SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Laurea Magistrale In Ingegneria Meccanica

POLITECNICO
MILANO 1863

ON-EDGE DEVICE FOR HUMAN TRACKING AND IDENTIFICATION BASED ON RGB CAMERA INPUT DATA

Supervisor: Prof. Marco Tarabini
Co-supervisor: Ing. Stefano Marelli

Lyubomyr Klymyuk

921135

Academic Year 2020/2021

Acknowledgements

Questo lavoro di tesi è stato possibile prima di tutto grazie al Prof. **Marco Tarabini**. Voglio ringraziarlo per aver risposto tempestivamente alla mia richiesta e per la sua disponibilità. Nonostante il periodo di incertezza in cui la maggior parte delle imprese lavoravano da remoto e difficilmente assumevano nuove risorse è riuscito a mettermi in contatto con una realtà aziendale.

Da li entro due settimane è iniziato il mio percorso di stage presso la Forwardinnovation srl di Gallarate. Ringrazio quindi il CEO **Davide Vallero** e tutti i colleghi di Forwardinnovation che mi hanno accolto nella loro squadra e fatto partecipare attivamente ai vari progetti interessanti.

In tutto questo percorso venivo guidato e seguito dall'Ing. **Stefano Marelli** nonchè PhD del Politecnico di Milano al quale va un caloroso ringraziamento per i suoi preziosi consigli.

Voglio dire grazie a tutti gli amici e colleghi del Politecnico di Milano con i quali abbiamo condiviso momenti bellissimi durante questi anni. Senza di voi questo percorso sarebbe stato molto più ostico e sciapo. Grazie per avermi supportato e sopportato nei momenti difficili oltre ad aver ascoltato pazientemente le mie lamentele. E' stato un onore avervi al mio fianco in questa avventura e spero di incontrarvi presto nel mondo del lavoro e non.

Un grazie speciale agli amici più stretti che mi hanno incoraggiato in questi anni ed aver creduto in me. Avrete sempre un posto speciale nel mio cuore.

Alla base di tutta questa piramide di duro lavoro e sacrifici ma anche lungo tutto il percorso ed ora alla fine ovviamente ci siete voi, i miei super genitori. Un enorme grazie per il supporto morale, economico e di ogni altro tipo che mi avete dato in questi anni. Senza di voi questo non sarebbe mai accaduto. Potete essere fieri ed orgogliosi di me. Dopo tutti questi anni di incertezze e difficoltà vostro figlio ce l'ha fatta!

Contents

List of Figures

Chapter 1

Introduction

There are currently several tracking systems on the market that use ultra wideband (UWB) and bluetooth low energy (BLE) technology and are very accurate down to tenths of a meter i.e Ubisense, Siemens, Eliko, Sewio ecc. The problem lies in the cost of the hardware that in particular for the UWB technology for now remains very high. In addition there is the need to wear battery-powered tags able to send the signal to the various gateways installed along the perimeter that in turn forward the various signals received to the central server for the processing of the position in real time.

Within the industrial context this is not a problem but if you consider a supermarket or a bank for example, making customers wear devices becomes impractical.

The goal then becomes to select hardware powerful enough to process the frames coming from the video input in real time while maximizing fps.

The processing and analysis of images is performed through special computer vision software and Deep Neural Network (DNN) libraries such as OpenPose, OpenCV and Caffé using Python as a programming language. This kind of applications as you can imagine having to do with the graphic processing, estimation and calculation of the position of the body within the frame and all this in real time, requires a significant computational effort in terms of GPU (Graphics Processing Unit).

Therefore, having to select a compact hardware with a decent graphics performance and a not excessive cost, our choice fell on an embedded device designed by Nvidia which is the Jetson Nano board.

Once the system is set up, the next step is to be able to distinguish one individual from another. This is because the software, as powerful as it is in recognizing a subject in the image and sketching it with a n-point skeleton, is not yet able to distinguish between them.

Therefore the concept of digital identity is introduced. It is a unique id that is assigned to the subject based on different parameters that can be for example physiological characteristics and colors of clothing. This is necessary at the moment when a subject "A" leaves the field of visibility of camera n.1 and appears in the field of visibility of camera n.2 while remaining "A" and not becoming "B" for example.

The same is true for a single camera with several subjects entering and leaving the camera's field of view.

The aim of this thesis is to create an on-edge device capable of identify people from camera stream video in an indoor environment. The device will be part of a more complex system that is the RTLS (Real-Time Locating System) designed to detect and track people in closed environments where the GPS signal is poor or absent. The device should be mounted in the close to the video source, hence the name 'on-edge', limiting the data traffic on the internal network and respecting the privacy of detected people.

Chapter 2

State of the art

2.1 Description of a Real-Time Location System

A Real-Time Location System (RTLS) is a set of hardware and software used to identify and track objects and/or people in space within a predefined area.

There are two types of tracking systems based on the type of environment in which they are employed. The first one is used outdoors and identifies the position of the subject thanks to the GPS (Global Positioning System) signal coming from the satellite. The second is defined as indoor and is what we typically refer to as RTLS, used in closed environments where the satellite signal is usually weak. To determine the position of objects and people we go to exploit other technologies available and more suitable for the purpose and will be discussed in detail in the following paragraphs.

The general operation is based on a series of devices deployed throughout the area of interest and will exchange between them different information and signals. In common language these are defined tags and anchors where the first are mobile and are placed on the object of which we are interested in the position, while the second are fixed and will be placed on the existing infrastructure Fig 2.1.

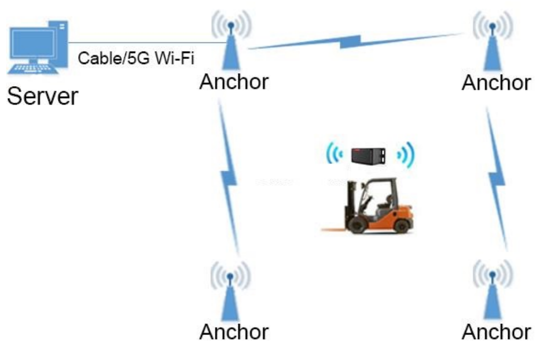


Figure 2.1: Generic RTLS system architecture

2.2 Technology overwiew

2.2.1 Radio Frequency Identification (RFID)

The technology is based on automatic radio frequency identification of objects and people using electronic tags called tags or transponders. These contain within them chips capable of storing information as well as responding to queries from other devices called readers or interrogators that despite their name are also able to write and update the information contained in the tags. An RFID system consists of three basic components as you can see below.

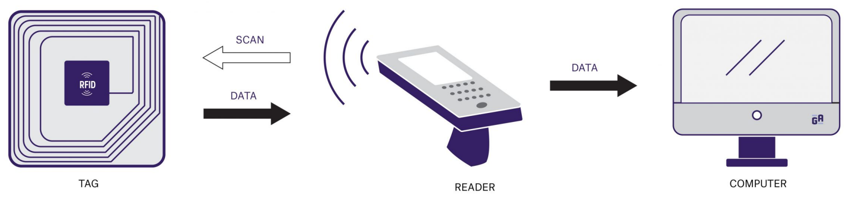


Figure 2.2: RFID system setup consisting of one or more RFID tags or transponders, a reading and/or writing device also called reader and a data management information system for transferring data to and from readers.

The transponder or tag inside contains a microchip for storing the information, an antenna for transmitting the data and a physical holder needed to hold the chip and antenna together. The latter may be active, passive, semi-passive or semi-active. Active if it has a battery to power the microchip and any sensors and can reach distances of up to 200m. Passive if the energy needed to activate the chip and transmit the information stored inside it is taken from the radio frequencies emitted by the reader. There are also mixed configurations in which the various components are powered by batteries as required at specific times to optimise operation and extend battery life. Depending on the application and configuration, the frequency range is from 125 kHz (Low Frequency Range) up to UWB (Super High Frequencies), i.e. $\geq$ 5.8 GHz.

Among the major advantages of this technology we definitely find: optimal traceability, remote reading, targetless reading, simultaneous multiple reading, information storage and update inside tags in real time and finally the tag battery life that can last several years. On the other hand it has small coverage and low detection range and it is much less precise in positioning than UWB technology for example.

2.2.2 Ultra-Wideband (UWB)

Ultra Wideband is a wireless technology used for digital data transmission over short distances at low power density. As the name implies, it occupies a very wide band compared to other transmission technologies[1], ranging from 3.1GHz to 10.6GHz, and it uses short-duration pulses over a spectrum of frequencies.

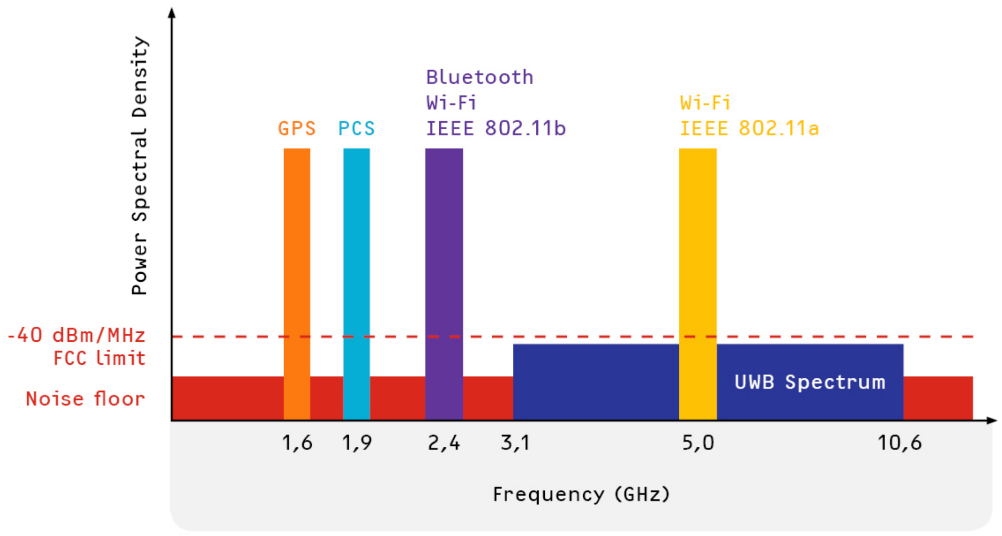


Figure 2.3: Frequency range comparison between different technologies

These intrinsic characteristics involve a series of advantages that we list below:

- Multipath - reflections on different surfaces (infrastructure, equipment and people) that are affected by the transmitted signals. Due to the short pulses distributed over a wide spectrum UWB is much more resilient compared to narrowband signals.
- Obstacles - low frequencies on the UWB frequency spectrum have long wavelengths $\lambda = c/f$ where c $\simeq 3 \cdot 10^8 m/s$ is the speed of light. The ability to penetrate different materials has made UWB technology an attractive choice for both RTLS and radar applications.
- Disturbances - there is no interference with other wireless technologies as the power level is very low and spreading evenly across a wide spectrum. One of the reasons why it has been in the military in fact this kind of signal is difficult to intercept.

[1]www.eliko.ee/uwb-technology-indoor-positioning/

- Precision - due to its wide frequency band (≥500MHz) and using techniques such as Two Way Ranging, Time Difference of Arrival or Angle of Arrival we are able to measure very precisely the final position up to few centimeters.

Among the main disadvantages we definitely have the cost of the equipment that remains very high for the moment. It must be said that according to estimates made by Global Real Time Location System (RTLS) Market (2020-2025) by Mordor Intelligence as well as many others, by 2025 we will have a strong growth of the RLTS market, consequently we will have a lowering of the costs of technology. This kind of application is essential, for example, in logistics to obtain analyses of warehouse flows, in production to know the processing stages of a particular object (product).

2.2.3 Bluetooth Low Energy (BLE)

Bluetooth technologies have become very valuable for location-based applications since the introduction of the standard Bluetooth 4.0 (Bluetooth Low Energy) which includes location-oriented specific profiles.

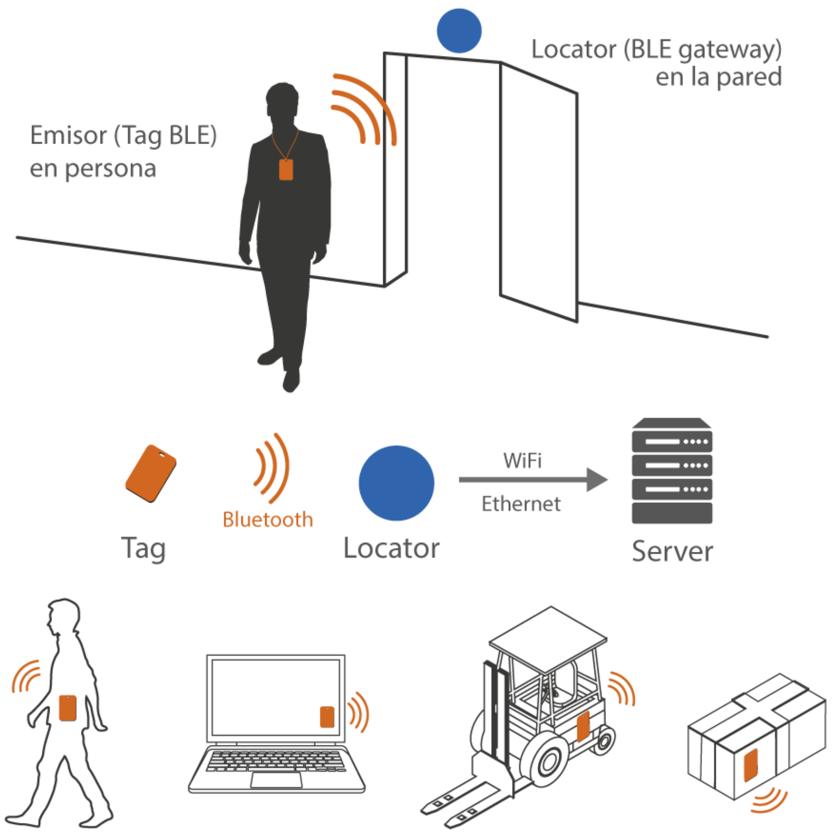


Figure 2.4: Bluetooth Low Energy RTLS system generic layout.

Like other identification technologies, it is based on two elements: gateways and tags (or transponders). In the case of BLE technology, these objects are called beacons. The gateway/controller is assigned the role of an intelligent communication tool, while the moving beacon tag (operating as an active RFID tag) transmits the signal (identification code) to the BLE gateway. There are also some types of beacons equipped with environmental sensors, which can measure temperature, humidity and other parameters. These beacons are mainly used in industrial production, storage and logistics.

The distinguishing features of this technology include: wireless communication up to a radius of 100 metres, very low energy consumption, low costs compared to other RTLS systems and easy recognition even from devices such as smartphones and tablets.

The main disadvantage is a lower accuracy with respect to UWB and similar to RFID technology. However, as Quuppa (one of the companies operating in the RLTS sector) states, by combining Bluetooth wireless technology with direction finding signal processing methodology, enables the superior positioning accuracy compared to solutions that are based on Received Signal Strength Indication (RSSI). And this can be implemented through intelligent use of position calculation algorithms based on angle of arrival (AoA) and time difference of arrival (TDOA).

2.2.4 Wi-Fi

The main advantage of these systems is that they can use the pre-existing network infrastructure and they are available in mobile phones and other wearable devices. This makes them easy to deploy and cheaper than ad-hoc installations. The frequency range is shown as usually in Fig.2.3 and given the reported power level, it is clear that consumption will not be among the lowest.

The operating principle is based on measuring the intensity of the RSSI (Received Signal Strength Indicator). The strength of the signal depends on the distance between the sender and the receiver. By simply measuring the RSSI of the tag (e.g. a mobile phone) to multiple WI FI access points (which act as anchors), it is possible to estimate the position of the mobile phone using trilateration, the same principle used in ultrasound IPS (Indoor Positioning System). The main difficulty for these systems is that WIFI signals vary enormously in the presence of obstacles and moving people. Also, different materials affect the signals differently which affects accuracy. On one hand, WIFI offers great coverage, although power use is considerable and something to be taken into account.

2.2.5 Infrared (IR)

Infrared radiation is electromagnetic radiation with a frequency band between 700nm and 1mm. Hence the name infra (below) red (visible colour with the lowest frequency 760nm). The wavelength range therefore corresponds to a frequency range of approximately 430THz to 300GHz.

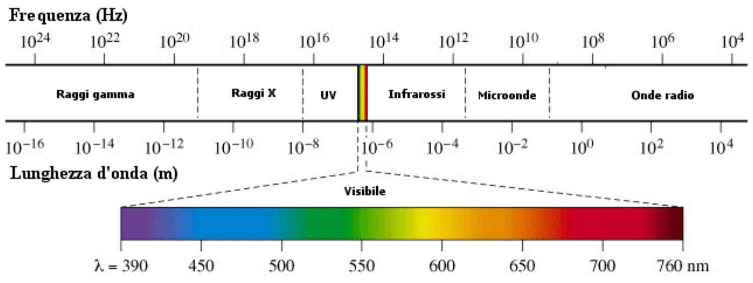


Figure 2.5: Frequency range of visible light and respective wavelenght

The technology originated in the military as usual to be able to identify targets in low light and visibility and/or night vision conditions. Infrared sensors convert the incoming radiation into an image where it can be monochromatic (grey scale) or a false colour system can be used to represent different temperatures. A tag emits a unique Infrared ID picked up by an Infrared reader. The tag is simply a diode emitting infrared radiation (infrared LED) that is focused and modulated by lenses so that it can carry information.

These highly reliable systems are also expensive to install. They require an **unobstructed** Line of Sight (LOS) between the anchor and the tag. This type of system can be used as a very reliable room detector. Since light cannot traverse walls, it is not possible for a tag to detect light from an anchor without being in the same room. For precise localization, they require installing many anchors and can struggle due to the low quality of the signal strength measurements required to compute the position from multiple anchors. While the tags are low-cost and long-lasting, a drawback of infrared is that every room needs a wired IR reader to be installed in the ceiling. That's fine if you're installing it in new construction, but retrofitting will be expensive. That's why infrared systems are commonly used in new hospital construction, where rooms are definitively segmented. In an open-space warehouse instead, infrared detection and tracking would be a challenge.

2.2.6 Ultrasound

Sound waves with a frequency above 20KHz, which is the threshold of hearing of the human ear. They can reach up to several GHz, which is why we are unable to perceive these acoustic signals.

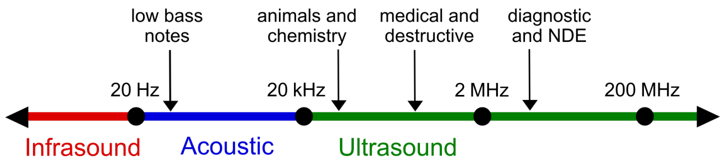


Figure 2.6: Sound waves frequency range and their applications

They are used in various fields including ultrasound imaging in medicine, non-destructive testing, cleaning, welding and last but not least object detection and distance measurement. The last two in particular are reproductions of what bats, porpoises and other animals already do to detect their prey and avoid obstacles/being eaten in nature.

An acoustic system works almost exactly like UWB except it uses sound instead of radio. One benefit of using sound has to do with resolving multipath. If you're sending a transmission and taking a time measurement, you can guess the location based on the signal speed. If that signal bounces off the wall on the way there, you now have a multipath or maybe dozens of them. The ability to mathematically differentiate between a direct path and a multipath is purely a function of the speed of the medium divided by the bandwidth.

Now suppose we have the radar installed at 10m height and an aircraft at an altitude of 10km with a distance of 5km between the two. If we use an acoustic signal ($c = 340m/s$) the arrival time of the information will be 0.117s using the following formula and scheme:

$$t_{signal} = \frac{2h_A h_B}{cd} \tag{2.1}$$

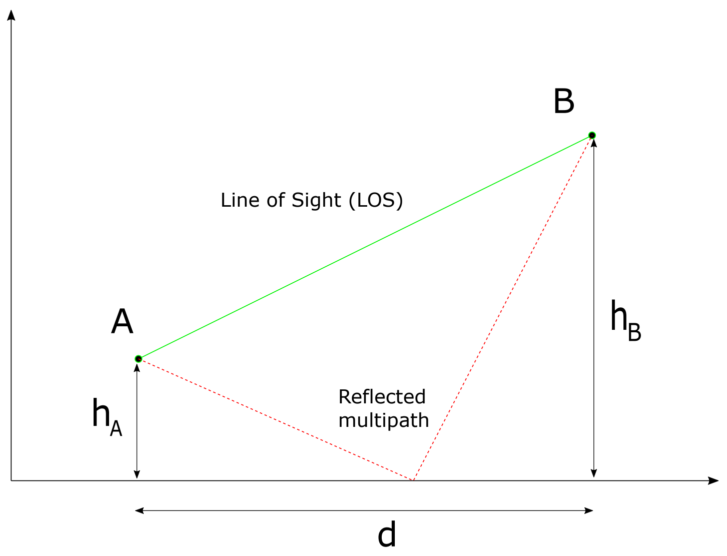


Figure 2.7: Diagram of a radio communication between two points. A is the radar receiver for example and B an aerial or satellite signal generator.

Using instead a transmission via an optical signal ($c \simeq 300 \times 10^6 m/s$) the reception time is 0.133μs. We can summarise the results in the following table where we also calculate the required bandwidth f=1/T.

Signal type	Velocity of propagation	Time of flight	Bandwidth
Acoustical	340 m/s	0.117 s	8.55 Hz
Optical	300×10^6 m/s	0.133μs	7.5 MHz

Acoustic systems require less signal bandwidth to resolve multipath because the speed of sound is so much less than the speed of light. The cost will depend on the type of installation. If the installation is to be made in a new building, the costs will be low, but if an existing infrastructure is to be adapted with sensors and wires, the costs tend to rise. However the tags are unexpensive.

2.3 On-edge hardware

2.3.1 On-edge device requirements

An on-edge device is defined as such because it performs its functionality according to its type and field of application located close to the main source of input data. Let us take as an example a weather station with its sensors measuring temperature, wind speed and direction, located high in the mountains. It will need to send information with a certain regularity so that it can be of help to mountaineers who intend to climb at high altitude in addition to its normal monitoring function.

The data generated by each of these weather stations located over a certain territory will have to be processed by a central server, usually far away, in order to be reliable. If we imagine the distances covered by the signals exchanged between the station and the server, these can easily reach hundreds of kilometres depending on the geographical location.

Let us assume that this information is transmitted via a physical cable connecting the two points, and this is where the problem of communication delay arises. This phenomenon is called **latency** and is directly proportional to the length of the cable in which the electrical signals travel. In addition we have to take into account the various delays introduced by the electronic and electrical components connected to the two ends of the cable and the interference experienced during the journey. All this leads to a more or less important time delay depending on the criticality of the system we are monitoring, as well as to the deterioration of the signal content.

Another application, this time closer to the subject of this thesis, could be a surveillance camera installed inside a bank vault that transmits video data to the server for processing. In this case, in addition to the distances involved, there is also the size of the data, which is particularly onerous for video streams.

It is precisely considerations such as those mentioned above, as well as many other practical applications that have triggered the evolution of technology to place the information processing unit ever closer to its source. In addition, we have to take into account security and privacy issues related to sensitive information that is transferred over the public network. Once we constrain this data flow on a local network, the system is much less vulnerable to attacks from outside. In addition, as already mentioned in the initial introduction, there is a considerable reduction in terms of bits travelling over the wired and wireless network, thus also reducing latency. And the latter becomes a vital parameter in applications involving real-time monitoring.

Unlike **cloud computing**, edge computing devices store customer information solely on the device, and devices can be configured to erase collected data within a certain period of time. This makes edge computing devices excellent self-service tools for the banking and finance, healthcare and e-commerce industries.

In the healthcare sector, edge computing devices are used on wearables and implantable medical devices to assist patients. In most of these situations, the device can handle biomedical signal processing schemes to help devices take specific actions. The integration of edge devices into healthcare also improves the delivery of personalised medical solutions to patients.

Manufacturers in the biomedical industry and service providers who own warehouses can also leverage edge computing devices to improve shop floor operations. Connecting edge devices to material handling equipment or within specific sections helps understand shop floor traffic, inventory management and productivity. Captured data can then be used to simplify warehousing and speed up order processing activities[2].

In order to design industrial edge computing hardware that meets the challenges of industrial applications, several requirements need to be considered.

Durability must be a key consideration in the design of industrial computing hardware. This is due to the confusion and physical characteristics of shop floor operations. This means that when designing hardware, it must be constructed from durable materials that can withstand high temperatures, enhanced vibration and liquid spills. The edge hardware must also be able to function under these conditions for the long haul.

Aesthetics and architecture-in order to bring computing into the deepest parts of a facility, the hardware housing the edge computing resources must be aesthetically designed for integration into different equipment. Therefore, its construction is a key factor to be considered by suppliers as well as the final appearance of the industrial edge computing device. This is due to the fact that commercial hardware that appeals to the end user is able to do well in different markets.

The **energy** used by hardware devices adds to the total cost of overheads (TOC) of running plants. Manufacturers interested in adopting industrial cloud technology typically calculate the adoption TOC before making a move, and **power requirements** are part of the overhead column. This is why industrial edge computing hardware should consist of ultra-low power microcontroller units.

Security of production data has always been a key consideration for integrating the industrial cloud into facilities. Therefore, both data captured by software ap-

[2]www.exorint.com/it/blog/cosa-sono-i-dispositivi-di-edge-computing

plications and hardware devices must be protected from eavesdroppers and hackers. To mitigate security risks, the integration of hardware cryptographic accelerators or reverse-engineering firewalls must be used to circumvent cyber threats.

2.3.2 Edge AI hardware Market

The various sources consulted report very similar forecasts for the market we have analysed. In particular, the different analyses examined all reflect a strong growth in the edge hardware market with an increasing trend until 2028 [3]. Key drivers for the market's growth are growing demand for low latency and real-time processing on edge devices and emergence of AI coprocessors for edge computing [4]. Further, underlying opportunities for the edge AI hardware market include growing demand for edge computing in IoT and dedicated AI processors for on-device image analytics. Major restraints for the market are limited on-device training and limited number of AI experts. Power consumption and size constraint pose major challenges to the edge AI hardware market.

Based on the data reported by [4] it is estimated that the value of the AI edge market will reach 1.5 billion USD in 2024 with a growth rate (CAGR) of 20.64% calculated over the period 2019-2024 as shown in 2.8.

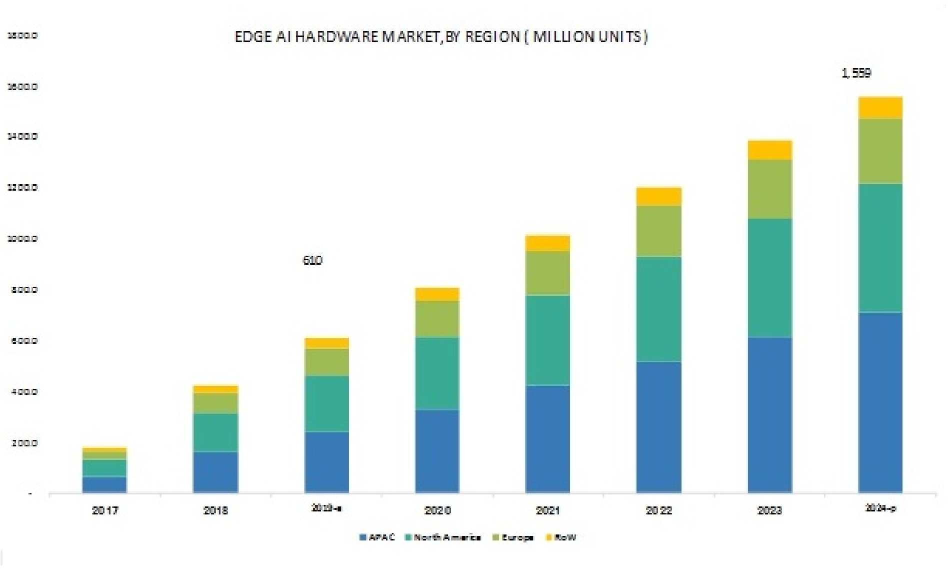


Figure 2.8: Edge AI hardware market forecast. Source: Industry experts, Secondary research, MarketsandMarkets

Four geographical macro-areas are analysed respectively: APAC (Asia-Pacific), North America, Europe and the rest of the world (RoW).

[3]www.databridgemarketresearch.com/reports/global-edge-computing-market
[4]www.marketsandmarkets.com/Market-Reports/edge-ai-hardware-market

Asia-Pacific region is expected to experience the highest growth rate in the global edge AI hardware market. The growing penetration of **smartphones** in China, Japan, India, and South Korea is expected to increase the adoption of AI processor-enabled smartphones [3]. As smartphones represent the most widely deployed edge devices, chips for smartphones have undergone rapid developments, and their capabilities have been extended to the acceleration of AI computing. To name a few, Qualcomm first applies AI hardware acceleration [8] in Snapdragon and releases Snapdragon Neural Processing Engine (SNPE) SDK, which supports almost all major DL frameworks. This is also confirmed by benchmarks carried out on the [5] site which illustrates the major players in the mobile industry including Qualcomm's Snapdragon series, Huawei and Samsung.

North America will dominate the edge computing market due to convergence of edge computing with the Industrial Internet of Things (IIoT) has emphasized manufacturers in the U.S. to move towards connected factories. Emergence of several startups providing platforms to develop an edge-enabled solution is also a driver for the market in this region.

While Asia-Pacific will be expected to witness the highest growth rate due to development of AI (Artificial Intelligence) technologies, with the opening of Microsoft Corporation's research labs in China and India. IoT device management and edge computing with 5G networks is also a driver for the market in this region[6].

Although Europe is in third place in terms of market share, it is expected to double its current share and reach USD 300 million by 2024 with almost linear growth.

Governmental bodies across the world, are embracing advanced technologies to address the important aspect of ensuring the security and safety of citizens. Surveillance serves to be a key factor in the process. Some of the major devices for edge AI hardware used by government agencies for the purpose includes surveillance cameras and drones.

With the ever-increasing population, environmental damage, and criminal activities, cities are facing new challenges each day, and this has resulted in the need for surveillance cameras. These are very much required for the prevention of incidents such as crime, burglary, and vandalism. Besides, governments also use surveillance cameras for enforcement of the law by analyzing the behavior, face recognition.

China is at the forefront of installing AI-based surveillance cameras to scan public places to track anomalies in behavior and criminal identification. A recent journal published in the New York Times revealed that the Chinese government had installed around 426 million surveillance cameras across the country in 2020. The country aims to spot crimes and accidents easily by integrating private and public cameras, to build a nation-wide surveillance network. The intention is to create a system similar to the one illustrated and anticipated in the 2011 television series 'Person of Interest' by David Semel.

[5]ai-benchmark.com/ranking

[6]www.databridgemarketresearch.com/reports/global-edge-computing-market

China is the largest market in the region, followed by Japan. Presence of several significant vendors in the automobile, electronics, and semiconductor companies, who are investing significantly in the AI technology, is driving the growth of the edge AI hardware market in the region. During a one-month period between June and July 2018, Beijing Municipal Commission of Economy and Information Technology counted around 4,040 AI companies in China. Besides, the presence of a large number of manufacturing companies makes the region an attractive market for industrial robots that implements AI technology[7].

Wearable devices also play a significant role in the increasing demand for integration with vision processing units to accelerate AI tasks. Cisco Systems estimates that the number of connected wearable devices could reach 1,105 million units by 2022. End-user industries like manufacturing, telecommunications, and automotive have huge potential in the region.

The edge AI hardware market is currently dominated by few players with their technological expertise in AI technology and the global market is expected to be consolidated in nature. Intel Corporation, NVIDIA Corporation, Qualcomm Inc., Samsung Electronics Co., Ltd., Huawei Technologies Co., Ltd., Google Inc., MediaTek Inc., Xilinx Inc., Imagination Technologies Limited, and Microsoft Corporation are some of the major players present in the current market. However, several prominent AI startups like Cambricon Technology, Horizon Robotics, Hailo Technologies, and Habana Labs are expected to compete with the key players, on the AI inferencing side[8].

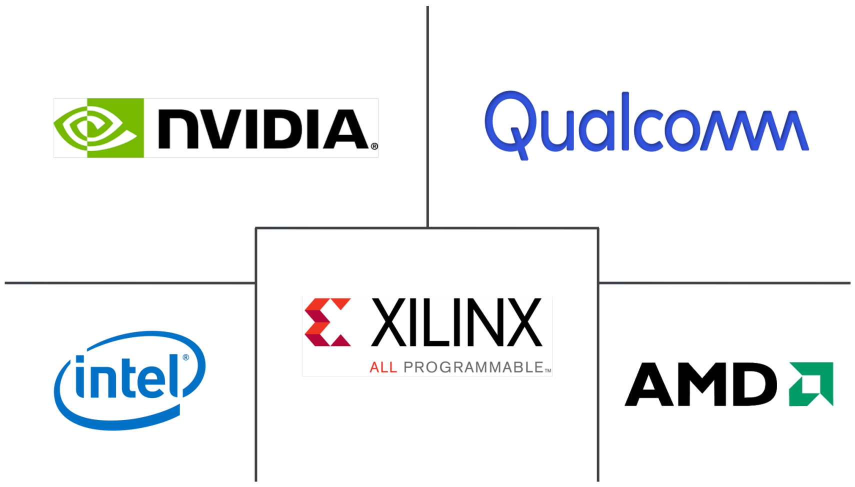


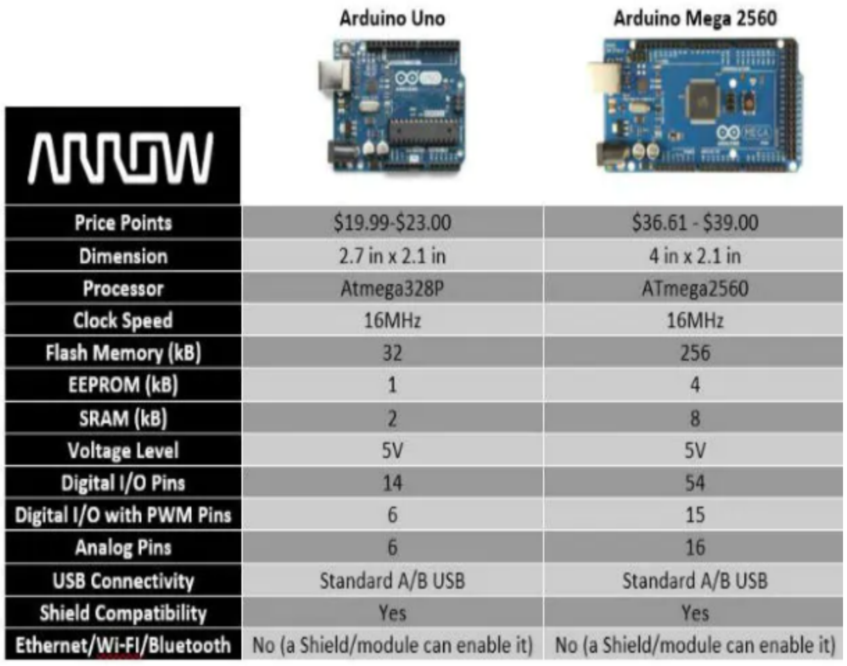
Figure 2.9: The most important players in AI chipset market

[7]www.researchandmarkets.com/reports/4828178/edge-ai-hardware-market-growth-trends-covid
[8]www.mordorintelligence.com/industry-reports/edge-ai-hardware-market

2.3.3 Hardware for computer vision on-edge

Arduino

Arduino is definitely the first thing that comes to mind when one approaches the world of automation, whatever it may be. It is a hardware platform made up of several integrated circuits with a microcontroller that manages the various digital and analogue inputs and outputs on the board. Its popularity comes from the fact that the Arduino world is simple and well-documented, it provides a great background to the world of robotics and automation, and you can find plenty of projects on the web by both hobbyists and professionals. The boards most commonly used for these purposes are the Arduino Uno and the Arduino Mega2560, the latter being an extended version of the former as you can see below[9].



ARROW	Arduino Uno	Arduino Mega 2560
Price Points	\$19.99-\$23.00	\$36.61 - \$39.00
Dimension	2.7 in x 2.1 in	4 in x 2.1 in
Processor	Atmega328P	ATmega2560
Clock Speed	16MHz	16MHz
Flash Memory (kB)	32	256
EEPROM (kB)	1	4
SRAM (kB)	2	8
Voltage Level	5V	5V
Digital I/O Pins	14	54
Digital I/O with PWM Pins	6	15
Analog Pins	6	16
USB Connectivity	Standard A/B USB	Standard A/B USB
Shield Compatibility	Yes	Yes
Ethernet/Wi-FI/Bluetooth	No (a Shield/module can enable it)	No (a Shield/module can enable it)

Figure 2.10: Main features of the most popular Arduino boards

The other key feature of these boards is the Arduino proprietary Integrated Development Environment (IDE). It is very user-friendly and guided step-by-step, allowing the writing of code even for those who are new to programming. The editor is also able to compile and load the working and executable program on the board with a single click. There is generally no need to create Makefiles or run programs from the command line.

Now, if we immerse the Arduino in computer vision, it can still give us some satisfaction, but only if we work with static images or almost static images (a few

[9]www.arrow.com

pfs). Using the OpenCV library, it is still possible to identify objects, faces and people, but with a lot of effort and very few details and points detected and/or analysed. And this, as you can imagine, is unthinkable for the industrial world and operation in real time conditions. The reason is simple, computer vision and in particular image processing or even worse pose estimation require a lot of calculations that have to be performed in a very short time. Since there is no GPU in the Arduino architecture, it cannot perform the required calculations. The ATMega processors with which the boards are equipped are good for simple calculations to drive servo motors, home automation and sensors, but they are certainly not capable of processing the huge amount of parallel calculations required by industrial computer vision.

Raspberry Pi

The Raspberry, unlike the Arduino, includes a GPU in its hardware architecture, so it can be called a single-board computer in its own right. The main operating system is based on GNU/Linux such as Debian and Fedora but other operating systems are also supported including Windows 10 and Android. Thanks to its affordable price, which can vary depending on the amount of RAM required, this device is widely used in industry and is also within reach of less experienced users or simply electronics/automation enthusiasts. And thanks to the latter, you can find a myriad of guides, documentation and projects with related codes on the web.

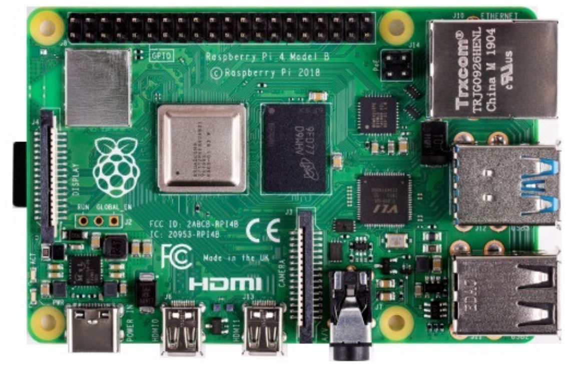


Figure 2.11: Raspberry Pi 4 Model B board

Among the main features are:

Price	38€/48€/58€/75€
Processor	1.5 GHz 64-bit quad-core ARM Cortex-A72 (ARMv8)
GPU	Broadcom VideoCore VI @ 500 MHz
SDRAM	1GB/2GB/4GB/8GB shared with GPU
Dimension	85mm x 56mm x 19.5mm, 46g
Voltage supply	5V via USB-C o via GPIO
Storage	microSD up to 64GB
USB ports	2x USB 2.0 and 2x USB 3.0
Networking	2.4GHz and 5GHz 802.11b/g/n/ac wireless LAN
Bluetooth	Bluetooth 5.0, Bluetooth Low Energy (BLE)
DIgital I/O Pins	40-pin GPIO header

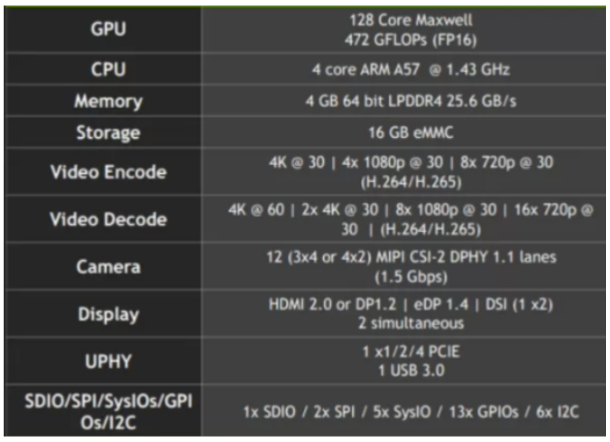
Despite the fact that it is an exceptional card for the vast majority of applications, it is still unable to guarantee a real time condition in human tracking and pose estimation.

After consulting various benchmarks[10], blogs[11], guides[12] and papers [7] we can conclude that the Raspberry Pi 4 is able to process video streams through computer vision libraries and frameworks such as OpenPose, TensorFlow Lite and Caffe with performance that can reach a maximum of **2-3 fps**. However, the result does not meet the processing speed required by the company, which must be above 5fps to meet the customer's specifications.

[10]www.arnabkumardas.com/platforms/nvidia/nvidia-jetson-nano-review-and-benchmark/
[11]aallan.medium.com/benchmarking-edge-computing-ce3f13942245
[12]qengineering.eu/opencv-c-examples-on-raspberry-pi.html

NVIDIA Jetson

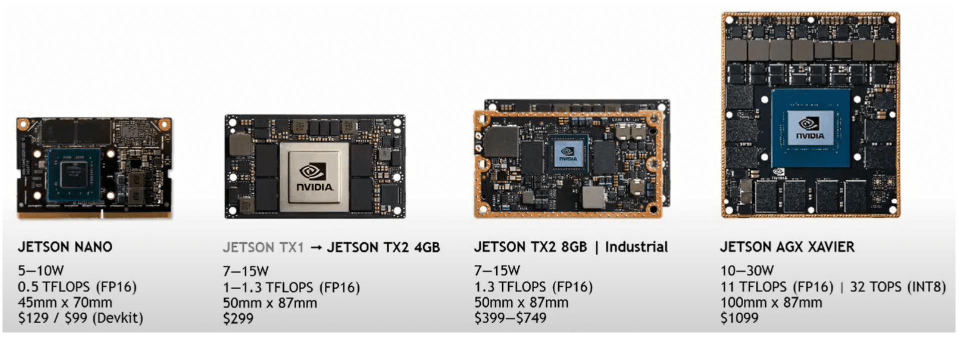
After further research, we found several benchmarks[13] and documentation [1] regarding the tremendous performance that NVIDIA's Jetson family of single boards were capable of delivering. It is no coincidence that NVIDIA is also the leading company in the GPU industry and has been dominating the market for several years with cutting-edge technology. For simplicity's sake, we only list the board's data below, as we will describe the device in more detail in Chapter 3.



GPU	128 Core Maxwell 472 GFLOPs (FP16)
CPU	4 core ARM A57 @ 1.43 GHz
Memory	4 GB 64 bit LPDDR4 25.6 GB/s
Storage	16 GB eMMC
Video Encode	4K @ 30 | 4x 1080p @ 30 | 8x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 16x 720p @ 30 | (H.264/H.265)
Camera	12 (3x4 or 4x2) MIPI CSI-2 DPHY 1.1 lanes (1.5 Gbps)
Display	HDMI 2.0 or DP1.2 | eDP 1.4 | DSI (1 x2) 2 simultaneous
UPHY	1 x1/2/4 PCIE 1 USB 3.0
SDIO/SPI/SysIOs/GPIOs/I2C	1x SDIO / 2x SPI / 5x SysIO / 13x GPIOs / 6x I2C

Figure 2.12: Jetson Nano main features

The hardware we selected is the first in a series of modules designed by NVIDIA and is the most modest in terms of cost and performance but still meets our requirements. As higher computational power is demanded the price increases until we arrive at the most powerful module of all which is the Jetson AGX Xavier that will be used for fully autonomous machines unlike its predecessors more suitable for the on-edge role.



JETSON NANO	JETSON TX1 → JETSON TX2 4GB	JETSON TX2 8GB | Industrial	JETSON AGX XAVIER
5—10W	7—15W	7—15W	10—30W
0.5 TFLOPS (FP16)	1—1.3 TFLOPS (FP16)	1.3 TFLOPS (FP16)	11 TFLOPS (FP16) | 32 TOPS (INT8)
45mm x 70mm	50mm x 87mm	50mm x 87mm	100mm x 87mm
$129 / $99 (Devkit)	$299	$399—$749	$1099

Figure 2.13: Jetson family boards

[13]developer.nvidia.com/blog/jetson-nano-ai-computing/

AMD Ryzen Embedded

Of course, we couldn't fail to mention the other leader in the video card and processors sector, AMD, which with its 27% market share shares shares a monopoly with its direct competitor NVIDIA with 73% [14]. AMD is competing for the on-edge and miniPC market share with the Ryzen Embedded family of 64-bit multi-core x86 embedded microprocessors introduced in early 2018 with the V1000 series based on the Zen microarchitecture. The Ryzen Embedded family is a low-power variant of the Ryzen line that primarily targets graphics-driven embedded devices such as medical imaging, industrial systems, digital gaming and thin clients.

Ryzen Embedded V1000 integrates Zen CPUs and Vega GPUs on a single die, offering up to 4 cores / 8 threads and up to 11 GPU Compute Units for a throughput of up to 3.6 TFLOPS at 16-bit computation. With a price tag expected to be around 200USD makes them a serious contender to the similar NVIDIA Jetson TX2 Series already on market with the 1.33 TFLOPS as AI Performance claimed on their website [15] at a known price starting at 249USD. Just for comparison, the Jetson Nano has a computing capacity of 471.6 GFLOPS with the same FP16 format. Key characteristics include:

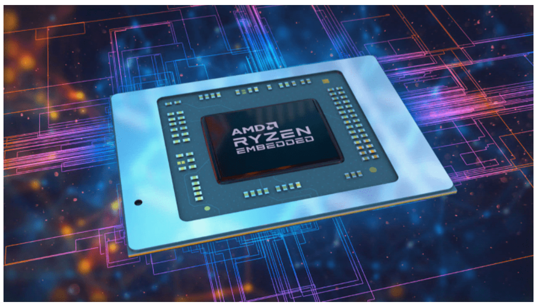


Figure 2.14: AMD Ryzen Embedded microprocessor serie 1000

Memory	Dual-channel 64-bit DDR4-3200 or DDR-2400 w/ ECC, up to 32 GiB
I/O	up to 16 PCIe lanes, 2 SATA ports, 6 USB ports; 2 Gigabit Ethernet ports
TDP	6W-45 W (with cTDP-up and cTDP-down options)
ISA	Everything up to AVX2
Tech	Precision Boost, 2-way SMT, AMD-Vi, AMD-V, SME, and SEV
L3$	4 MiB

In addition, was launched in November 2020, the processors of the V2000 embedded series based on the Zen 2 microarchitecture in a single monolithic die fabricated on a TSMC 7 nm process, doubling the performance per Watt and core count over the prior generation, and increasing IPC by 15%. These SoCs come in either hexa-core or octa-core models with SMT and a Vega integrated graphics processor with 6 or 7 compute units in package FP6. The integrated Radeon graphics have also been improved by 40 per cent.

[14]businessquant.com/global-gpu-market-share
[15]https://developer.nvidia.com/embedded/jetson-modules

The Ryzen V2000 offers configurations of up to 8 cores/16 threads, 20 PCI-E 3.0 lines, supports up to 4K displays, and has a configurable TDP of 10 to 54 watts.

Processors defined as 'embedded' are designed for integration into Edge systems, Mini PCs and Thin Clients. The AMD Ryzen Embedded R1000 is available worldwide since 2019 and is already supported by numerous hardware and software companies including Advantech, Alphainfo, ASRock Industrial, Axiomtech, DFI, iBase, Kontron, MEN, Mentor, Sapphire[16], zSpace, Nuvo-2700DS Series[17] and more.

We did not select the AMD device because of the lack of documentation and literature on human tracking and identification as well as computer vision technology, and the risk of taking too long to finish the job. But this does not preclude the fact that in the next few years it could be a valid alternative to the Jetson Nano and its heirs.

Google Coral Dev board

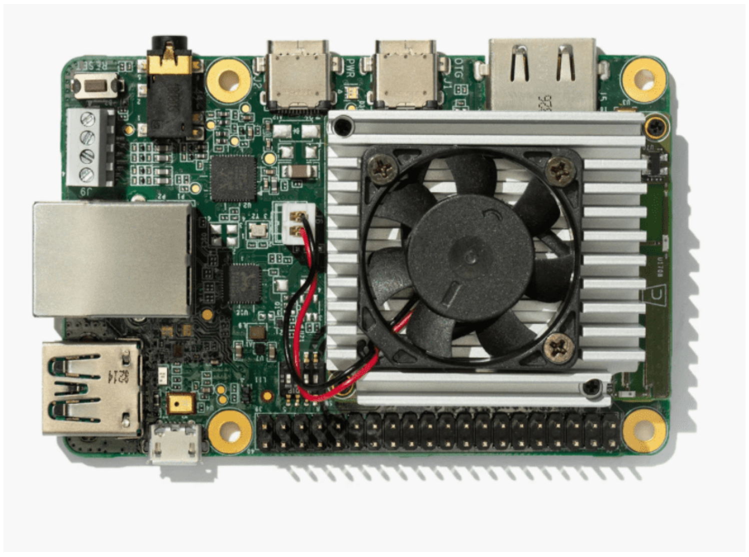


Figure 2.15: Google Coral Dev board

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. 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 (Tensor Processing Unit) coprocessor. The Edge TPU is a small ASIC designed by Google that provides high performance ML inferencing with a low power cost. The on-board Edge TPU coprocessor is capable of performing

[16] www.anandtech.com/show/15549/sapphire-announces-new-4x4-amd-ryzen-embedded-motherboards

[17] www.neousys-tech.com/en/

4 trillion operations (tera-operations) per second (TOPS), using 0.5 watts for each TOPS (2 TOPS per watt). 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. We summarise the main features in the table below:

CPU	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)
GPU	Integrated GC7000 Lite Graphics
RAM	1 GB / 4 GB LPDDR4
Price	130 USD / 170 USD
Flash memory	8 GB eMMC, MicroSD slot
Wireless	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz); Bluetooth 4.2
USB	Type-C OTG; Type-C power; Type-A 3.0 host; Micro-B serial console
LAN	Gigabit Ethernet port
GPIO	3.3V power rail; 40-255Ω programmable impedance; 82 mA max current
Power	5V DC (USB Type-C)
Dimensions	88 mm x 60 mm x 24mm
ML Accelerator	Google Edge TPU coprocessor:4 TOPS (int8); 2 TOPS per watt

We conclude by saying that the Google Coral Dev board remains much less used and documented in the AI and computer vision world than the Jetson Nano, but despite its higher price it is more suitable for ML (Machine Learning) applications and ensures better performance in terms of inference. If, on the other hand, we move into the world of computer vision and image processing, the most suitable board is certainly the Jetson Nano with its Maxwell GPU with 128 CUDA cores.

As far as the industrial world is concerned, Jetson is more suited to small and medium-sized enterprises and start-ups, while the Google Dev board is more suited to medium-sized and large companies, due to the fact that it is much more restrictive in terms of available software and for the reasons mentioned above.

Google Coral Stick

Now let's get back to the hardware accelerators we mentioned in the previous paragraphs (i.e edge TPU). The first of these is also made by Google Coral and is a USB stick that brings machine learning inferencing to existing systems. Works with Linux, Mac, and Windows systems by simply plugging the memory stick into the USB port.

Figure 2.16: Google Coral USB Accelerator

Technical specifications

ML accelerator	Google Edge TPU coprocessor:4 TOPS (int8); 2 TOPS per watt
Connector	USB 3.0 Type-C (data/power) and USB2.0 compatible
Dimensions	65 mm x 30 mm
Price	starting from $59.99

The on-board Edge TPU coprocessor is capable of performing 4 trillion operations (tera-operations) per second (TOPS), using 0.5 watts for each TOPS (2 TOPS per watt). 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. To confirm the above statements, the bench-marking performed by [18] actually shows

[18]blog.usejournal.com

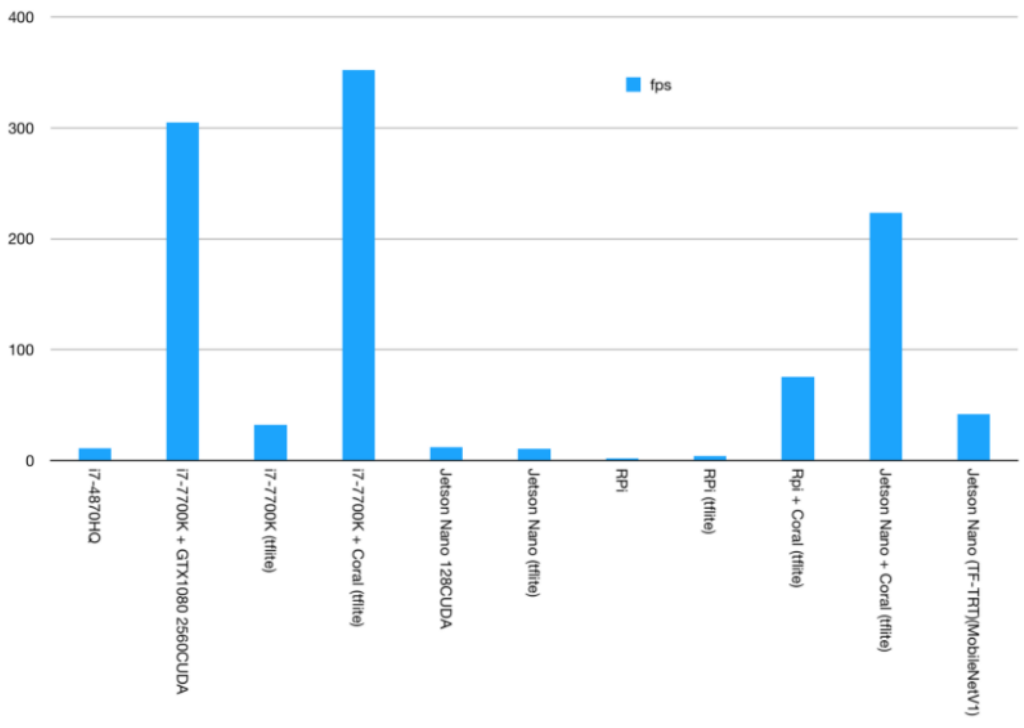


Figure 2.17: Inference increasing benchmarks thanks to Google Coral USB stick

that inference in terms of fps increases dramatically with the addition of the Google USB stick accelerator. The model used for the test is the MobileNetV2 as classifier, pre-trained on ImageNet dataset. The benchmarks are performed on the I7 processor with GTX1080 and Coral USB stick, Jetson Nano with and without Coral USB stick and finally the Raspberry PI with and without the Coral HW accelerator. We did not use the accelerator basically for two reasons: cost and scope. In our project we are dealing with image processing but not with image classification but with human tracking and pose estimation.

Intel Neural Computer Stick 2

It should not surprise us that the other player in hardware acceleration is Intel Corp, the market leader in microprocessors. Intel® Neural Compute Stick 2 is powered by the Intel Movidius™ X VPU to deliver industry leading performance, wattage, and power.

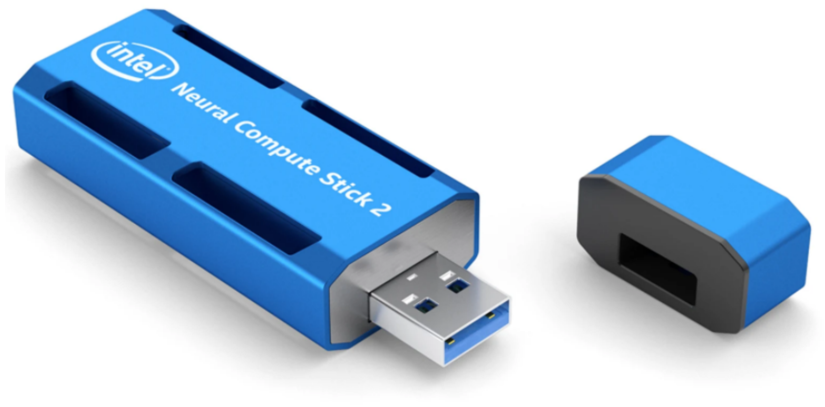


Figure 2.18: Intel NCS2

NCS 2 speeds up the development of deep neural networks, providing up to 8 times the performance of the first version, giving developers the ability to test, optimise and prototype more advanced deep neural networks. Intel talks about a set of 16 programmable SHAVE (Streaming Hybrid Architecture Vector Engine) cores representing a 33 per cent increase over the previous generation, supported by a memory interconnect with high throughput.

Intel assures that simply plugging the NCS 2 into a laptop will have AI and computer vision applications ready in minutes. NCS 2 connects to a standard USB 3.0 port and requires no additional hardware, allowing users to convert and deploy PC models on a wide range of devices natively and without Internet or cloud connectivity.

Intel's NCS 2 is also supported by the OpenVINO toolkit, which offers deep learning, computer vision and hardware acceleration for the development of applications with human-like vision capabilities. The combination of the NCS 2 with Intel's distribution of the OpenVINO toolkit enables a rapid development and deployment cycle: from prototyping DNNs trained on the Compute Stick to simply porting DNNs to an embedded device based on Intel's Movidius VPU or a system requiring little or no code changes. The NCS 2 also supports the popular open-source software libraries Caffe and TensorFlow. The Neural Compute Stick 2 is available in Europe from RS Components and Mouser for around 85$.

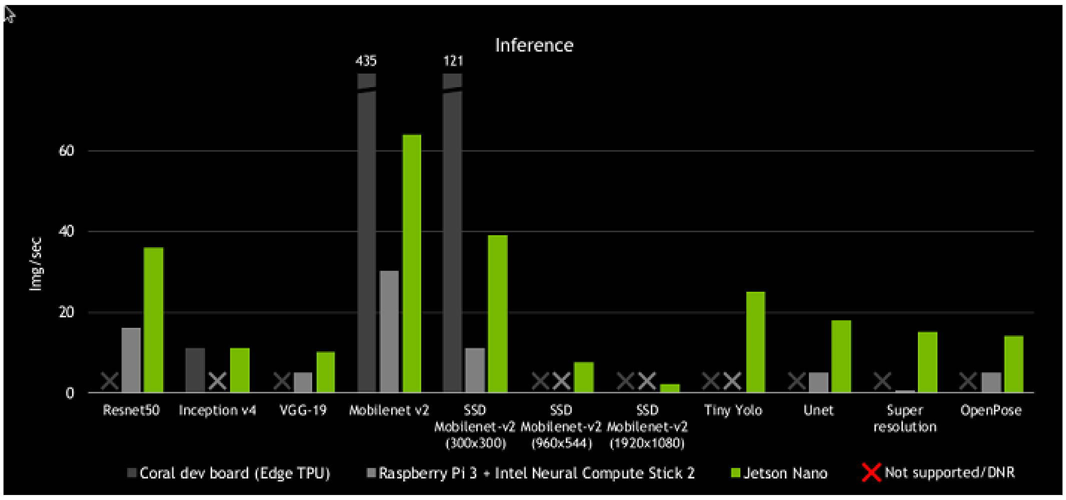


Figure 2.19: Inference increasing benchmarks due to NCS2 stick

As with the previous solution, we report these very interesting results comparing the Coral Dev board which already includes the accelerated Edge TPU, Raspberry Pi3 with the NCS2 and the Jetson Nano alone tested with various tools including OpenPose and Tiny YOLO-v3 on different image classification models such as MobileNet-v2.

2.4 Computer vision software, frameworks and tools

OpenCV

It is a free software library originally developed by Intel in its research centre in Russia in Niznij Novgorod. It was later maintained by Willow Garage and now by Itseez. The main programming language used to develop with this library is C++, but it is also possible to interface with C, Python and Java. In fact, to feed the frames to be processed to OpenPose, we use OpenCV through scripts in Python. It is so well documented that we have found everything we need to capture frames from a webcam or video file on their website.[19]. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.[20]

TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow was developed by the Google Brain team and made available on 9 November 2015, under the terms of the Apache 2.0 open source licence. TensorFlow is compatible with major 64-bit operating systems (Windows, Linux and Mac OS X) and Android. Although the official documentation initially spoke of limited hardware compatibility, the library can run on many types of CPUs and even GPUs, thanks to the support of languages such as CUDA or OpenCl. Furthermore, Google has designed and built an ASIC (Application specific integrated circuit) processor specifically for this language, called

[19]docs.opencv.org/4.5.2/dd/d43/tutorial_py_video_display.html
[20]https://opencv.org/about/

TPU (Tensor Processing Unit), which we have already discussed in the previous paragraphs.

CUDA

CUDA (Compute Unified Device Architecture) is a hardware architecture for parallel computing created by NVIDIA. Through the CUDA development environment, software programmers can write applications capable of parallel computing on NVIDIA video card GPUs that are of particular interest in the world of computer vision and human tracking. CUDA accelerates applications across a wide range of domains from image processing, to deep learning, numerical analytics and computational science.

SciPy and NumPy

SciPy is an open source library of algorithms and mathematical tools for the Python programming language. It contains modules for optimization, linear algebra, integration, special functions, FFT, signal and image processing, ODE solvers, and other tools common in science and engineering.

NumPy, on the other hand, is an open source library for the Python programming language that adds support for large matrices and multidimensional arrays along with a large collection of high-level mathematical functions to efficiently operate on these data structures.

Caffe

CAFFE (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework, originally developed at University of California, Berkeley. It is open source[21], under a BSD license. It is written in C++, with a Python interface[22]. Caffe supports many different types of deep learning architectures geared towards image classification and image segmentation. It supports CNN, RCNN, LSTM and fully connected neural network designs. Caffe supports GPU and CPU-based acceleration computational kernel libraries such as NVIDIA cuDNN and Intel MKL. Caffe is being used in academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia.

PyTorch and Torchvision

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing. Although the Python interface is more polished and the primary focus of development,

[21]https://github.com/BVLC/caffe
[22]https://caffe.berkeleyvision.org/

PyTorch also has a C++ interface. A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, HuggingFace's Transformers, PyTorch Lightning and Catalyst. PyTorch provides two high-level features: Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU) and Deep neural networks built on a type-based automatic differentiation system. It is free and open-source software released under the Modified BSD license.

TorchVision is a computer vision library for manipulating images. It contains utility functions for processing images so that they can be fed into neural networks. It also houses popular image datasets, deep CNN model architectures along with pretrained models.

OpenPose

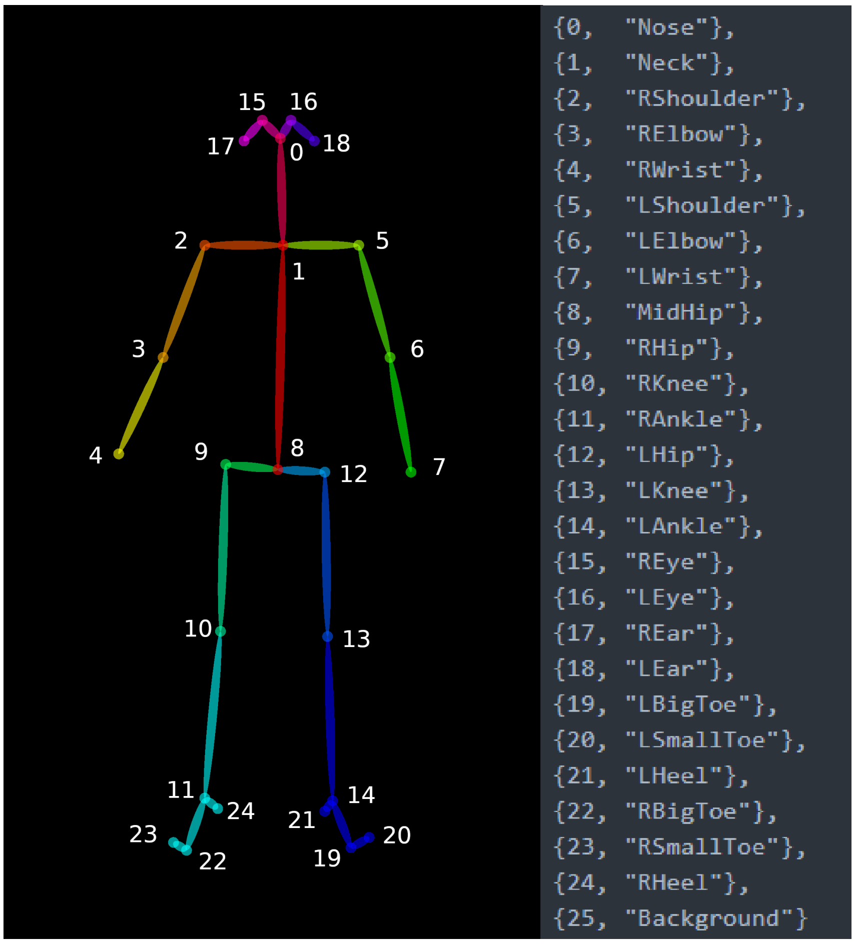
OpenPose[2] has represented the first real-time multi-person system to jointly detect human body, hand[6], facial, and foot keypoints (in total 135 keypoints) on single images.

Having limited hardware resources and working at distances allowed by a classic surveillance camera, it was decided to adopt the twenty-five keypoints model BODY_25[23]. So hands and face are not taken into account because they require a significant computational effort and the latter is not justified because at distances close to 10m from the video source additional points are not detected.

To confirm what has been said above we report the official requirements of OpenPose taken from the official repository on GitHub. An NVIDIA GPU with at least 16GB of memory is required to run on the full version. For the simplified model BODY_25 at least 10.5GB of memory is required and the GPU's of reference are TITAN X, QUADRO P100, QUADRO V100. From what is stated on the repository it runs at about 2 FPS on a Titan X for BODY_25. As we will illustrate in the following paragraphs has been used the demo version of OpenPose making small changes to the scripts already present in Python language to adapt them to our needs.

OpenPose is freely available for free non-commercial use, and may be redistributed under these conditions.

[23]https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/02_output.md



```
{0,  "Nose"},
{1,  "Neck"},
{2,  "RShoulder"},
{3,  "RElbow"},
{4,  "RWrist"},
{5,  "LShoulder"},
{6,  "LElbow"},
{7,  "LWrist"},
{8,  "MidHip"},
{9,  "RHip"},
{10, "RKnee"},
{11, "RAnkle"},
{12, "LHip"},
{13, "LKnee"},
{14, "LAnkle"},
{15, "REye"},
{16, "LEye"},
{17, "REar"},
{18, "LEar"},
{19, "LBigToe"},
{20, "LSmallToe"},
{21, "LHeel"},
{22, "RBigToe"},
{23, "RSmallToe"},
{24, "RHeel"},
{25, "Background"}
```

Figure 2.20: BODY_25 model of the body stick used by OpenPose.

TensorRT

TensorRT is Nvidia software solution for generating optimized models for production deployment of Deep Learning Models.

With TensorRT, you can optimize neural network models trained in all major frameworks, calibrate for lower precision with high accuracy, and deploy to hyperscale data centers, embedded, or automotive product platforms.

TensorRT is built on CUDA®, NVIDIA's parallel programming model, and enables you to optimize inference leveraging libraries, development tools, and technologies in CUDA-X™ for artificial intelligence, autonomous machines, high-performance computing, and graphics.

TensorRT provides INT8 and FP16 optimizations for production deployments of deep learning inference applications such as video streaming, speech recognition, recommendation, fraud detection, and natural language processing. Reduced precision inference significantly reduces application latency, which is a requirement for many real-time services, as well as autonomous and embedded applications.

With TensorRT, developers can focus on creating novel AI-powered applications rather than performance tuning for inference deployment.

YOLO

You Only Look Once (YOLO) is a state-of-the-art, real-time object detection system[5]. Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections. YOLO uses a totally different approach. A single neural network is applied to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

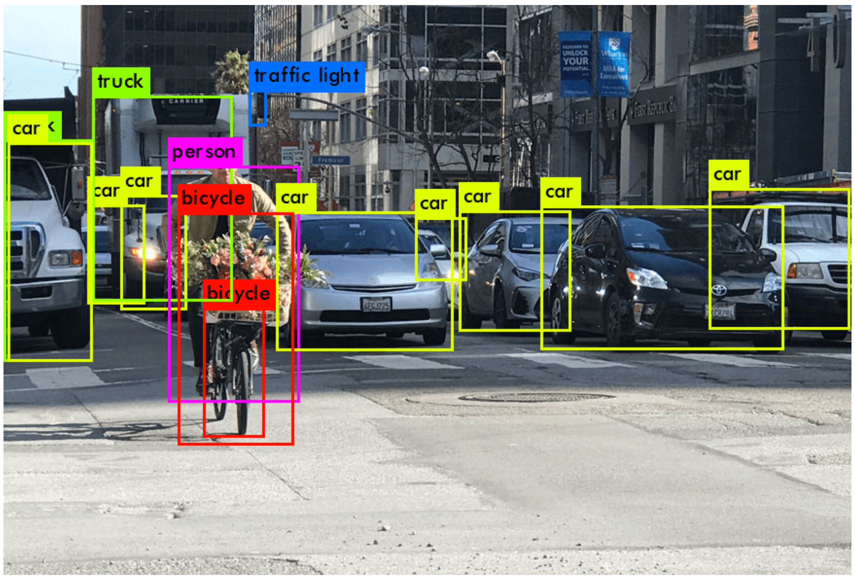


Figure 2.21: YOLO object detection.

YOLO has 24 convolutional layers followed by 2 fully connected layers (FC). Some convolution layers use 1×1 reduction layers alternatively to reduce the depth of the features maps. For the last convolution layer, it outputs a tensor with shape (7, 7, 1024). The tensor is then flattened. Using 2 fully connected layers as a form of linear regression, it outputs $7 \times 7 \times 30$ parameters and then reshapes to (7, 7, 30), i.e. 2 boundary box predictions per location. A faster but less accurate version of YOLO, called Fast YOLO, uses only 9 convolutional layers with shallower feature maps.

To pursue the purpose of the work, it was not enough to identify the people on the monitor but it was also necessary to track them over time and assign them

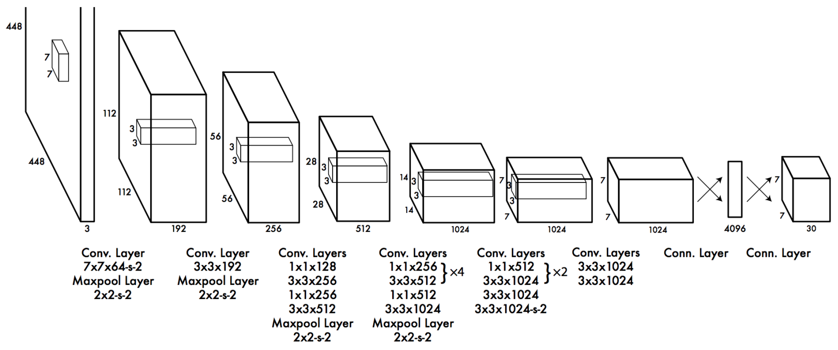


Figure 2.22: YOLO network structure.

a unique id. For this reason it was decided to implement an additional tool that would use YOLO as a detector and track the object or person. Deep-SORT[9] was the most suitable solution. As you can see in the figure below is able to identify a pedestrian in the street, assign him an identifier and track his movements over time thanks to complex techniques such as Kalman Filter and Ungarian Algotirthm.

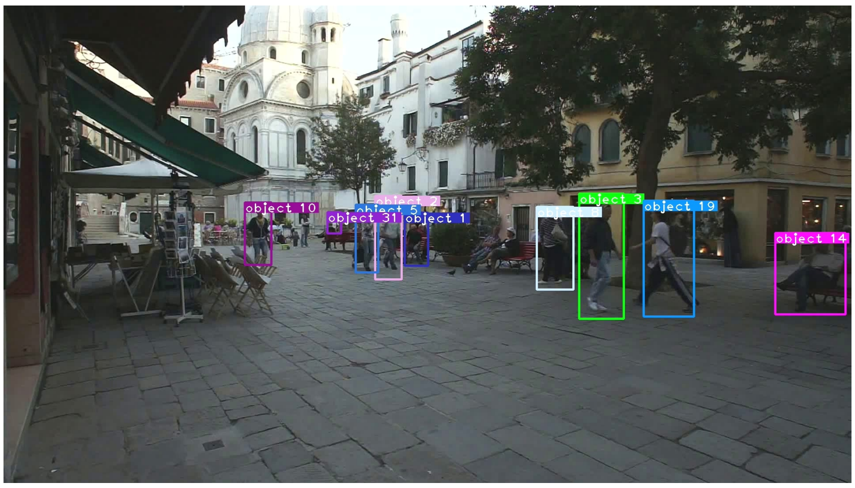


Figure 2.23: DeepSort that uses YOLOv3 in order to detect and track the pedestrians.

Chapter 3

On-edge architecture

The 'on-edge' device must be able to process information coming from a video stream of one or more surveillance cameras in the shortest time possible. By doing this, large video streams from the cameras to the server are avoided by relieving data traffic on the customer's local network. The only information that will be sent to the central processor will be spatial coordinates of each subject in the video of very small size (a few bytes). In addition, it avoids all problems related to the transfer of sensitive data and privacy due to video streams and confining them to remain in the domain of one or more cameras.

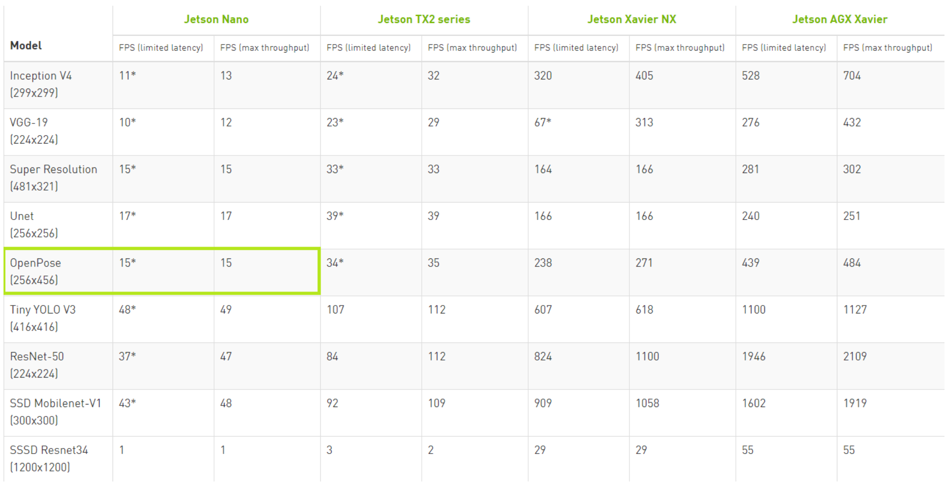
The device is used to identify and track objects and people in closed environments such as warehouses, depots and large distributions where the GPS signal is poor or absent altogether. By using a simple surveillance camera we avoid making people wear special tags in order to identify them with technologies such as Bluetooth, WI-FI and UWB that involve some additional cost.

The three fundamental requirements of the selected device can be summarized as follows: cheap, compact and powerful.

The cost was reduced to ensure competitiveness. The size of the device had to remain compact so that it could be installed in prearranged locations with very limited space adjacent to the surveillance cameras.

The performance that the electronic device had to guarantee in order to be able to process the amount of data coming from the video sources through a network of machine learning and deep learning software on board required a not indifferent calculation power.

At the end of the research and analysis of various benchmarks present in the literature was selected the **Jetson Nano** by NVIDIA Corporation. It was the best trade-off in terms of cost-performance as you can see in the Figure3.1 based on the software we intended to use which is OpenPose. Moreover it was suitable for installations next to surveillance cameras on walls and ceilings.



Model	Jetson Nano		Jetson TX2 series		Jetson Xavier NX		Jetson AGX Xavier	
	FPS (limited latency)	FPS (max throughput)	FPS (limited latency)	FPS (max throughput)	FPS (limited latency)	FPS (max throughput)	FPS (limited latency)	FPS (max throughput)
Inception V4 (299x299)	11*	13	24*	32	320	405	528	704
VGG-19 (224x224)	10*	12	23*	29	67*	313	276	432
Super Resolution (481x321)	15*	15	33*	33	164	166	281	302
Unet (256x256)	17*	17	39*	39	166	166	240	251
OpenPose (256x456)	15*	15	34*	35	238	271	439	484
Tiny YOLO V3 (416x416)	48*	49	107	112	607	618	1100	1127
ResNet-50 (224x224)	37*	47	84	112	824	1100	1946	2109
SSD Mobilenet-V1 (300x300)	43*	48	92	109	909	1058	1602	1919
SSSD Resnet34 (1200x1200)	1	1	3	2	29	29	55	55

Figure 3.1: NVIDIA Jetson benchmarks

3.1 Jetson Nano board

The Jetson Nano is a tool designed and engineered for AI (Artificial Intelligence) and small robotics applications with low power consumption (5-10W). Being equipped with optimization software and libraries for deep learning, computer vision, GPU computing and multimedia processing such as NVIDIA CUDA and cuDNN it is able to provide excellent performance in parallel computing and run most of the modern neural networks. All this is possible thanks to the deep knowledge of the GPU architecture (Maxwell) designed by NVIDIA itself that implements hardware acceleration libraries to bring out the best possible performance. The more expensive 4GB RAM version has been selected based on OpenPose requirements (at least 2GB) so we could work with some margin.

The first step to be able to work on the card is to load the operating system conveniently available on the site [1] as an SD card image named Jetpack. In fact the Jetson is equipped with an SD card slot where to put a 32GB microSD formatted according to the procedure provided by the manufacturer. Once prepared the latter the image has to be written on the SD card with the operating system. This latter includes all the necessary libraries in addition to software NVIDIA needed for optimization and hardware acceleration. Among these the most important key features for the OS support are: NVIDIA L4T bootloader, Linux kernel, necessary firmwares, NVIDIA drivers, sample filesystem. TensorRT which assures high performance deep learning inference runtime for image classification, segmentation, and object detection neural networks. CUDA Deep Neural Network library which provides high-performance primitives for deep learning frameworks such as forward and backward convolution, pooling, normalization, and activation layers. CUDA development environment for

[1] https://developer.nvidia.com

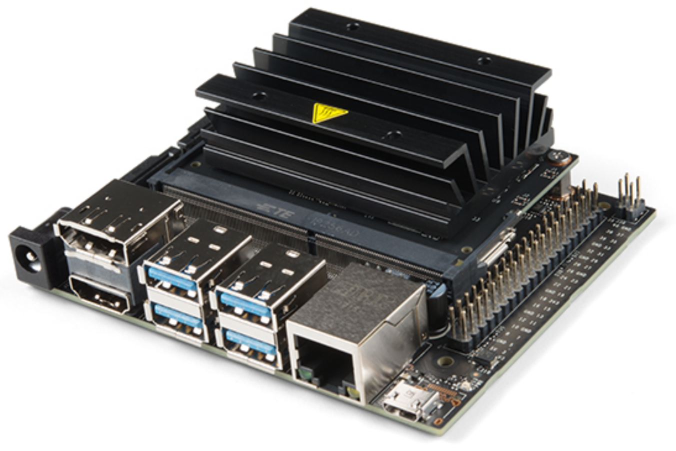


Figure 3.2: NVIDIA Jetson Nano 4GB board

C and C++ developers building GPU-accelerated applications for CUDA hardware architecture. Multimedial and low level APIs for flexible application development for camera applications, video decode, encode, format conversion and scaling functionality. VisionWorks, OpenCV and Vision Programming Interface libraries for computer vision, image processing and machine learning. And finally supported SDKs and Tools like Deepstream SDK and PowerEstimator.

Following several tests and installation attempts of the Jetpack 4.4 and Jetpack 4.4.1 versions, we found several incompatibilities in terms of software version between the various frameworks already present and those to be installed later. Going backwards, we have therefore focused on the Jetpack 4.3 version, which has given us fewer problems than its predecessors. To achieve our goal, we followed the guides on the [2] blog, which was very useful and helpful.

We then format the microSD (at least 16GB recommended) with the SD Memory Card Formatter tool. If for some reason there is already an operating system on it, the card will not be recognised by the formatting wizard. You have to delete the existing version manually with the following procedure:

Start→Disk management→Disk corresponding to the micro SD used (32GB in

[2]https://spyjetson.blogspot.com

our case with 29.15GB effective)→Delete volume→Create partition

After that, the PC with which we are going to format the microSD is able to recognise a new disk drive. Download the JetPack 4.3 image from [3] and write it to the microSD using the Etcher tool. Once this is done, insert the microSD into the microSD slot on the Jetson under the black heat sink and connect all the peripheral devices as shown in the figure above 3.3.

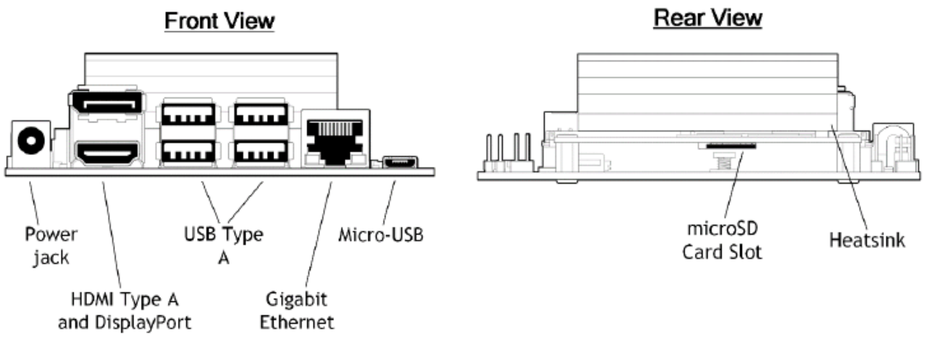


Figure 3.3: NVIDIA Jetson Nano leteral views

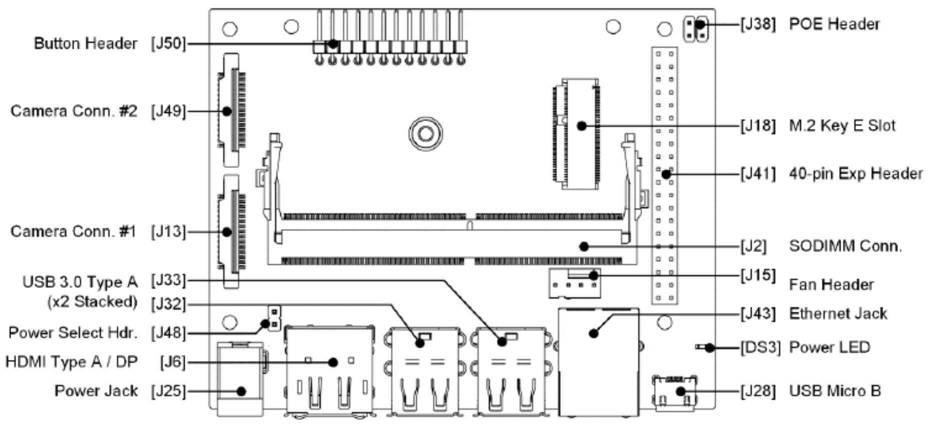
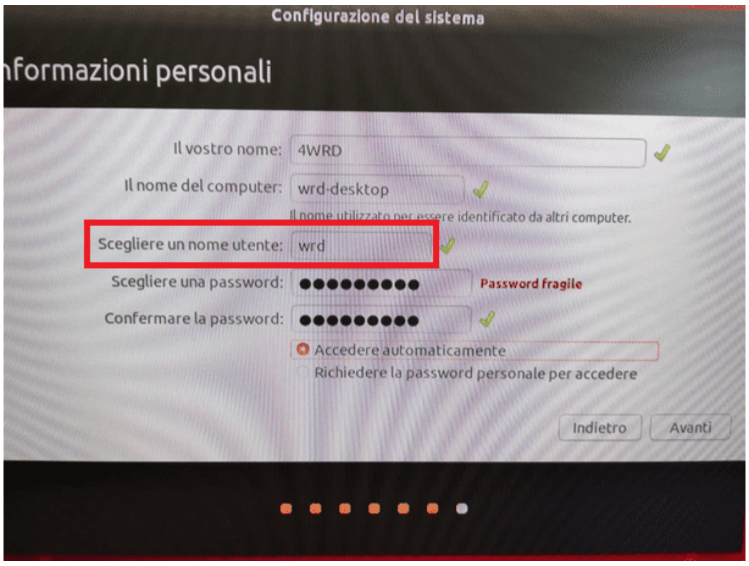


Figure 3.4: NVIDIA Jetson Nano top view

We are going to power the Jetson with a 5V and 4A power supply on the RS(J25) jack, as in the picture, and this must guarantee the effective supply of 5V and not less, otherwise the board won't even turn on. For this reason, NVIDIA recommends a specific power supply for this application that takes into account the voltage drop across the power supply cable (about 0.2V). To change the power supply mode from

[3]https://developer.nvidia.com/jetpack-43-archive

microUSB(J28) to DC(J25), you must first insert the jumper on J48 to short the pins. Respectively we will have J48 with open pins - microUSB power and J48 shorted - DC power. Finally, we connect the remaining peripheral devices including monitor (HDMI), mouse and keyboard (USB 3.0), Ethernet cable (LAN) and proceed to power-up.


Configurazione del sistema

nformazioni personali

Il vostro nome: 4WRD

Il nome del computer: wrd-desktop

Il nome utilizzato per essere identificato da altri computer.

Scegliere un nome utente: wrd

Scegliere una password: ●●●●●●●●● Password fragile

Confermare la password: ●●●●●●●●●

[✓ Accedere automaticamente]

Richiedere la password personale per accedere

Figure 3.5: First boot screen - system configuration

The Jetson board doesn't have an on/off button, so you just plug in the power supply and it goes. As for the shutdown, we will do it directly from the onboard operating system (Linux). At the first power on we have to enter some data that will identify our device 3.5.

Particular attention should be paid to the "user name" entry, which will identify the card when it is connected to other PCs. In particular, when you want to use the "Headless Mode" documented in the User Guide. This allows you to configure Jetson Nano with an external PC using only the microUSB cable (removing the jumper from J48), thus avoiding the need to connect the power supply, HDMI, mouse and keyboard thanks to the SSH (Secure SHell) connection protocol. The above mentioned mode could prove very useful, for example, at the time of the first installation or subsequent debugging at the customer's premises.

A few intermediate steps are required before proceeding with the actual installation. First of all, since the operating memory is limited to 4GB, a trick is used to "extend" it and free up part of it using the mass memory. We go to uninstall applications and software not useful for our purposes as Libre Office on Ubuntu. Finally we're going to interface with the dependencies that must be respected in order for applications to work properly. We go to update the version of CMake to have a correct compilation of OpenPose.

3.1.1 Memory swap expansion

When dealing with computer vision, RAM memory runs out at lightning speed. The primary function of swap space is to substitute disk space for RAM memory when real RAM fills up and more space is needed.

The kernel uses a memory management program that detects blocks, aka pages, of memory in which the contents have not been used recently. The memory management program swaps enough of these relatively infrequently used pages of memory out to a special partition on the hard drive specifically designated for “paging,” or swapping. This frees up RAM and makes room for more data to be entered into your spreadsheet. Those pages of memory swapped out to the hard drive are tracked by the kernel’s memory management code and can be paged back into RAM if they are needed.

The total amount of memory in a Linux computer is the RAM plus swap space and is referred to as virtual memory.

NVIDIA recommended swap space of 4GB for a system with 4GB of RAM. Before starting, in order to avoid having to type the ”sudo” command every time, from the terminal type

```
sudo –
```

followed by the password to enter the root account mode. First of all we type

```
free –m
```

in order to check the virtual memory swap and should give us 1978MB (2GB) so let’s expand it. Type in the following commands one by one

```
systemctl disable nvzramconfig

fallocate –l 4G /mnt/4GB.swap

chmod 600 /mnt/4GB.swap

mkswap /mnt/4GB.swap

echo ”/mnt/4GB.swap swap swap defaults 0 0” >> /etc/fstab
```

and restart the system to make the changes effective. We verify the result again with

```
free –m
```

and this time we should get 4GB.

3.1.2 Libre Office removal

We remove everything that takes up memory and is not essential for our purpose. Among these we have Libre Office, which we uninstall with

```
apt-get remove --purge libreoffice*

apt clean

apt-get autoremove
```

3.1.3 Cmake library update

We must check the version of Cmake, which must be higher than 3.12.2

```
cmake --version
```

and find 3.10.2 so it needs to be updated.

```
apt-get install libssl-dev libcurl4-openssl-dev

apt-get remove cmake

cd /usr/local/src

wget https://github.com/Kitware/CMake/releases/download/v3
    .17.2/cmake-3.17.2.tar.gz

tar -xvzf cmake-3.17.2.tar.gz

cd cmake-3.17.2./bootstrap

make -j4

make install

service ssh restart
```

3.2 Software for human tracking and identification

Computer vision software is required to accomplish the recognition and subsequent identification of people.

Detection and tracking are possible thanks to state-of-the-art tools made available on the web and documented by the scientific community. Among the best known we have OpenPose, TrtPose, AlphaPose, wrnchAI, YOLO and DeepStream. In this thesis work OpenPose, TrtPose and TrackLite have been used for the detection while for the identification after having briefly tested TrackLite we decided to create our own algorithm because the results were not satisfactory.

It was possible to run OpenPose on the Jetson Nano but the output performance was not as desired. Unfortunately, it was not possible to reach 5fps as requested by the research and development department and that's why we continued with the search for available software.

After further research we discovered the existence of a tool made by NVIDIA itself specifically for our Jetson Nano card and optimized to work on the hardware architecture of the latter. The above application is TrtPose and is an analogue of OpenPose only lighter from the point of view of computation. In fact, after several tests and trials on images, video files and webcams we finally managed to get the desired results. These were in line with what NVIDIA claimed in their benchmarks and articles. They were able to identify people's bodysticks by processing frames in real time at 8-10 fps. For a lighter software, unfortunately, there was a price to pay for the performance at long distances. Compared to OpenPose, it could not identify people more than 10-12 meters away from the camera.

In addition, we still had to realize what would have been the real identification with the assignment of a progressive id for each subject appeared on the screen.

Since the company and its development team were using YOLOv3 on a fixed workstation with large hardware resources available to realize the identification, we were asked to implement the same application on the on-edge workstation with limited computing capacity.

It was interesting to see the differences between the two in order to be able to make conclusions about performance and subsequent solutions to propose to the final customer.

After further research in literature and in the online computer vision world we found a very interesting application called TrackLite [4]. This tool was capable of processing up to 2fps video files and streams from the test webcam in the lab. Although the output was below the desired threshold (5fps) was a very interesting solution considering the fact that can re-identify a person even after leaving the screen for a small interval of time and is a heavy operation from the computational point of view.

Once the person has been recognized through their body stick via OpenPose, TrtPose or similar the next step is identification. This is done by writing an ad hoc algorithm based on the lengths of body segments obtained from the various body sticks which we will discuss in the following paragraphs.

Below we report the installation procedures of the various applications and the outputs obtained with the demo versions used in the test phase.

3.2.1 OpenPose

We need to enter the directory in which we are going to install OpenPose

```
cd /usr/local/src
```

and then copy the contents of the existing repository on GitHub to our directory

[4]https://github.com/Stephenfang51/tracklite

```
git clone https://github.com/CMU-Perceptual-Computing-Lab/
    openpose
```

once this is done, we enter the `/openpose` directory

```
cd /openpose
```

and run the command

```
bash ./scripts/ubuntu/install_deps.sh
```

which may generate a version compatibility error such as ”python setup.py egg-info” failed. However, this should not cause any major problems, so we continue with the installation, which will be successful. We now install the version of OpenPose downloaded above and compile it with Cmake from the build directory we are going to create

```
mkdir build

cd build

cmake -D BUILD_EXAMPLES=ON  -D BUILD_PYTHON=ON  -D
    USE_OPENCV=ON  ..

make -j4

make install

cd python

make -j4
```

We must also install this additional canberra-gtk module

```
sudo apt-get install libcanberra-gtk-module
```

Finally, to check if the installation was successful, type the following commands

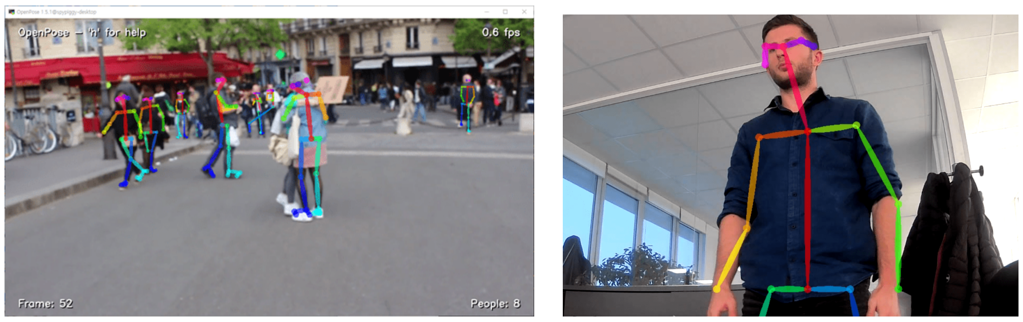
```
cd /usr/local/src/openpose
```

```
./build/examples/openpose/openpose.bin --video ./examples/
    media/video.avi --net_resolution 128x64
```

where the `--net_resolution` command is used to reduce the output resolution on the display to reduce the lag of the video stream. Another interesting command to check correct operation after connecting the webcam to the Jetson is the following

```
./build/examples/openpose/openpose.bin --net_resolution 128
    x64
```

and the output we would expect is



(a) (b)

Figure 3.6: OpenPose testing. (a) Demo video file. (b) Webcam test.

Finally, if we want to test the program on simple images we can also use

```
./build/examples/openpose/openpose.bin --image_dir examples/
    media/image_to_analyze
```

3.2.2 TrtPose

Before proceeding with the explanation, it is necessary to make a small premise. As already mentioned in the introduction, it was necessary to process the video stream coming from a video surveillance camera in real time, guaranteeing a certain number of frames per second (fps).

OpenPose is a very powerful tool for human pose estimation, but it requires a considerable effort in terms of performance. Once the tool was set up and while testing, we noticed that the performance was not exactly as reported in the NVIDIA bechmark tables (15fps). So we investigated the matter further, and after contacting NVIDIA we discovered that they were using their own custom tool optimised for compact (on-edge) devices such as the Jetson Nano.

Its name is TensorRT, and as reported on the NVIDIA® website, TensorRT™ is an SDK for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that deliv ers low latency and high throughput for deep learning inference applications. With TensorRT, you can optimize neural network models trained in all major frameworks, calibrate for lower precision with high accuracy, and deploy to hyperscale data centers, embedded, or automotive product platforms.

TensorRT is built on CUDA®, NVIDIA's parallel programming model, and enables you to optimize inference leveraging libraries, development tools, and technologies in CUDA-X™ for artificial intelligence, autonomous machines, high-performance computing, and graphics.

TensorRT provides INT8 and FP16 optimizations for production deployments of deep learning inference applications such as video streaming, speech recognition, recommendation, fraud detection, and natural language processing. Reduced precision inference significantly reduces application latency, which is a requirement for many real-time services, as well as autonomous and embedded applications.

For the reasons listed above, we decided to test it on our platform and try to increase the output in terms of fps in order to get a smoother video stream.

In order to avoid confusion and overwriting of files and folders we decided to install TensorRT on a different microSD in order to have the two environments completely separate and independent. So what we'll do is to follow again the procedure explained in 3.1 with the only difference on the version of the operating system image that will be **JetPack 4.4** this time. Once the Jetpack installation is complete, we enter the terminal and start installing the additional libraries and frameworks required for TensorRT to work.

The first of these is Pytorch which consists of two main parts which are tensor computing (NumPy) with strong acceleration via graphics processing units (GPU) and Deep Neural Networks (DNN).

```
sudo apt-get update

wget https://nvidia.box.com/shared/static/
    yr6sjswn25z7oankw8zy1roow9cy5ur1.whl -O torch-1.6.0-cp36-
    cp36m-linux_aarch64.whl

sudo apt-get install python3-pip libopenblas-base libopenmpi
    -dev

pip3 install Cython

cd /usr/local/src

pip3 install torch-1.6.0-cp36-cp36m-linux_aarch64.whl
```

continue with the Torchvision machine learning framework, which is a separate module of Pytorch

```
sudo apt-get install libjpeg-dev zlib1g-dev

wget https://github.com/pytorch/vision/archive/v0.6.1.tar.gz

tar -xvzf v0.6.1.tar.gz

cd vision-0.6.1

sudo python3 setup.py install
```

```
apt-get install libfreetype6-dev

pip3 uninstall pillow

pip3 install --no-cache-dir pillow
```

we need to add the touch2trt converter to the latter two to be able to work easily between the various formats of Pytorch and TensorRT

```
cd /usr/local/src

git clone https://github.com/NVIDIA-AI-IOT/torch2trt

cd torch2trt

sudo python3 setup.py install --plugins
```

These packages are also needed

```
pip3 install tqdm cython pycocotools

apt-get install python3-matplotlib

cd /usr/local/src

git clone https://github.com/NVIDIA-AI-IOT/trt_pose

cd trt_pose

python3 setup.py install
```

Now we move on to the pre-trained models that we will feed to the machine learning frameworks for human recognition and identification.

We can easily download the first model:

`resnet18_baseline_att_224x224_A`

at the link [5] and the second one:

`densenet121_baseline_att_256x256_B`

at [6] and we move it in `/human_pose` directory from `/Scaricati` or `/Downloads` depending on the language of the operative system we have installed.

[5] `https://drive.google.com/file/d/1XYDdCUdiF2xxx4rznmLb62SdOUZuoNbd/view`

[6] `https://drive.google.com/file/d/13FkJkx7evQ1WwP54UmdiDXWyFMY1OxDU/view`

```
cd /home/user/Scaricati

mv nome_file_1 /usr/local/src/trt_pose/tasks/human_pose

mv nome_file_2 /usr/local/src/trt_pose/tasks/human_pose
```

In order to recognise video devices including the webcam, TensorRT relies on JetCam, an archive of drivers which we install with

```
cd /usr/local/src/trt_pose/tasks/human_pose

git clone https://github.com/NVIDIA-AI-IOT/jetcam

cd jetcam

sudo python3 setup.py install
```

As usual, we verify the correct installation of the software with a series of commands that call up demo versions of Python scripts already present in the installation itself. We enter the working directory

```
cd /usr/local/src/trt_pose/tasks/human_pose
```

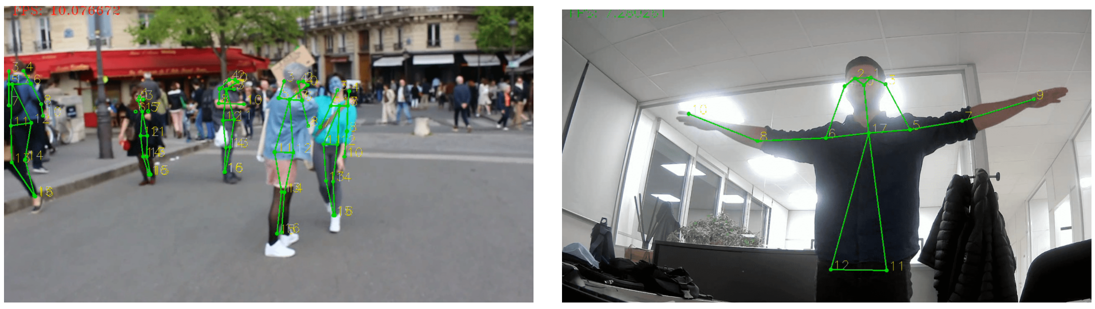
and type individually one of the following commands

```
sudo python3 detect_video.py

sudo python3 detect_webcam.py

sudo python3 detect_image.py
```

in order to test reading from video, webcam and image files respectively. The output should look something like this



(a) (b)

Figure 3.7: TensorRT testing. (a) Demo video file. (b) Webcam test.

If, on the other hand, you need to modify the various scripts in Python to suit your needs or create new ones, you can do so with

```
sudo gedit nome_script.py
```

as it is in our case where we are going to modify them to extract particular coordinates of points to be fed to subsequent processing in the algorithms.

3.2.3 Digital identity

As mentioned in the introduction, part of the thesis work also consisted of developing a subject identification algorithm based on limb lengths and t-shirt colors. This last can be implemented and easily transformed in Python language and then loaded on the Jetson Nano board in order to distinguish one person from another.

The very first attempt to implement it was carried out by colleagues during their master thesis work a few months ago. And now resuming their work and analyzing what has been done so far we are going to optimize and to solve a series of still remaining problems on it.

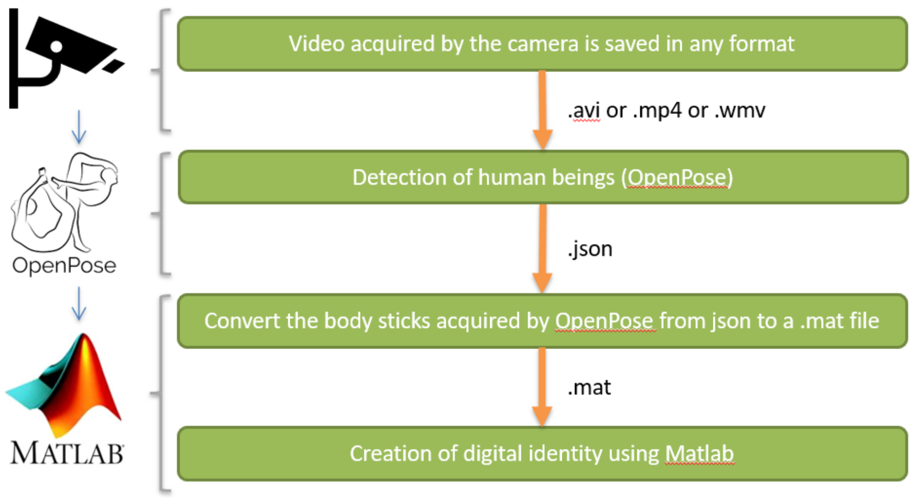
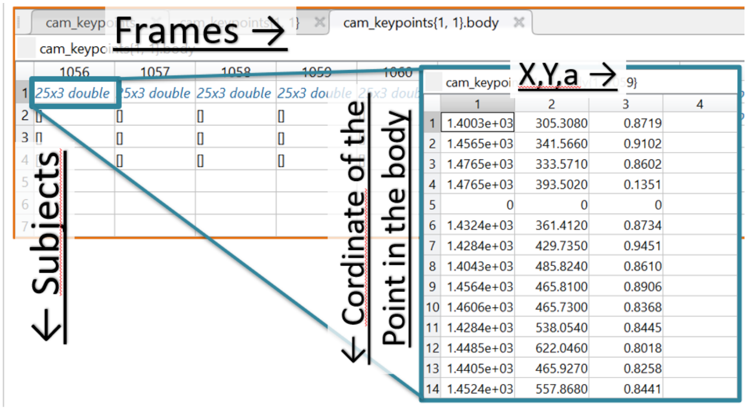

Video acquired by the camera is saved in any format
.avi or .mp4 or .wmv
Detection of human beings (OpenPose)
.json
OpenPose
Convert the body sticks acquired by OpenPose from json to a .mat file
.mat
MATLAB
Creation of digital identity using Matlab

Figure 3.8: Preliminary operations to the algorithm

In the figure above you can see what are the preliminary operations to perform on one of the standard formats coming from the surveillance cameras which are .avi, .mp4 and .wmv. The video files are fed to an identification and tracking software that in our case is OpenPose. The software returns in output a .json file containing the spatial coordinates of all the keypoints belonging to the body sticks of all the subjects detected. The coordinates are organized according to a logic not immediately readable.

For this reason there is a post-processing of the .json file in MatLab in order to obtain an ordered structure of data that the .mat. It is nothing more than a structured matrix within which the coordinates are organized by rows and columns. The rows refer to the various subjects on the screen while the columns represent the frames coming from the camera.



	1056	1057	1058	1059	1060
1	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double
2	[]	[]	[]	[]	
3	[]	[]	[]	[]	
4	[]	[]	[]	[]	
5					
6					
7					

	1	2	3	4
1	1.4003e+03	305.3080	0.8719	
2	1.4565e+03	341.5660	0.9102	
3	1.4765e+03	333.5710	0.8602	
4	1.4765e+03	393.5020	0.1351	
5	0	0	0	
6	1.4324e+03	361.4120	0.8734	
7	1.4284e+03	429.7350	0.9451	
8	1.4043e+03	485.8240	0.8610	
9	1.4564e+03	465.8100	0.8906	
10	1.4606e+03	465.7300	0.8368	
11	1.4284e+03	538.0540	0.8445	
12	1.4485e+03	622.0460	0.8018	
13	1.4405e+03	465.9270	0.8258	
14	1.4524e+03	557.8680	0.8441	

Figure 3.9: Internal structure of the input file originating from OpenPose and processed in Matlab

Following a careful analysis of the output file coming from OpenPose two main problems emerged.

The first one concerns the presence of fake subjects that are generated when complicated situations are created. For complicated situations we mean the overlapping between subjects and occlusion, the initial and final moment in which there is a partial visibility of the bust and finally the rotation on itself that causes a degeneration of the body stick in a line. All this causes the software to go into crisis and create phantom body sticks having only part of the spatial coordinates and moreover belonging to real body sticks. What happens in reality is a separation of coordinates where one part is assigned to a new non-existent or fake subject and the other is assigned to the real subject.

The second problem concerns the order in which the various inputs and outputs of the subjects are recorded. Often this chronological order is lacking because of one of the above reasons and continuity is lost. Let us explain better what is meant by chronological order.

If a subject enters the visible field of the camera, one expects that it will be recorded on line one and remain there until the moment it leaves the screen. Unfortunately the order is not maintained and often the subjects are inverted and recorded intermittently on two different rows. This involves considerable difficulties when we intend to apply an algorithm for identification and recognition.

The **fake detection algorithm** uses the matrices of the points, of the presences and that one of the subjects to individualize and to eliminate the fakes. The logic followed is as follows.

Whenever there is a change in the number of subjects on the frame, two cases are possible. The first is that it is simply a regular input or output. The second is

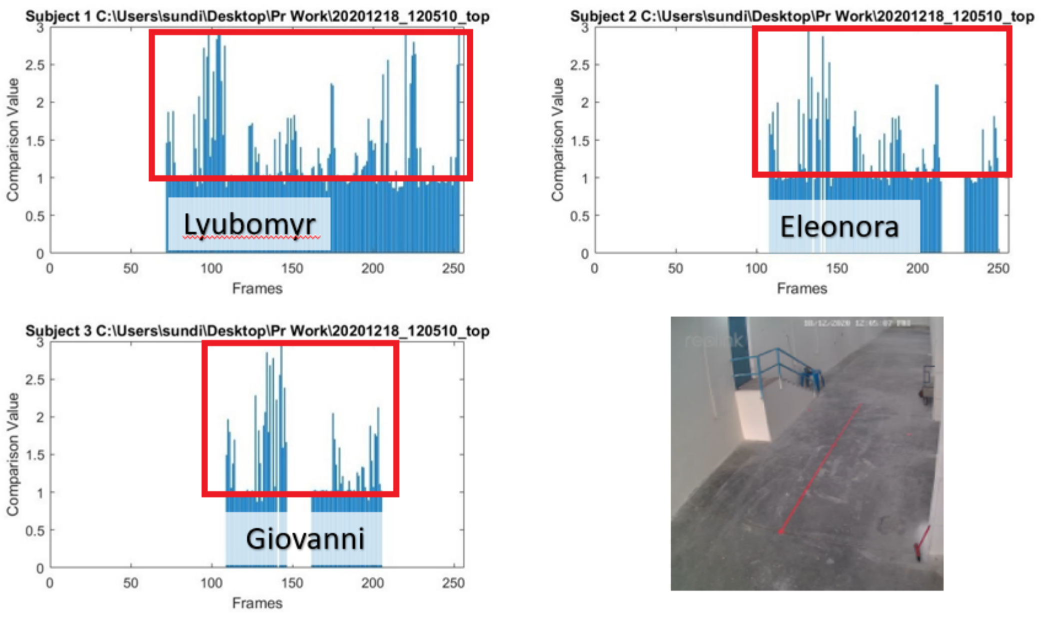


Figure 3.10: Unreliable identification due to fake detections, distorsions and subjects interchange.

that it is a fake generated by the overlap between subjects or coming from a non-continuous detection.If the new appearance is maintained for n successive frames then it is not a fake otherwise we remove it. But before doing that a check is done in case the subjects on screen are more than one. In these cases we try to recover some points from the famous split fakes caused by overlapping.

The **ordering algorithm** makes use of the matrix of subjects and distances to restore the chronological odrine of inputs and outputs. This is based on the sequence between the current and previous frames. The matrix of the average distances between all the subjects present in the frames is calculated and the minimums within the matrix are searched for. Once identified n minima where n will be given by the maximum number of subjects between the two frames and bring them on the main diagonal. So the condition that must be met to have a continuous flow of subjects in the file is that the minimum distances between all pairs are reported on the main diagonal.

The following are flow diagrams representing the logic used for the detection of fakes, their elimination and the chronological ordering of inputs and outputs.

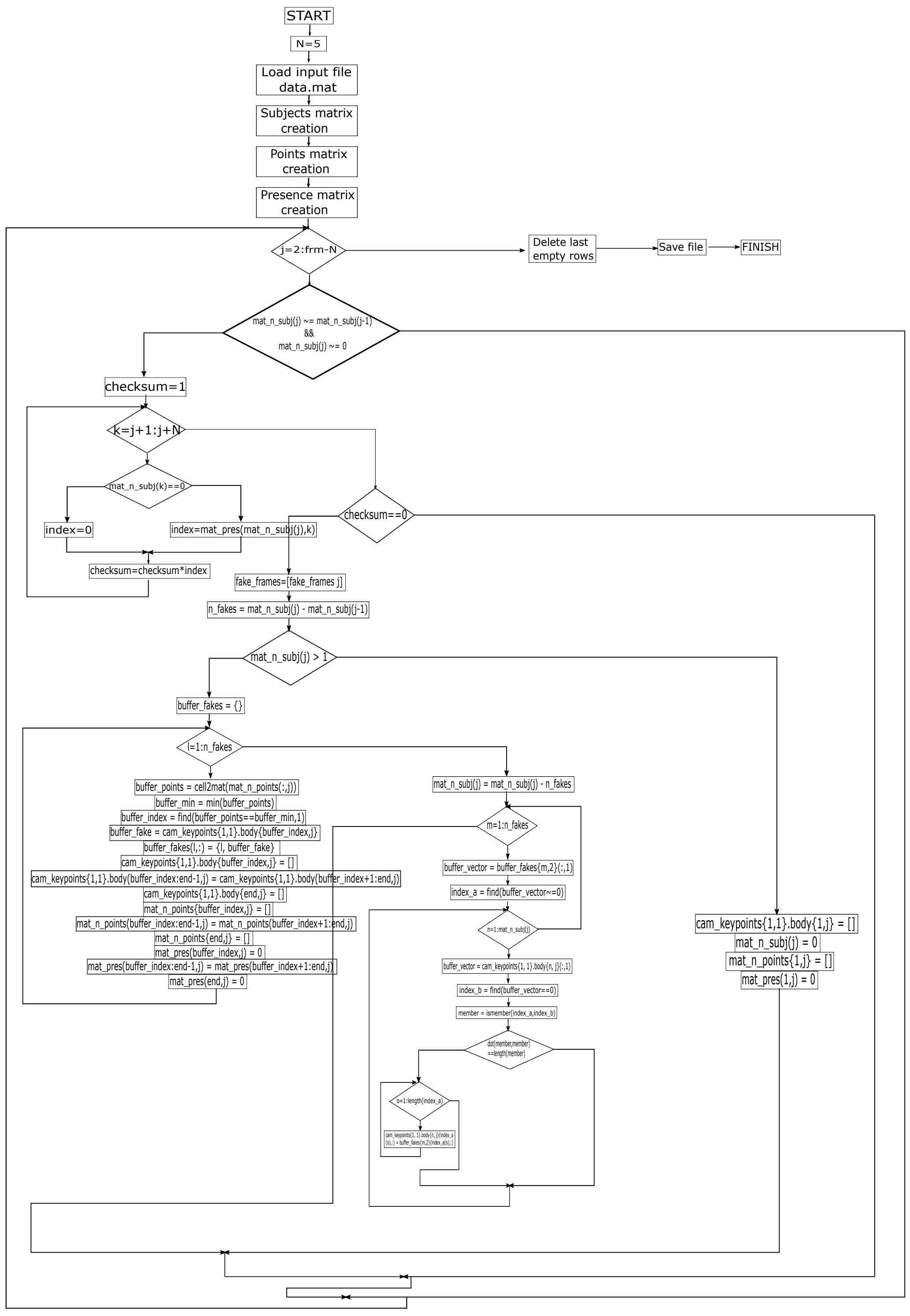

START
N=5
Load input file data.mat
Subjects matrix creation
Points matrix creation
Presence matrix creation
j=2:frm-N
Delete last empty rows
Save file
FINISH
mat_n_subj(j) ~= mat_n_subj(j-1) && mat_n_subj(j) ~= 0
checksum=1
k=j+1:j+N
mat_n_subj(k)==0
index=0
index=mat_pres(mat_n_subj(j),k)
checksum=checksum*index
checksum==0
fake_frames=[fake_frames j]
n_fakes = mat_n_subj(j) - mat_n_subj(j-1)
mat_n_subj(j) > 1
buffer_fakes = {}
l=1:n_fakes
buffer_points = cell2mat(mat_n_points(:,j))
buffer_min = min(buffer_points)
buffer_index = find(buffer_points==buffer_min,1)
buffer_fake = cam_keypoints{1,1}.body{buffer_index,j}
buffer_fakes(l,:) = {l, buffer_fake}
cam_keypoints{1,1}.body{buffer_index,j} = []
cam_keypoints{1,1}.body(buffer_index:end-1,j) = cam_keypoints{1,1}.body(buffer_index+1:end,j)
cam_keypoints{1,1}.body{end,j} = []
mat_n_points{buffer_index,j} = []
mat_n_points(buffer_index:end-1,j) = mat_n_points(buffer_index+1:end,j)
mat_n_points{end,j} = []
mat_pres(buffer_index,j) = 0
mat_pres(buffer_index:end-1,j) = mat_pres(buffer_index+1:end,j)
mat_pres(end,j) = 0
mat_n_subj(j) = mat_n_subj(j) - n_fakes
m=1:n_fakes
buffer_vector = buffer_fakes{m,2}(:,1)
index_a = find(buffer_vector~=0)
n=1:mat_n_subj(j)
buffer_vector = cam_keypoints{1, 1}.body{n, j}(:,1)
index_b = find(buffer_vector==0)
member = ismember(index_a,index_b)
dot(member,member) ==length(member)
o=1:length(index_a)
cam_keypoints{1, 1}.body{n, j}(index_a(o),:) = buffer_fakes{m,2}(index_a(o),:)
cam_keypoints{1,1}.body{1,j} = []
mat_n_subj(j) = 0
mat_n_points{1,j} = []
mat_pres(1,j) = 0

Figure 3.11: MatLab find fakes block diagram

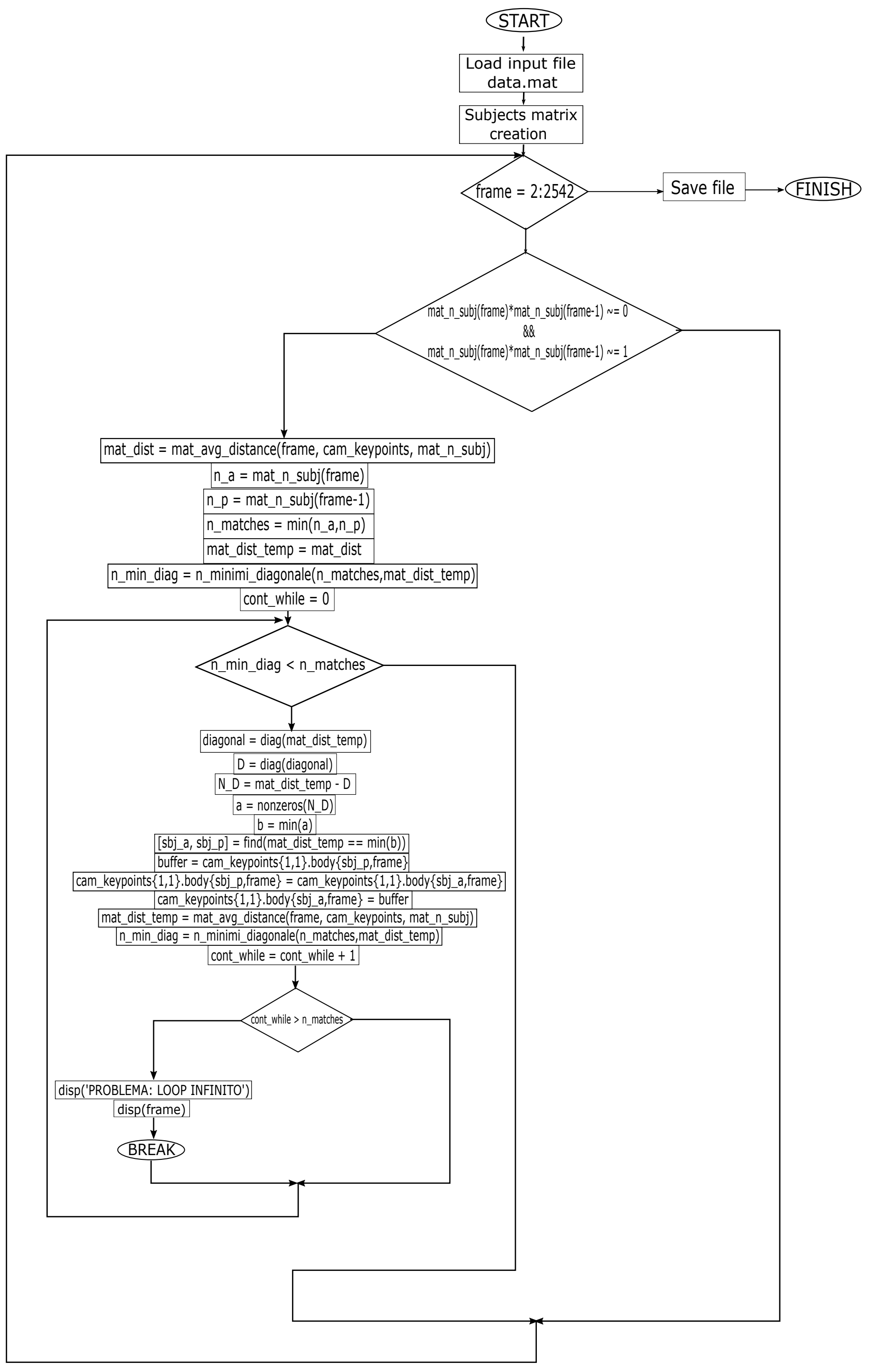

START
Load input file data.mat
Subjects matrix creation
frame = 2:2542
Save file
FINISH
mat_n_subj(frame)*mat_n_subj(frame-1) ~= 0
&&
mat_n_subj(frame)*mat_n_subj(frame-1) ~= 1
mat_dist = mat_avg_distance(frame, cam_keypoints, mat_n_subj)
n_a = mat_n_subj(frame)
n_p = mat_n_subj(frame-1)
n_matches = min(n_a,n_p)
mat_dist_temp = mat_dist
n_min_diag = n_minimi_diagonale(n_matches,mat_dist_temp)
cont_while = 0
n_min_diag < n_matches
diagonal = diag(mat_dist_temp)
D = diag(diagonal)
N_D = mat_dist_temp - D
a = nonzeros(N_D)
b = min(a)
[sbj_a, sbj_p] = find(mat_dist_temp == min(b))
buffer = cam_keypoints{1,1}.body{sbj_p,frame}
cam_keypoints{1,1}.body{sbj_p,frame} = cam_keypoints{1,1}.body{sbj_a,frame}
cam_keypoints{1,1}.body{sbj_a,frame} = buffer
mat_dist_temp = mat_avg_distance(frame, cam_keypoints, mat_n_subj)
n_min_diag = n_minimi_diagonale(n_matches,mat_dist_temp)
cont_while = cont_while + 1
cont_while > n_matches
disp('PROBLEMA: LOOP INFINITO')
disp(frame)
BREAK

Figure 3.12: MatLab sorting subjects flow chart

Chapter 4

Experimental results

4.1 Setup description

The test station for the Jetson Nano platform was initially installed and tested on the table as shown in the figure below 4.1.

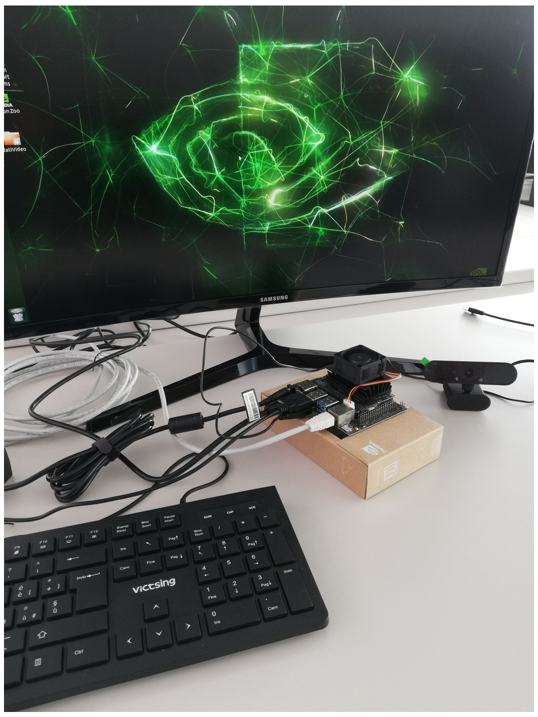


Figure 4.1: Experimental workstation setup and peripheral devices on Jetson Nano

Subsequently, in order to test it in what would have been its final configuration, it was moved to the lab and connected to the camera via USB cable.

We list below the hardware and peripherals used to equip the test bench. The

Jetson Nano has been connected to a monitor through the HDMI cable in order to work in an agile way on the terminal and related Linux commands. Need then a USB keyboard and mouse to give commands and be able to interface with the same. At the third USB 3.0 port we connect a **Teaisiy PC webcam HD 1080P 30fps** that acts as a surveillance camera. It was also purchased a fan to be installed on the passive heatsink because the temperatures were still too high, especially after prolonged video processing.

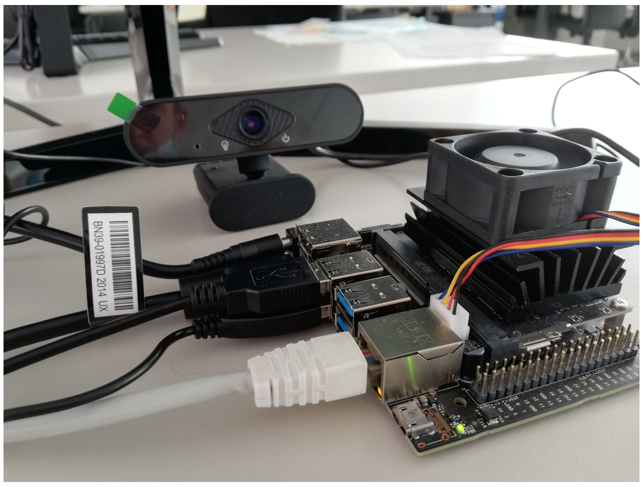


Figure 4.2: Jetson Nano setup

The power supply of the Jetson has been extensively discussed in previous paragraphs so we will power it with a Power Jack J25. To communicate with the outside world there are two possibilities: additional WI-FI module to be installed on the board or the classic LAN socket. There are several pros and cons between the two ways to send information to the outside world. In the initial phase it was decided to connect it via Ethernet cable to the LAN socket.

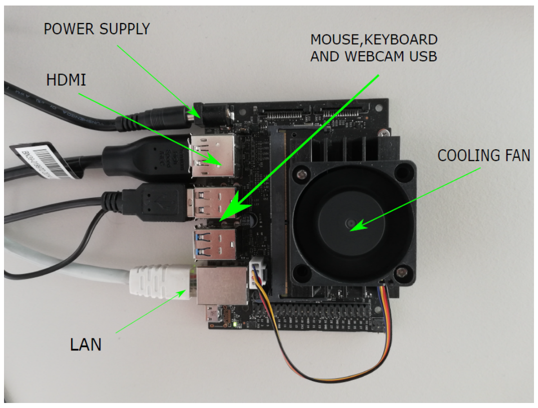


Figure 4.3: Jetson Nano and its devices

4.2 OpenPose output

The first test on the Jetson Nano was done with a demo video file provided by OpenPose developers and our USB webcam getting bad performances around 0.6fps. It immediately became apparent that the limited resources of the Nano cannot support a heavy application like OpenPose. In order to get as close as possible to the desired processing speed which was about 5fps, the output resolution was reduced to 128x64. Lowering it with the command net_resolution has allowed us to reach the max processing speed performance which was **4.2fps** but we also went to worsen precision.

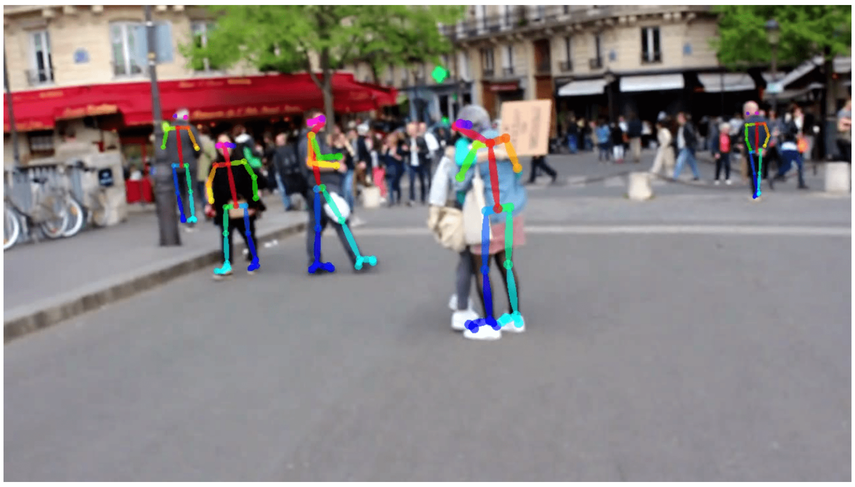


Figure 4.4: OpenPose tested on demo video file

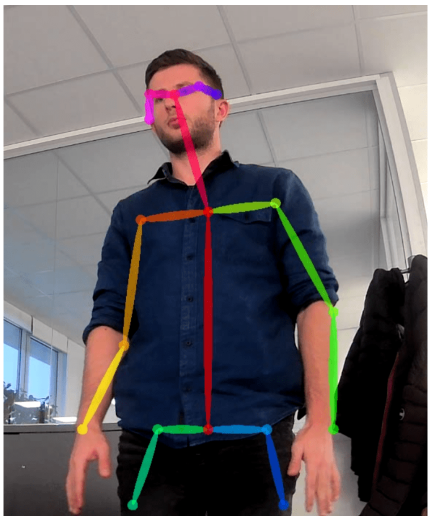


Figure 4.5: OpenPose tested on usb webcam

After that, the Jetson was moved to the research and development lab to be tested in real working conditions. As you can see in the picture 4.6 on the floor was drawn with red tape a path to follow in order to have references. The room where the tests took place is 15m long and 6m wide. The results of the test have confirmed the fact that OpenPose is very efficient over long distances and can perfectly identify the body stick even in conditions of partial visibility due to the tables in the lab. The problem remained the processing speed that could not meet the requirements. For this reason we continued with the research on available software and applications that could bring us closer to the goal.

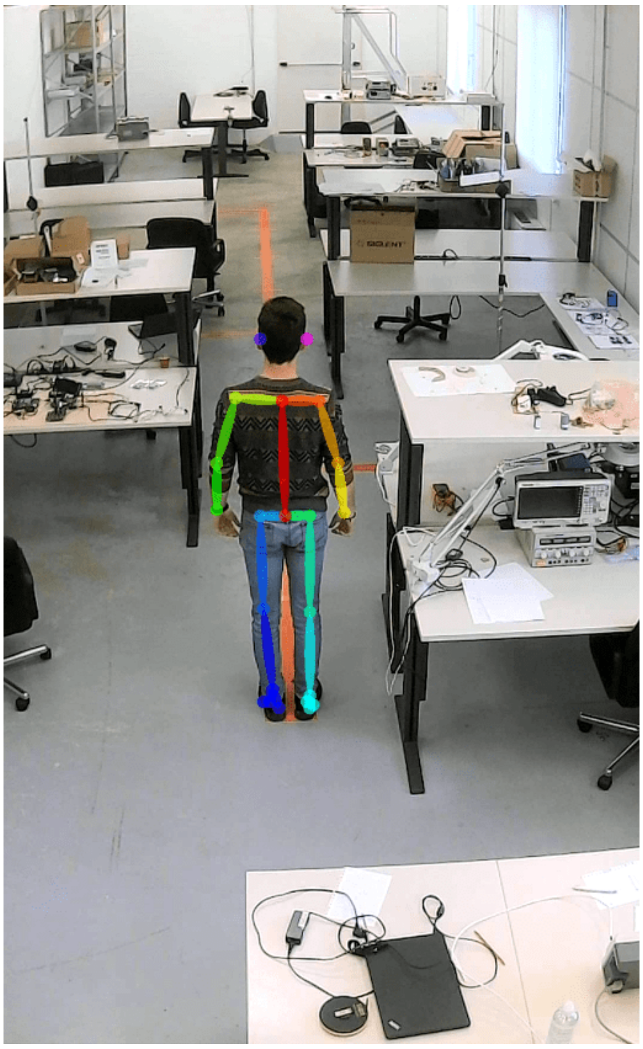


Figure 4.6: OpenPose tested in the laboratory. Starting point of the path

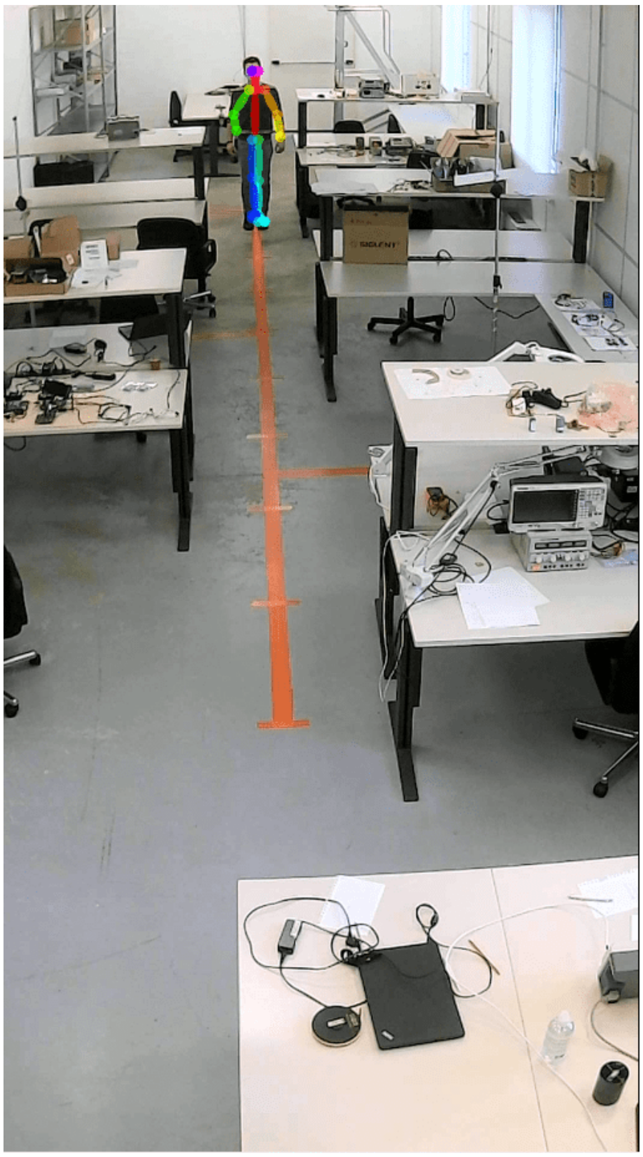


Figure 4.7: OpenPose tested in the laboratory. Ending point of the path

4.3 TrtPose output

Unlike OpenPose this time the subject at the bottom of the lab is not recognized. This can be attributed to the lower computational power required by TrtPose. You gain in processing speed and lose the ability to identify at long distances.

On the positive side, you can get **up to 10 fps** of processing, so in short distance applications it can be a valid alternative. Especially on on-edge devices where hardware resources are very limited.

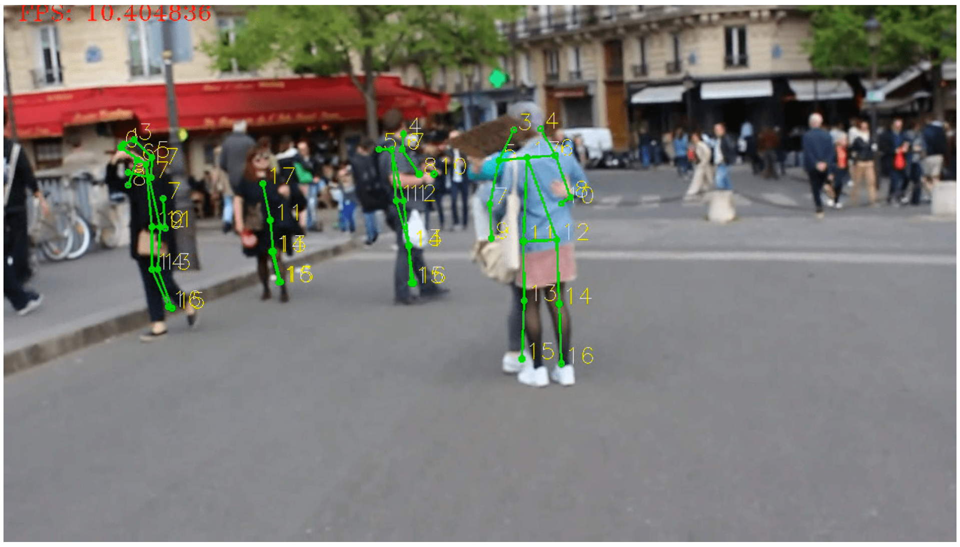


Figure 4.8: TrtPose tested on demo video file

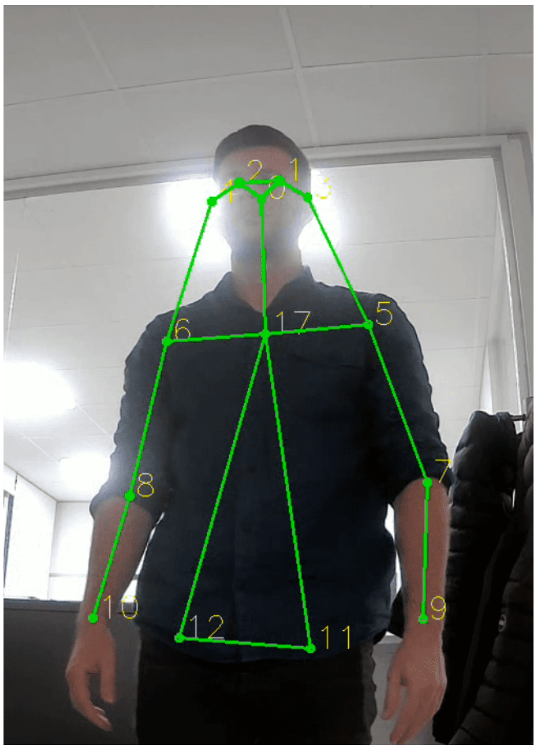


Figure 4.9: TrtPose tested on usb webcam

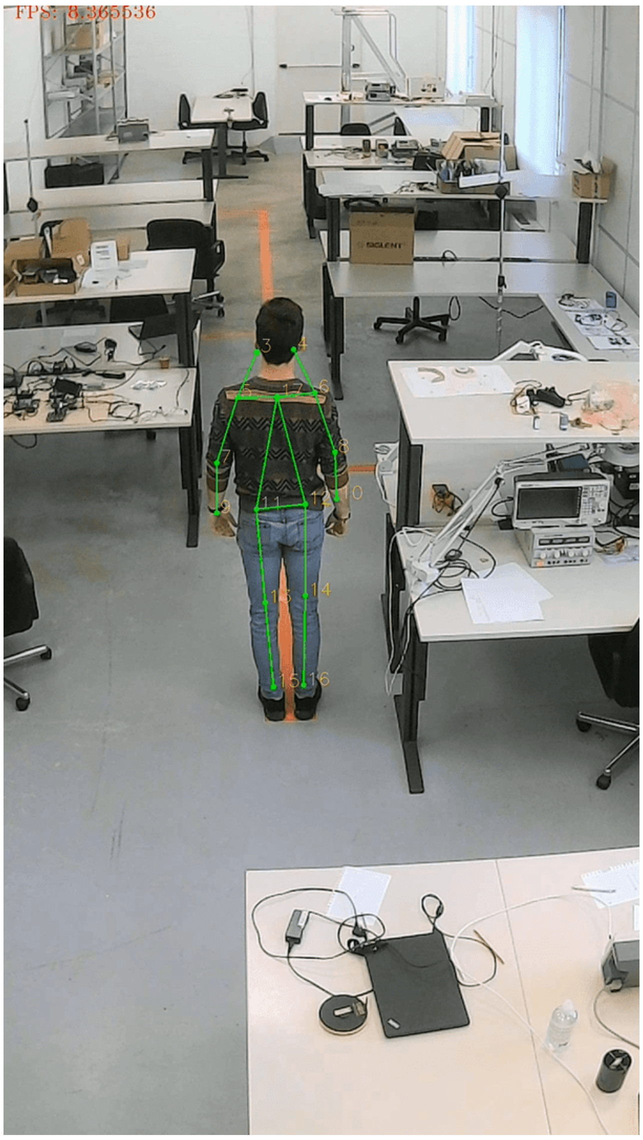


Figure 4.10: TrtPose tested in the laboratory up to 10 fps. Starting point of the path

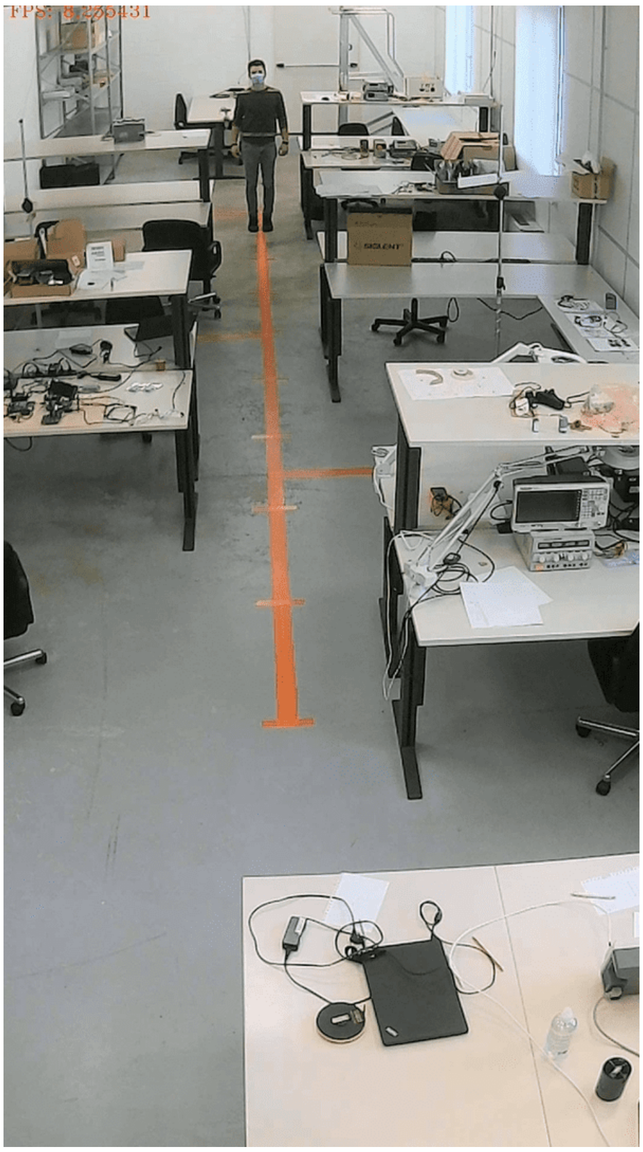


Figure 4.11: TrtPose tested in the laboratory up to 10 fps. Ending point of the path

4.4 TrackLite output

The third and final application used is TrackLite[1] with the maximum performance achieved on the Jetson Nano of **2.2fps**. A computer vision library that puts together YOLOv3 and DeepSORT to try to achieve what is called by experts in the field the re-identification. YOLO identifies people by drawing a rectangle that encloses the identified person. Once identified, DeepSORT assigns a progressive identifier to

[1]https://github.com/Stephenfang51/tracklite

the rectangle and tries to keep it in memory even after the exit of the subject. The retention time and accuracy depend on many factors. As you can see in the explanatory video recorded in the laboratory, the quality of the re-identification depends for example on the color of the clothing used. In fact, with two subjects identified on the video, one with a red shirt and the other with a gray shirt, the one that returns better results is precisely the red. This is because the gray shirt is confused with the laboratory tables that have a similar shade of gray and then the software often loses count.

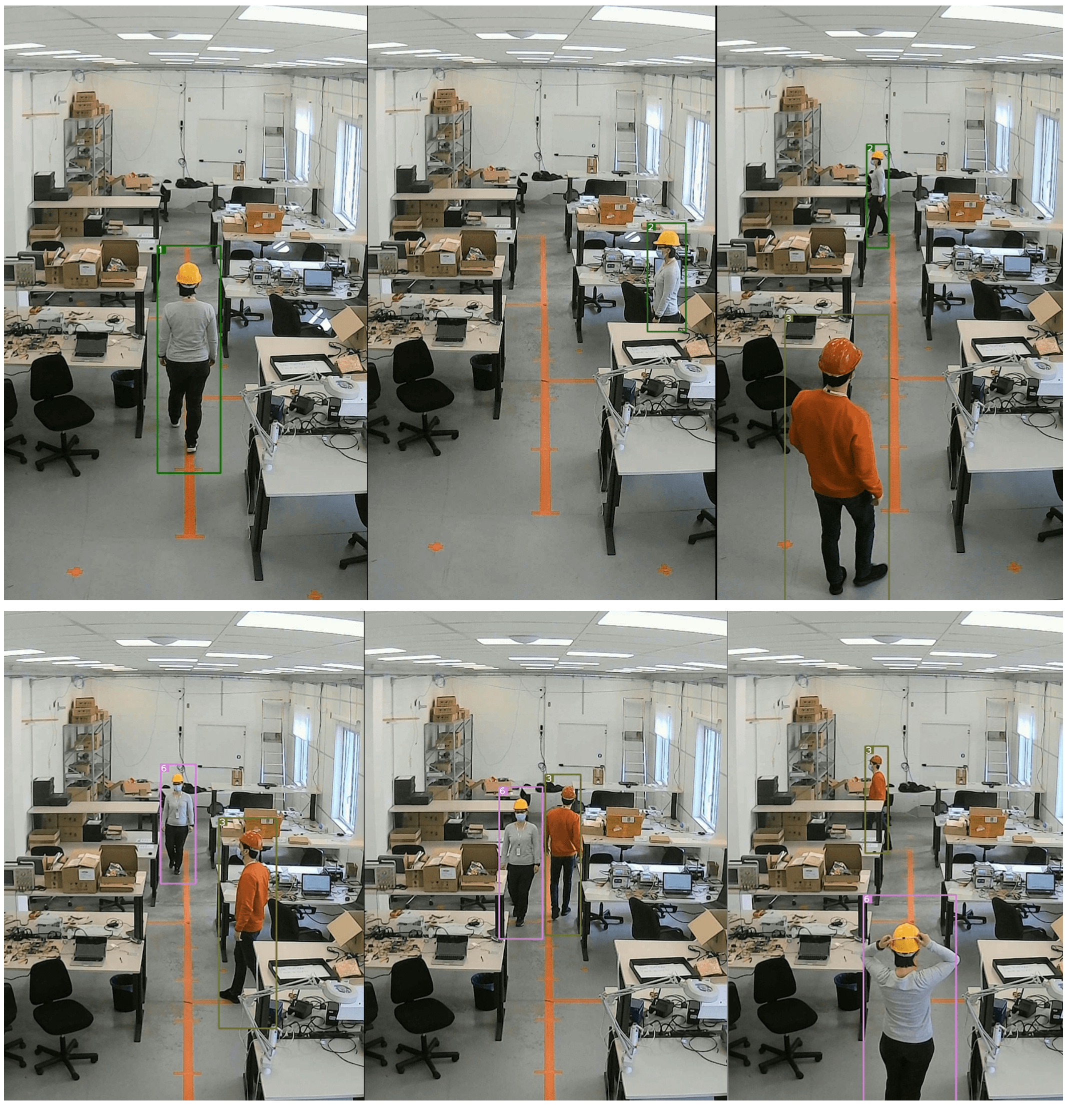


Figure 4.12: Sequence extracted from the workshop video. From this we can see the loss of count due to the color of the shirt similar to the surrounding environment.

The other test was conducted concurrently with the R&D team in the outdoor environment this time with natural sunlight. It served to understand how natural light affected the quality of identification and re-identification in addition to measuring the decrease in speed due to the number of increasing appearances.

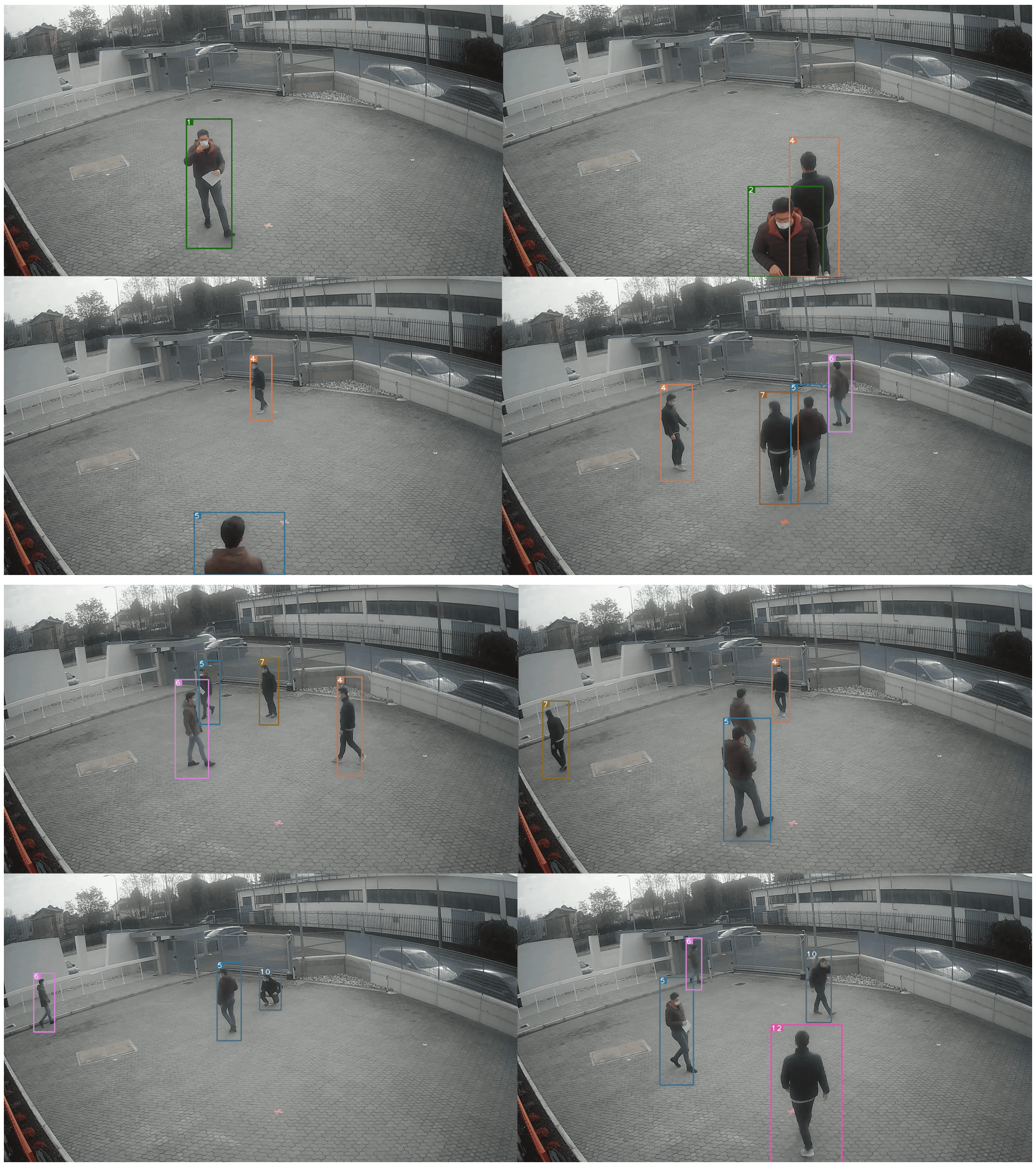


Figure 4.13: Sequence extracted from the parking video testing. From this we can see the even worse loss of count with respect to the internal environment.

The last test carried out was to confirm the fact that as the resolution and the number of people present on the frame at the same time increased, it had a negative impact on the processing speed measured in fps. The final test was carried out indoors this time in the research and development lab with artificial light.

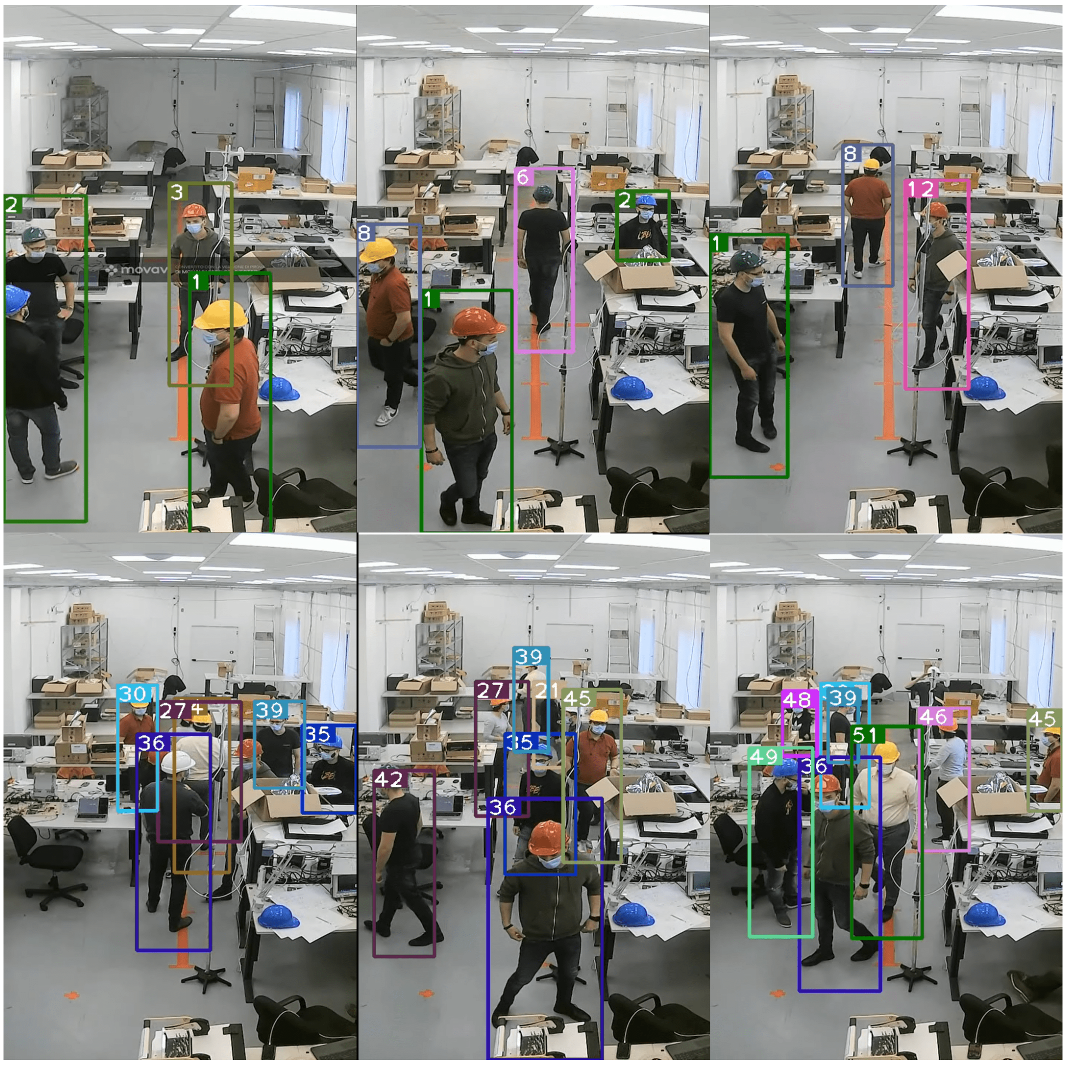
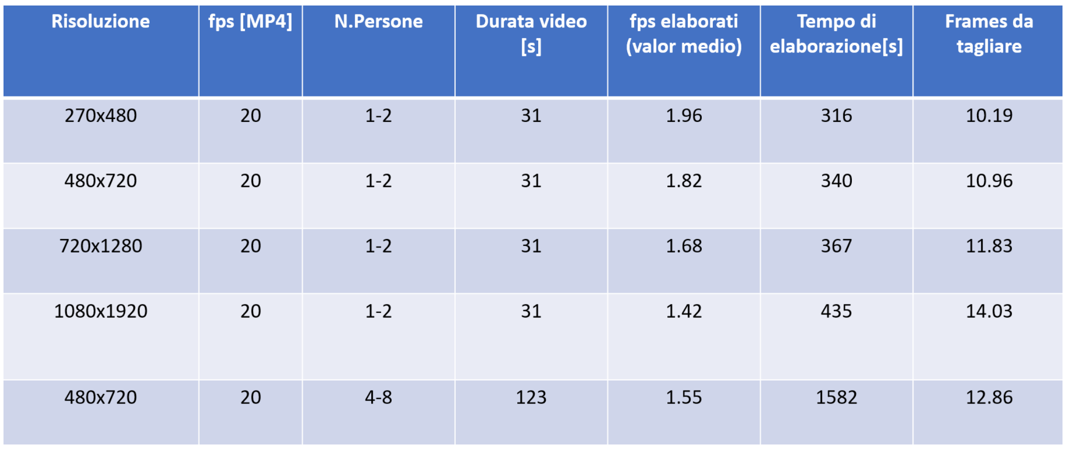


Figure 4.14: Final test done within the lab confirming the performance decrease due an increase of human detections.

In the following table 4.15 instead we report the results obtained by analyzing two video streams within which the number of persons detected changes from 1-2 Figure 4.12 to 4-8 Figure 4.14. Moreover they have been analyzed at different frame resolutions starting from 270x480 and up to 1080x1920.

From the results obtained it can be seen that as the resolution of the frame increases, the fps returned in output decreases and therefore the processing speed slows down. It is something that was expected since as the resolution increases the



Risoluzione	fps [MP4]	N.Persone	Durata video [s]	fps elaborati (valor medio)	Tempo di elaborazione[s]	Frames da tagliare
270x480	20	1-2	31	1.96	316	10.19
480x720	20	1-2	31	1.82	340	10.96
720x1280	20	1-2	31	1.68	367	11.83
1080x1920	20	1-2	31	1.42	435	14.03
480x720	20	4-8	123	1.55	1582	12.86

Figure 4.15: Final results based on pixel resolution and number of detections on TrackLite framework

number of pixels to analyze increases.

Moreover, with the same resolution, quadrupling the number of subjects present on the frame implies a reduction of 15% of the processed fps passing from 1.82 to 1.55fps.

4.5 Performances benchmark

In the above figure, it is important to make a premise. The results shown on the graph represent the best performance obtained in terms of frames per second processed with a maximum of 2 people present on the screen. The resolution adopted is the optimal one for each of the applications in order to optimize the output. For example for OpenPose the optimal resolution on Jetson is 256x128. If increased it causes delays in processing and the output on the monitor is intermittent. The reason is always the limited resources of the on-edge device.

Another very important factor to mention is the type of application and its ultimate purpose. As mentioned earlier OpenPose is very powerful and able to detect people even at great distances. Having limited hardware resources on which to run the on-edge device is not able to exploit its potential. TrtPose being lighter and less performant on distance returns higher fps. Now it's fair to compare OpenPose vs TrtPose because both return in output the body-stick of the subject on the screen. Instead TrackLite with YOLOv3 and DeepSORT inside does something slightly different and much more challenging. It detects the people present and goes to calculate their positions with the Kalman algorithm to be able to re-identify them in the following moments by assigning an id to each. For this reason the processing speed suffers a lot and the resulting performance is only 2fps.

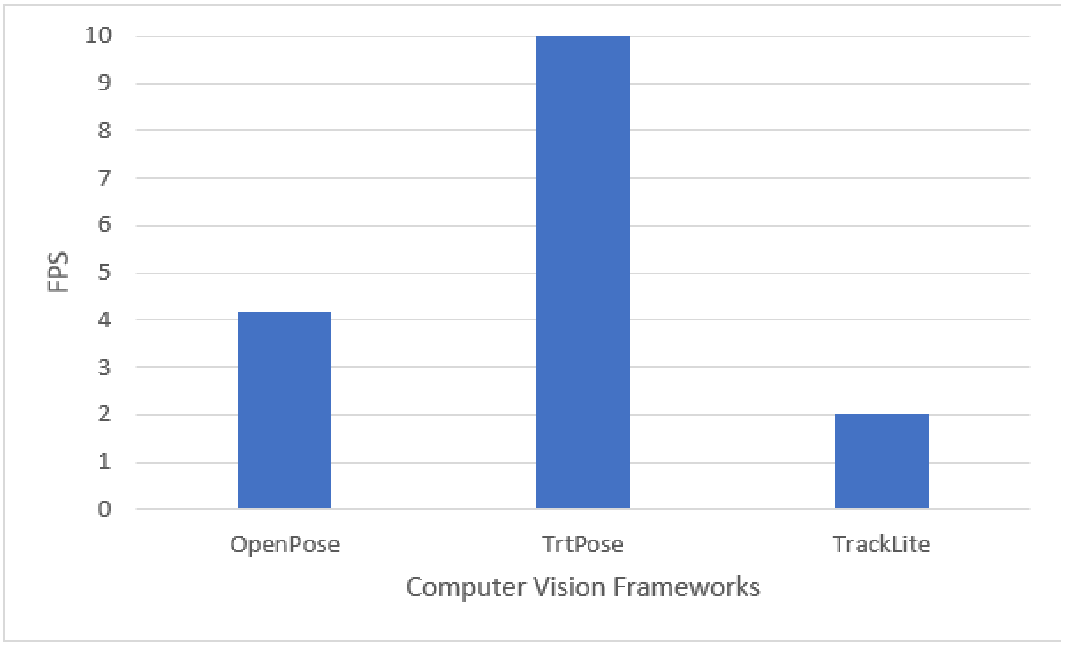


Figure 4.16: Final benchmark showing the best performance obtained with three different CV frameworks

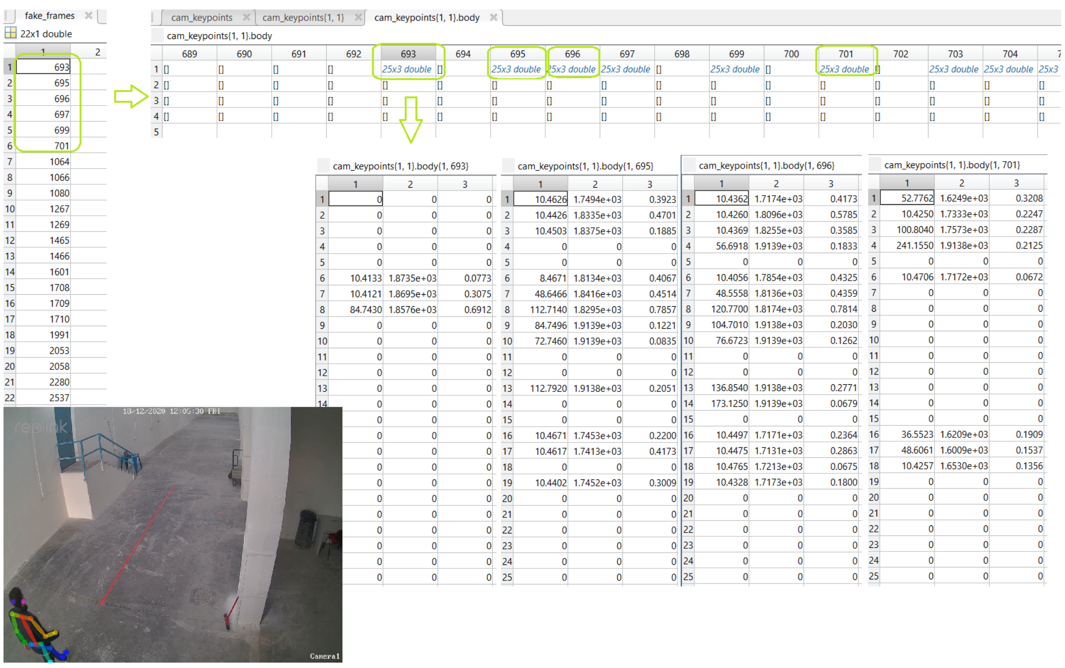
4.6 Digital Identity results

The very first result achieved was the identification and subsequent elimination of fake detections. In some particular cases it was also possible to recover split coordinates due to overlapping.

Twenty-two fake frames were identified for the video considered with the Find_fakes algorithm illustrated in 3.11.

In the first case below the fakes are identified as such due to partial visibility of the incoming subject, incomplete identification and intermittent detection. And this results are shown in a few detected body stick point coordinates where the remaining are zero. So what we improperly identify as a fake here is actually an unreliable detection. Which translated into code means that the subject is not detected correctly for **at least five successive frames**.

In the second case illustrated instead and in particular frame 1080 is a real fake. In the video there are three subjects while the OpenPose cam_keypoints file shows four, which is clearly an error. The problem arises from the fact that the two subjects are very close together and overlaps are created, which in turn causes problems in terms of the final coordinates recorded in the input file.



fake_frames — 22x1 double

	1	2
1	693	
2	695	
3	696	
4	697	
5	699	
6	701	
7	1064	
8	1066	
9	1080	
10	1267	
11	1269	
12	1465	
13	1466	
14	1601	
15	1708	
16	1709	
17	1710	
18	1991	
19	2053	
20	2058	
21	2280	
22	2537	

cam_keypoints | cam_keypoints{1, 1} | cam_keypoints{1, 1}.body

cam_keypoints{1, 1}.body

	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	7
1	[]	[]	[]	[]	25x3 double	[]	25x3 double	25x3 double	25x3 double	[]	25x3 double	[]	25x3 double	[]	25x3 double	25x3 double	25x3
2	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]
3	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]
4	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]
5																	

cam_keypoints{1, 1}.body{1, 693}

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	10.4133	1.8735e+03	0.0773
7	10.4121	1.8695e+03	0.3075
8	84.7430	1.8576e+03	0.6912
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0

cam_keypoints{1, 1}.body{1, 695}

	1	2	3
1	10.4626	1.7494e+03	0.3923
2	10.4426	1.8335e+03	0.4701
3	10.4503	1.8375e+03	0.1885
4	0	0	0
5	0	0	0
6	8.4671	1.8134e+03	0.4067
7	48.6466	1.8416e+03	0.4514
8	112.7140	1.8295e+03	0.7857
9	84.7496	1.9139e+03	0.1221
10	72.7460	1.9139e+03	0.0835
11	0	0	0
12	0	0	0
13	112.7920	1.9138e+03	0.2051
14	0	0	0
15	0	0	0
16	10.4671	1.7453e+03	0.2200
17	10.4617	1.7413e+03	0.4173
18	0	0	0
19	10.4402	1.7452e+03	0.3009
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

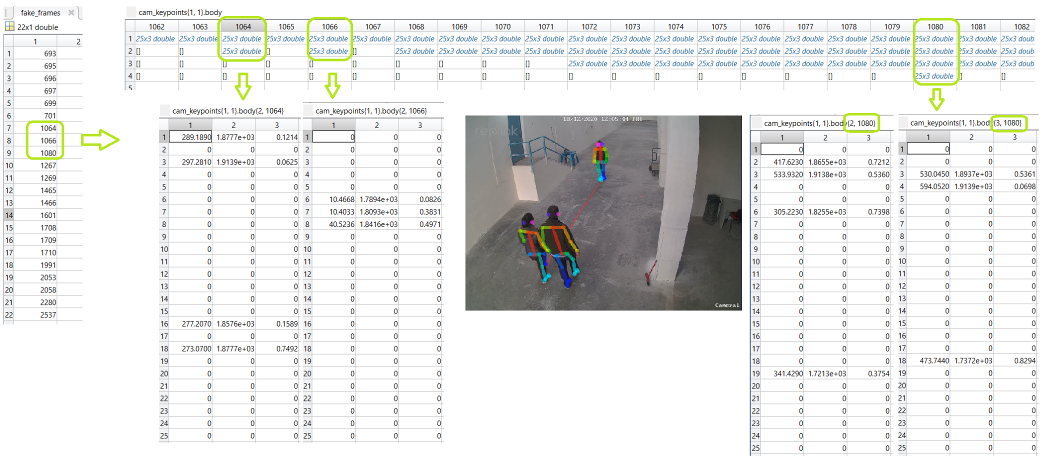
cam_keypoints{1, 1}.body{1, 696}

	1	2	3
1	10.4362	1.7174e+03	0.4173
2	10.4260	1.8096e+03	0.5785
3	10.4369	1.8255e+03	0.3585
4	56.6918	1.9139e+03	0.1833
5	0	0	0
6	10.4056	1.7854e+03	0.4325
7	48.5558	1.8136e+03	0.4359
8	120.7700	1.8174e+03	0.7814
9	104.7010	1.9138e+03	0.2030
10	76.6723	1.9139e+03	0.1262
11	0	0	0
12	0	0	0
13	136.8540	1.9138e+03	0.2771
14	173.1250	1.9139e+03	0.0679
15	0	0	0
16	10.4497	1.7171e+03	0.2364
17	10.4475	1.7131e+03	0.2863
18	10.4765	1.7213e+03	0.0675
19	10.4328	1.7173e+03	0.1800
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

cam_keypoints{1, 1}.body{1, 701}

	1	2	3
1	52.7762	1.6249e+03	0.3208
2	10.4250	1.7333e+03	0.2247
3	100.8040	1.7573e+03	0.2287
4	241.1550	1.9138e+03	0.2125
5	0	0	0
6	10.4706	1.7172e+03	0.0672
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	36.5523	1.6209e+03	0.1909
17	48.6061	1.6009e+03	0.1537
18	10.4257	1.6530e+03	0.1356
19	0	0	0
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

Figure 4.17: First example of fakes detection frames



fake_frames — 22x1 double

	1	2
1	693	
2	695	
3	696	
4	697	
5	699	
6	701	
7	1064	
8	1066	
9	1080	
10	1267	
11	1269	
12	1465	
13	1466	
14	1601	
15	1708	
16	1709	
17	1710	
18	1991	
19	2053	
20	2058	
21	2280	
22	2537	

cam_keypoints{1, 1}.body

	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082
1	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 doub
2	[]	[]	25x3 double	[]	25x3 double	[]	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 doub
3	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 double	25x3 doub
4	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	[]	25x3 double	[]	[]
5																					

cam_keypoints{1, 1}.body{2, 1064}

	1	2	3
1	289.1890	1.8777e+03	0.1214
2	0	0	0
3	297.2810	1.9139e+03	0.0625
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	277.2070	1.8576e+03	0.1589
17	0	0	0
18	273.0700	1.8777e+03	0.7492
19	0	0	0
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

cam_keypoints{1, 1}.body{2, 1066}

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	10.4668	1.7894e+03	0.0826
7	10.4033	1.8093e+03	0.3831
8	40.5236	1.8416e+03	0.4971
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0	0
18	0	0	0
19	0	0	0
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

cam_keypoints{1, 1}.bod{2, 1080}

	1	2	3
1	0	0	0
2	417.6230	1.8655e+03	0.7212
3	533.9320	1.9138e+03	0.5360
4	0	0	0
5	0	0	0
6	305.2230	1.8255e+03	0.7398
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0	0
18	0	0	0
19	341.4290	1.7213e+03	0.3754
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

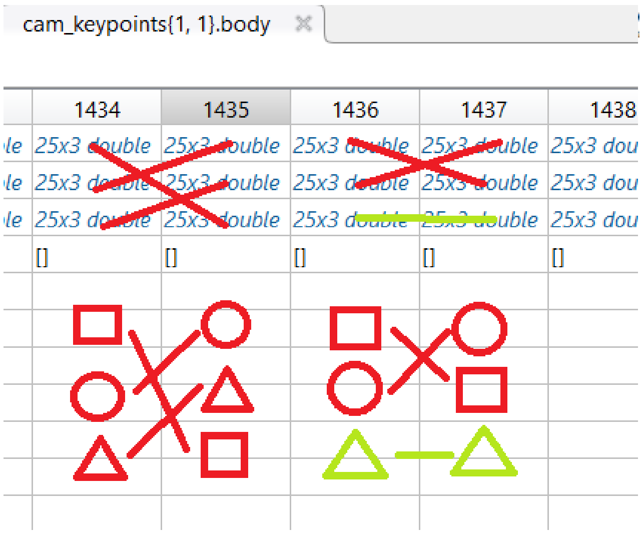
cam_keypoints{1, 1}.bod{3, 1080}

	1	2	3
1	0	0	0
2	0	0	0
3	530.0450	1.8937e+03	0.5361
4	594.0520	1.9139e+03	0.0698
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0	0
18	473.7440	1.7372e+03	0.8294
19	0	0	0
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0

Figure 4.18: Second example of fakes detection

The second part of the algorithm already shown in 3.12 is responsible for the chronological sorting of people entering and leaving the camera's field of view. The logic is as follows.

Whenever a new entry occurs it should always be recorded on the same row of the coordinate matrix for each frame called cam_keypoints. Doing so we have a continuity in the identification and we can say with certainty that once the subject 'A' has entered the field of view it will remain so until the moment it exits. Its coordinates can be found always on the line 'a' for all frames in which it is present on the screen.


cam_keypoints{1, 1}.body

	1434	1435	1436	1437	1438
le	25x3 double	25x3 double	25x3 double	25x3 double	25x3 dou
le	25x3 double	25x3 double	25x3 double	25x3 double	25x3 dou
le	25x3 double	25x3 double	25x3 double	25x3 double	25x3 dou
	[]	[]	[]	[]	[]

Figure 4.19: Scheme representing the usual situation without continuity in detection

Using this logic it is possible to obtain the temporal diagram of the subjects and have a clear situation of what are inputs and outputs.

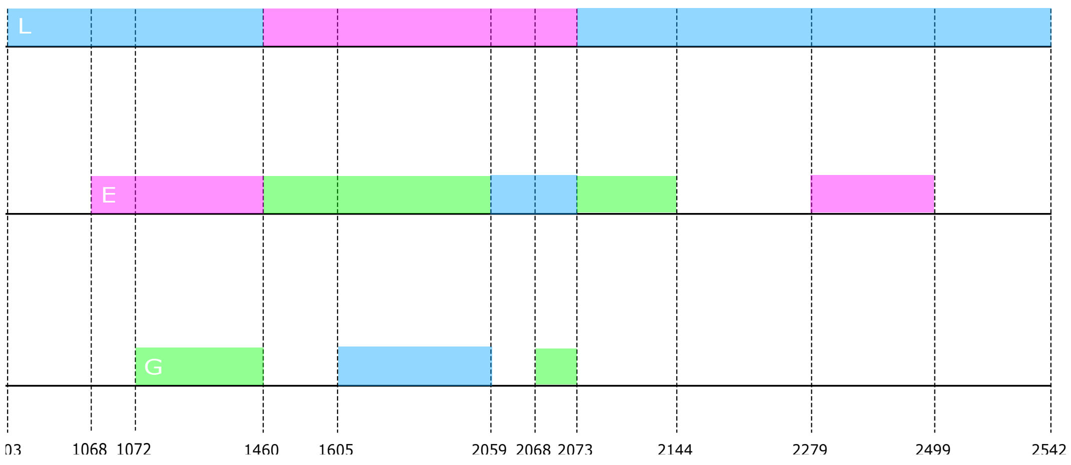


Figure 4.20: Timeline diagram of the video stream input. Each color represents a different person on the screen.

The algorithm is based on two fundamental matrices: the subject matrix introduced in the first part and the distance matrix.

The **distance matrix** itself represents the logic used to organize the flow of people and their identification. A specific function was created to generate the matrix. The function takes as input the current frame number, the input file cam_keypoints and the matrix with the number of subjects it calculates all possible possible average distances for all pairs of subjects in the current and previous frames 4.19.

By doing so it is possible to identify the pair with the shortest distance which will correspond to the same subject that is identified in the next frame. The distance matrix is a square matrix with a dimension corresponding to the maximum number of subjects detected between the previous and the current frame.

$$Dim = max(n_p, n_a) \tag{4.1}$$

If in the previous frame three subjects were detected and in the next frame two, the matrix will still be a 3x3 of which three elements will be null. This is to account for the fact that the subject has left the field of view. Same reasoning applies to a new entry. Below are the results obtained with the implementation of the sorting algorithm and the differences between before and after for each row of the input matrix of coordinates.

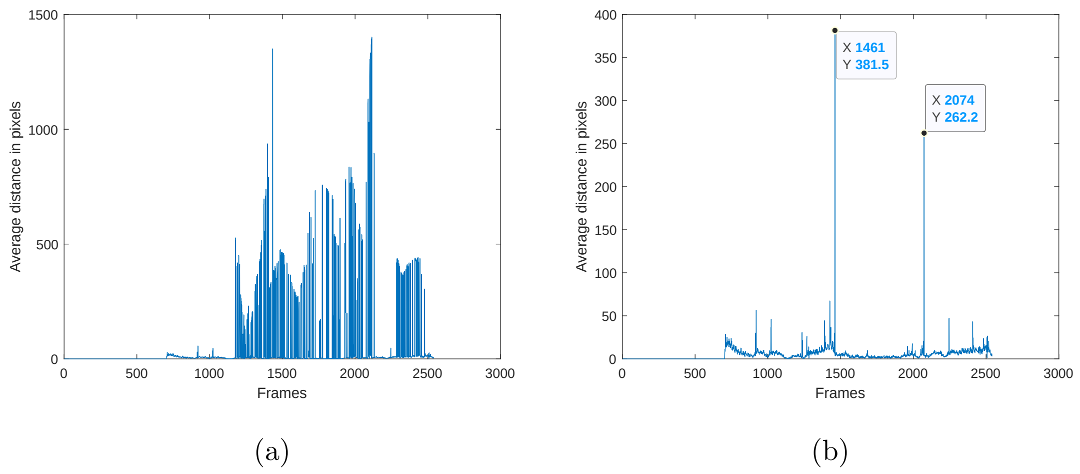
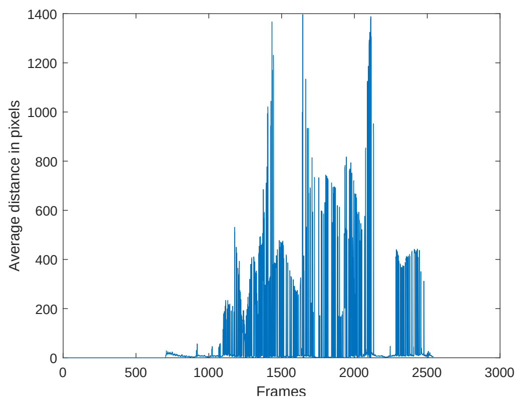
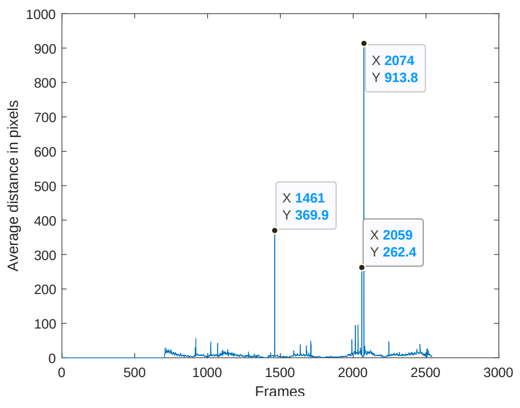


Figure 4.21: Average distances between subjects of the first row of the input file cam_keypoints. Before (a) and after (b) the application of ordering algorithm.

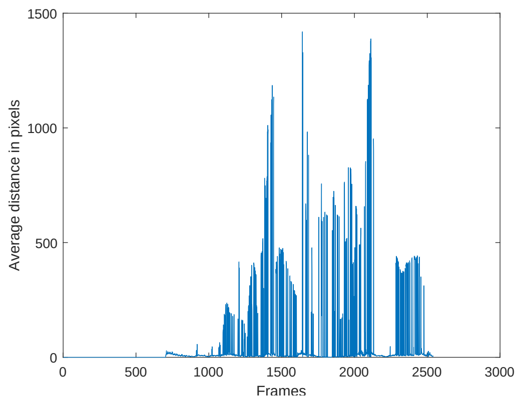


(a)

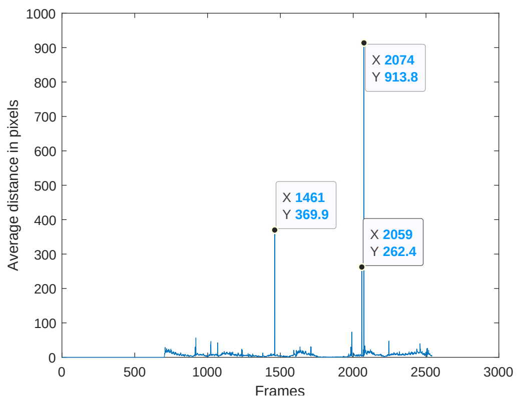


(b)

Figure 4.22: Average distances between subjects of the second row of the input file cam_keypoints. Before (a) and after (b) the application of ordering algorithm.



(a)



(b)

Figure 4.23: Average distances between subjects of the third row of the input file cam_keypoints. Before (a) and after (b) the application of ordering algorithm.

Sorting is considered complete when all the minima found within the distance matrix are on the main diagonal. And this must hold for all n-1 pairs between current and next frames. After the sorting procedure the distances are cut down and consequently the subject recorded on the first line of the file is maintained and therefore we can say with certainty that we are always the same person. It is important to point out one thing, when the subject leaves the screen there is a distance peak because on the same line a new subject is recorded. This can be seen very well in the diagram in the figures 4.20 and 4.21(b).

4.7 Conclusions and future work

These months of work have shown that it is possible to achieve human detection and identification with an on-edge device. It was also possible to do it inexpensively compared to a desktop computer with dedicated hardware. This latter can hardly be installed close to one or more surveillance cameras due to its size and cost related issues.

The target was reached with an NVIDIA designed Jetson Nano, a compact hardware optimized for computer vision and AI applications. Body stick detection was achieved through OpenPose and TrtPose at 4 and 10 fps respectively. A first attempt of identification was made with TrackLite that puts together YOLOv3 and DeepSORT to detect subjects with rectangles instead of body sticks and assign an ID to each of them. As can be seen from the paragraphs above, identification is a difficult task and IDs grow as quick as the number of subjects on screen get higher. In addition, the software is sensitive to the color of the shirt being worn as well as occlusions and overlaps.

In light of the limitations that emerged from the tests performed in the laboratory with the different software, it was decided to implement its own algorithm for identification. As recognition software was used OpenPose from which we obtained the file with the coordinates of the body sticks for each frame from the video stream. The identification algorithm is composed of three logics. The first consists in the identification of fake frames and their subsequent elimination. Fake frames are those frames in which the subject is detected in a partial and not continuous way. In particular, the detection is considered continuous if it continues for at least five consecutive frames. In addition, a phantom subject is also considered a fake if it is created as a result of a superimposition which in turn causes a doubling of identity in terms of coordinates.

The second part of the algorithm sorts the subjects in chronological order and prepares the file for further processing. In this way we are sure that from the moment the person appears until he leaves the screen the subject always occupies the same line of the file. All this to ensure continuity in the detection and have reliable data.

Third and last part of the algorithm performs what is the identification of subjects based on their lengths calculated from the body sticks. It then goes on to solve two types of problems found in the previous work of our colleagues. The distortion of the frame is handled with a maximum acceptable difference below which the frame together with its bodysticks is not considered valid. In addition, a check is made on the completeness of the coordinates through a minimum number of points. In this way we are sure that the detection is done on frames not hidden and without overlapping. These logics obviously imply limitations in their use since the results obtained are the first preliminary data and must be further tested and analyzed. The cases treated are particular situations and it is necessary to collect many more data samples on different types of video with different camera angles. It is hoped that this work can be a starting point for future colleagues in the field of human identification and tracking as well as in the design of on-edge systems capable of achieving the purpose.

Bibliography

[1] Artiom Basulto-Lantsova, Jose A Padilla-Medina, Francisco J Perez-Pinal, and Alejandro I Barranco-Gutierrez. Performance comparative of opencv template matching method on jetson tx2 and jetson nano developer kits. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0812–0816. IEEE, 2020.

[2] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2019.

[3] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[4] Ajay Katangur, Shih-Chun Lin, Jinpeng Wei, Shuhui Yang, and Liang-Jie Zhang. Edge computing–edge 2020. @, 2020.

[5] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[6] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

[7] Ahmet Ali Süzen, Burhan Duman, and Betül Şen. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5. IEEE, 2020.

[8] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. *Edge AI: Convergence of Edge Computing and Artificial Intelligence.* Springer Nature, 2020.

[9] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.