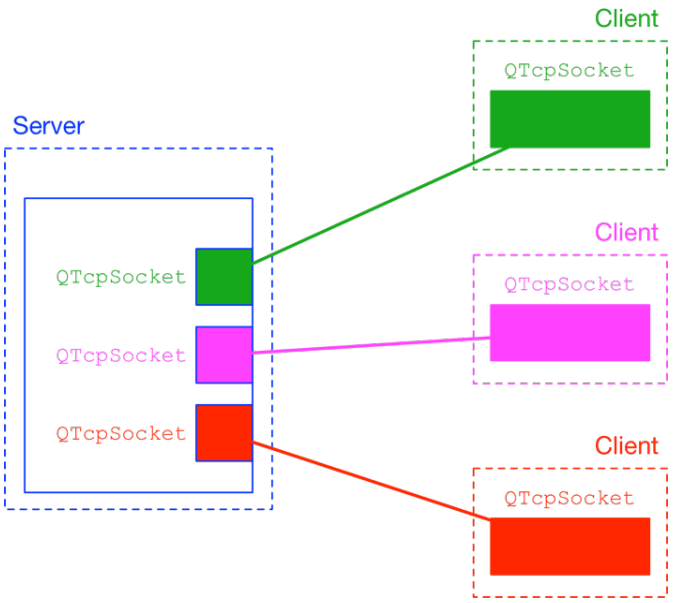


Figure 2.6: Connection between server-clients.

which sends the image just captured by the RGB camera to the server to be encapsulated in the message before send. In the function it is possible to observe the presence of a mutex which is locked to protect the resource and avoid overwriting through function calls. The frame, i.e. the resource, is converted and stored in buffer before sending. Upon exiting the function body, the mutex is unlocked, thus giving free access to resources again.

```
84  connect(this, &MainWindow::update_rgb_image, [=](QImage image) {
85    QMutexLocker locker(&mutex);
86    QPixmap img =
87        QPixmap::fromImage(image.scaled(512, 512, Qt::KeepAspectRatio));
88    QByteArray bImage;
89    QBuffer bBuffer(&bImage);
90    // Putting every image in the buffer
91    bBuffer.open(QIODevice::ReadWrite);
92    img.save(&bBuffer, "JPG");
93    // Sending to TCPServer function to display the image
94    server->is_newImg(bImage);
95  });
```

Listing 2: Particular report function sending image.

2.4 Client

The software that acts as a client on the Coral dev-board, presented in chapter (1.6), created by Google is based on the same Qt framework previously introduced in order to guarantee portability and reliability of the code. This, remember, has an ARM Cortex-A53 processor, but unlike the one mounted on Raspberry it executes 64-bit code and takes advantage of the new armv8 architecture with a significant performance gain. This difference arises from the execution of Machine Learning operations supported by the TPU, described in detail in section (1.7). As seen for the main program, here too we find a main interface executed in the main thread. The simplest interface is divided into two areas: the first one where it is possible to observe the flow of images coming from the TCP socket. As hows in figure (2.6).
On the other hand, the second zone offers the possibility of connecting to the TCP socket by entering the address and port of the machine on which the server is run, which is listening for possible connection requests. Once the connection between server and client is established, the label is updated with the images received.

2.4.1 Software Analysis

To avoid blockages and unpleasant delays in receiving images from the TCP socket, multi-thread programming was used. In fact, as previously described, the interface managed by QWidget is run on the main thread. If the connection is stable, the thread starts which allows reception in a queue waiting to leave.
On the other hand, this is prepared when the class is instantiated within the main function. Since this infinite cycle is the critical factor, we analyse its structure in detail below.

```
12  void StreamerThread::run() {
13    QMutexLocker locker(&mutex);
14    socket = new QTcpSocket;
15    socket->connectToHost(m_address, m_port);
16    QByteArray buffer;
17    while (m_quit == false) {
18      if (socket->waitForReadyRead(3000)) {
19        buffer.append(socket->readAll());
20        msleep(350);
21        emit newImageAvailabe(buffer);
22        buffer.clear();
23      }
24    }
25    socket = nullptr;
26  }
```

Listing 3: Particular report function sending image.

As you can see in the function code, shown in Listing (3), we observe the instance of the `socket` member object of the `QTcpSocket` class type. By starting the connection, the server. The critical section from the *while loop* is protected by a mutex, highlighted by the `QMutexLocker` class, a mutex indicates a process of synchronization between concurrent processes or threads, with which multiple parallel tasks are prevented from simultaneously accessing data in memory or other resources subject to race condition.[12]
Locking and unlocking a `QMutex` in complex functions and statements or in exception handling code is error-prone. `QMutexLocker` can be used in such situations to ensure that the state of the mutex is always well-defined. `QMutexLocker` should be created within a function where a `QMutex` needs to be locked. The mutex is locked when `QMutexLocker` is created. If locked, the mutex will be unlocked when the `QMutexLocker` is destroyed. Using `QMutexLocker` greatly simplifies the code, and makes it more readable.[13]
The buffer is filled with reading from the socket and before putting the signal to update the image in the interface a small interval of time is waited to guarantee the complete reception of the image. Before updating the label on the dashboard, filter the image to verify that it is consistent and different from an empty or corrupt image, shows in listing (4). If this occurs, the function is immediately exited to avoid viewing an image being ready to receive a new image.

```
88  void TcpClient::imageAvailabe(QByteArray baImage) {
89    QPixmap pixImage;
90    QImage image;
91
92    if (!pixImage.loadFromData(baImage, "JPG")) return;
93    image = pixImage.toImage();
94    if (image.pixel(image.width() - 1, image.height() - 1) == 4286611584 &&
95        image.pixel(image.width() / 2, image.height() - 1) == 4286611584 &&
96        image.pixel(0, image.height() - 1) == 4286611584)
97      return;
98
99    emit updatePixmap(pixImage);
100 }
```

Listing 4: Implantation filter.

2.4.2 CNN-PART-TPU-INFERENCE

...

Chapter 3

Neural Networks

THIS ...

3.1 Introduction

In this era, a large amount of structured and unstructured data became available. **Machine Learning** (**ML**) has evolved as a branch of artificial intelligence: it envisages the development of self-learning algorithms, which are capable of acquiring knowledge from data with the aim of making predictions. Instead of requiring a human presence who manually enact the rules and build models for the analysis of large amounts of data, machine learning offers a more efficient alternative to capture the knowledge in the data. Machine learning aims to gradually improve the performance of forecasting models and to make data driven decisions. In this section we will examine the three different types of machine learning: *supervised learning*, *unsupervised learning* and *reinforcement learning*. Where we will show the fundamental differences between these types of learning.[14]

3.1.1 Supervised learning

The main purpose of supervised learning is to derive a model from training data, which allows us to make predictions for data that are not available or future. Here, the term "supervision" refers to the fact that the output signal labels of the sample sets are already known. A supervised learning task, which is based on discrete class labels, is also called a classification task, in figure 3.1 it is possible to observe a process diagram. Another supervised learning subcategory is regression, whose resulting signal is a continuous value. Classification is a sub-category of supervised learning, which has the goal to provide class category labels for new instances, based on observations made in the past.
These labels are discrete, unordered values that can be considered as belonging to a group of instances. However, the set of class labels does not necessarily have to be a binary nature. The predictive model identified by a supervised learning algorithm can consider each class label that is present in the learning dataset of a new instance, which is not labelled. A typical example of *multi-class classification* is the recognition of hand-written text.[14]

3.1.2 Reinforcement learnig

Another type of machine learning is reinforcement learning. Here, the goal is to develop a system (*agent*) for people to improve their performance. In order to do so, that system is based on interactions with the environment. Since information relating to the current state of the environment include also a *reward* signal, we can consider strengthening learning as an example of supervised learning. However, this feedback is not the correct label or the true value of truth, but it represents the quality of the measurement of the per-

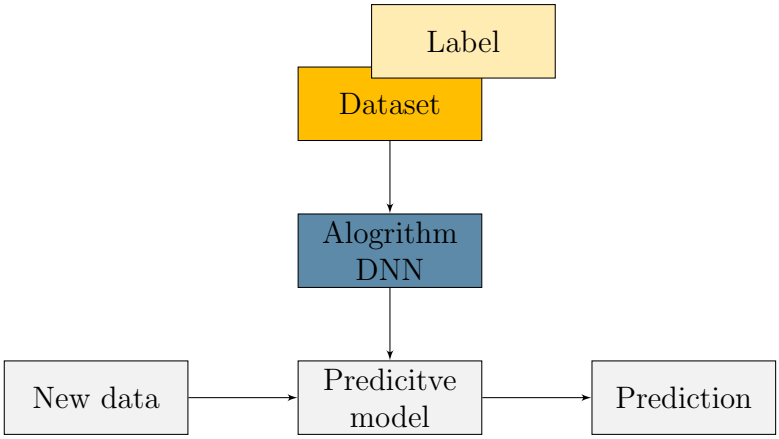


Figure 3.1: supervised learning scheme

formance measured by the reward function. Through interaction with the environment, an agent can then use reinforcement learning to learn a series of actions, which maximize this reward through a trial-and-error exploratory approach or deliberative planning.[14]

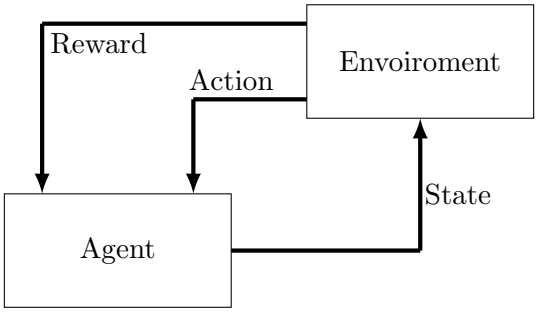


Figure 3.2: reinforcement learning scheme

3.1.3 Unsupervised learning

In supervised learning, we know in advance the correct answer when we describe our model, while in reinforcement learning we define a measure, or reward, for the specific actions performed by the agent. In unsupervised learning, on the other hand, we are dealing with unlabelled data or data from the unknown structure. Using unsupervised learning techniques, we are able to observe the structure of our data, to extract meaningful information from them without being able to rely on the guide nor a variable known relative result, nor a reward function. Clustering is an exploratory technique of data analysis that allows us to organize a series of information within meaningful groups (*cluster*) without having any previous knowledge of memberships in such groups. Each cluster that can be derived during the analysis defines a group of objects that share a certain degree of similarity, but which are more dissimilar than the objects present in the other clusters,

which is why clustering is sometimes called *"unsupervised classification"*. Clustering is an excellent technique for structuring information to identify meaningful relationships in the data.[14]

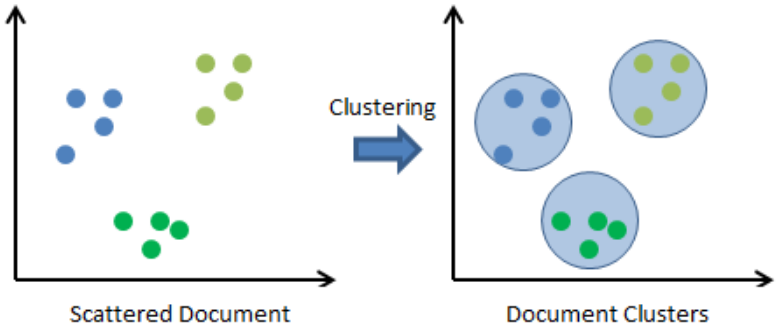


Figure 3.3: example of clustering

3.2 Deep Learning for Object Detection

3.3 Data Augmentation Strategies for Object Detection

Bibliography

[1] E. Upton, "Raspberry pi 2 on sale now at $35," *Raspberry Pi*, 2016.

[2] M. A. Pagnutti, R. E. Ryan, G. J. C. V, M. J. Gold, R. Harlan, E. Leggett, and J. F. Pagnutti, "Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes," *Journal of Electronic Imaging*, vol. 26, no. 1, pp. 1 – 13, 2017. [Online]. Available: https://doi.org/10.1117/1.JEI.26.1.013014

[3] R. Foundation. (2016) New 8-megapixel camera board on sale at $25. [Online]. Available: https://www.raspberrypi.org/blog/new-8-megapixel-camera-board-sale-25/

[4] ——. (2016) Camera module. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera/

[5] *FLIR LEPTON® Long Wave Infrared (LWIR) Datasheet*, 1st ed., October 2014.

[6] Flir lepton breakout board v1.4 by getlab | groupgets. [Online]. Available: https://groupgets.com/manufacturers/getlab/products/flir-lepton-breakout-board-v1-4

[7] Google coral edge tpu explained in depth - q-engineering. [Online]. Available: https://qengineering.eu/google-corals-tpu-explained.html

[8] Signals & slots | qt core 5.14.1. [Online]. Available: https://doc.qt.io/qt-5/signalsandslots.html

[9] Wikipedia, "Thread (informatica) — wikipedia, l'enciclopedia libera," 2019, [Online; in data 11-febbraio-2020]. [Online]. Available: http://it.wikipedia.org/w/index.php?title=Thread_(informatica)&oldid=107249669

[10] D. V. James, "Multiplexed buses: the endian wars continue," *IEEE Micro*, vol. 10, no. 3, pp. 9–21, June 1990.

[11] D. Mandrioli, *Informatica: arte e mestiere*, ser. Collana di istruzione scientifica. McGraw-Hill Companies, 2008. [Online]. Available: https://books.google.it/books?id=aae9NwAACAAJ

[12] Wikipedia, "Mutex — wikipedia, l'enciclopedia libera," 2020, [Online; in data 11-febbraio-2020]. [Online]. Available: http://it.wikipedia.org/w/index.php?title=Mutex&oldid=110301420

[13] Qmutexlocker class | qt core 5.14.1. [Online]. Available: https://doc.qt.io/qt-5/qmutexlocker.html#details

[14] S. Raschka, *Machine learning con Python. Costruire algoritmi per generare conoscenza*, ser. Guida completa. Apogeo, 2016. [Online]. Available: https://books.google.it/books?id=G7OvDAEACAAJ