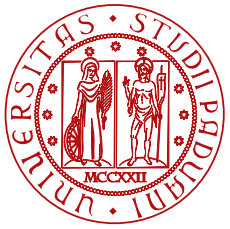


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Real-time multi-camera 3D human pose estimation on edge devices

Master Candidate

Emanuele Francesco Savoia

Student ID 2088248

Supervisor

Prof. Stefano Ghidoni

University of Padova

Co-supervisor

Dott. Matteo Terreran

University of Padova

Academic Year
2023/2024

Abstract

Human pose estimation is the process that aims to locate body parts and build human body representations from input data such as images and video. It is typically a computationally difficult operation, where, in order to achieve accurate results, the use of expensive GPUs is mandatory. Nowadays new use cases, such as augmented reality, demand to make this kind of operations viable on mobile and edge devices and research in other fields, such as human-robot collaboration, is leaning towards building portable and inexpensive solutions. This thesis describes the design and prototyping process of a real-time human pose estimation network made using edge devices, building a network using only Raspberry Pi boards for image processing, exploiting the TensorFlow lite (TFlite) library for running the necessary Deep Convolutional Neural Network (DCNN) components and utilizing the Robot Operating System (ROS2) framework to build a fast, real-time system. Additionally some techniques for creating DCNN that are capable of real-time execution will also be discussed and evaluated in order to try and surpass the hardware limitations imposed by the setup. A solution that archives real-time results with good accuracy is obtained with this work.

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

HPE Human Pose Estimation

DCNN Deep Convolutional Neural Network

ROS2 Robot Operating System

TFlite TensorFlow lite

PCK Percentage of Correct Keypoints

PCP Percentage of Correct Parts

MPJME Mean Per Joint Position Error

AP Average Precision

AR Average Recall

SOTA state-of-the-art

LLM Large Language Model

IMU Inertia Measurement Unit

Glossary

bounding box A set X,Y,W,H representing the coordinates and dimensions of the smallest box containing the desired object. 8, 24, 33, 34, 50

future An object that provides access to results of asynchronous operations. 33–35, 39

skeleton An ordered vector representing the 2D or 3D joints of a human in space. 1, 3, 5–7, 25, 34, 39

1 Introduction

1.1 Use cases for Human Pose Estimation

Human Pose Estimation (HPE) is the process that aims to locate body parts and build human body representations (usually represented in the form of skeletons) from input data such as images and video [28]. HPE can vary widely in form, from 2D or 3D representations to single-person or multi-person models each bringing it's costs and benefits trade off in terms of accuracy and real-time capabilities. As of today HPE is a tool that can be used in many different scenarios, each with different requirements and challenges.

The most promising application area is human-robot collaboration where, in order to make human operators and robots interact seamlessly, an accurate 3D real-time representation of the humans inside and near the robot workspace is necessary. Nowadays, this type of collaboration is usually made possible by utilizing a series of expensive high performance setups that relies on powerful GPUs to run inferences on state-of-the-art (SOTA) DCNNs. Tackling this problem using integrated boards with limited resources could open up new uses of this technologies even for applications with tighter budgets.

One of the newer applications of this technology is the augmented (and virtual) reality. This particular field aims at seamlessly fuse the real world with the digital by utilizing devices that can sense the real world and reproduce it on screen for the user with the addition of some new assets and, in some cases,

digital avatars representing real people visible to the user. For this application real-time capabilities are essentials in order to enhance the user experience and portability is becoming more and more important as the target platform of this field is gradually becoming the smartphone.

Other common applications of this field are found in fitness, where smartphone applications are being developed for evaluating one's mistakes or overall level when practicing sports, general healthcare, where the assessment of the physical capabilities of a patient is needed, and animation, where older mechanisms to obtain a reconstruction of the human body (such as reflective markers and Inertia Measurement Unit (IMU))

1.2 The objective

The current objective of this project is to prototype a system that can perform real-time HPE on edge devices and that could be used to perform human-robot collaboration or other similar tasks. In order to understand how to accomplish this objective it is important to know the main challenges of this endeavor.

3D: in order to be able to use this system with robots in the real world the human representation used in this project must account for three-dimensional positions of the joints. Unfortunately this kind of task is extremely computationally intensive when it's performed on single images, in order to obtain reliable 3D data another solution must be found. Ideally a solution where inferences are computed in 2D and then the 3D solution is achieved would be better for dealing with low-power devices.

Occlusion: a system that is used in dangerous tasks, such as human-robot collaboration, needs to be resistant to occlusions. A single camera system would surely not be enough, especially when the workspace is occupied by moving objects that can completely occlude people in the frame.

A multi-camera setup would alleviate both of these problems since it is possible to obtain the 3D joint locations data from multiple 2D estimations performed on images obtained from different views and, thanks to the multiple point of views, the probability of occlusion is extremely reduced.

Knowing this, it is possible to affirm that this work aims to study HPE and it's possible applications in real-time embedded systems, the objective of the proposed prototype will be to generate the 3D skeleton of a single person. In order to reach this objective a network of Raspberry Pis with associated cameras will be employed to acquire 2D image data and to run the necessary neural networks. By leveraging edge devices as primary processing devices, this work aims to demonstrate that HPE can be executed effectively even on a low budget, possibly making this kind of technology more accessible removing the high-performance hardware limitations.

2

Human Pose Estimation

2.1 Evolution of human representations

In this section some of the methods [9] used for representing human poses are exposed and discussed. While most of the current research is focused on skeletons, some of these unconventional methods have been used and researched as late as 2016.

2.1.1 Pictorial Structures and Flexible Mixture-of-Parts

The pictorial structures, method that was first proposed in [3], consists in representing the human body as a set of parts organized in a deformable structure [10] where each part is connected to it's neighbors in a tree-like fashion by the use of virtual springs in order to encode spatial relationships between parts. Each part is represented using a set of features that are matched against the input image and the spatial encoding in order to retrieve the 2D positions of the body parts. This model used to be very efficient, but unfortunately it proved to be weak to occlusion and diverse poses, leading to the development of new more robust models.

The Flexible Mixture-of-Parts method builds on the concepts of the previous one, representing the human body by transforming parts into mixtures. These objects were made to represent the sets of different possible configurations that could be found in a body part in order to be able to represent it in many different

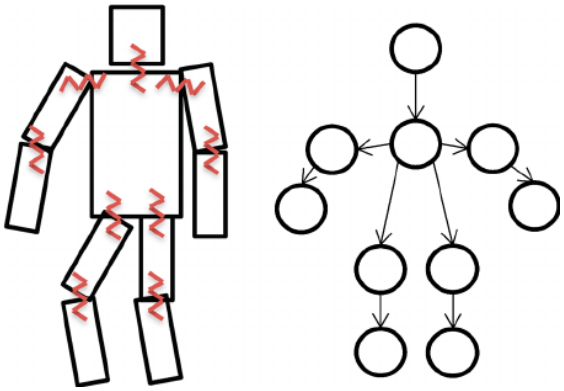


Figure 2.1: Pictorial Structures example

configurations and from different views [21]. This method was an improvement on it's predecessor, but challenges due to occlusion and extreme poses still persisted.

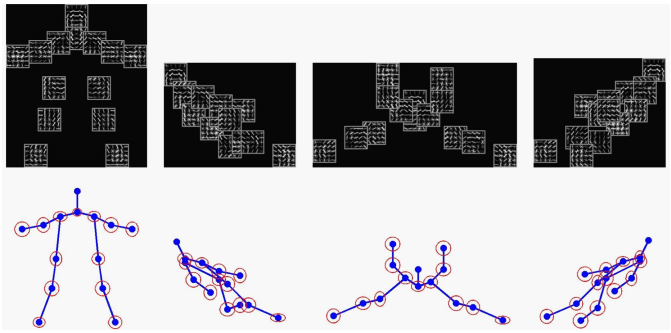


Figure 2.2: Flexible Mixture-of-Parts example

2.1.2 Skeletons

The skeleton is the preferred way to represent a human pose at this time. There are many different conventions used, but all of them can be defined as an ordered set of vectors where each vector represents a specific body joint. The extreme variety in skeleton conventions adds some difficulties when dealing with DCNN training, since usually one must be chosen as the network representation and each dataset provides a very small variety of annotation conventions to choose from. Fortunately, if different datasets are needed during training, the authors of the MeTRAbs DCNN published a promising paper [16] that details the steps they used to train their network to output multiple different skeleton formats for the same detection using a geometry-aware autoencoder 2.3.

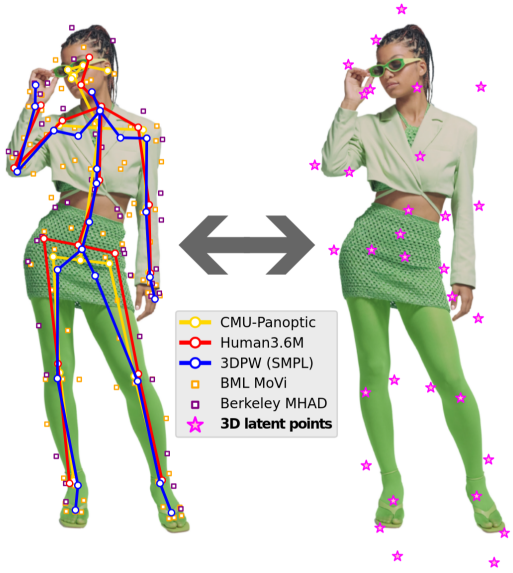


Figure 2.3: Different skeleton models and MeTRAbs autoencoder solution

2.2 Approaches

This section provides an overview of the key differences in approaches commonly used in modern skeleton-based HPE applications. HPE evolved a lot since it's inception, in the beginning the field was dominated by approaches leveraging image features, while today's standards for performing HPE are DCNNs.

State of the art DCNNs are fast, versatile and resistant to most of the old methods' weaknesses such as strange poses and light occlusions. These new advancements are possible due to the sheer volume of data that is now available to research in the form of public data (COCO [11], H3.6M [7], etc...) and due to the vast improvements in computational power in the latest years.

There are many differences among different HPE DCNNs, but most of today's methods are typically categorized based on three main variables: the maximum number of subjects that can be in the input image, the dimensions of the joint vector representations and the method used to identify joints.

2.2.1 Single-person and multi-person

Different DCNNs are suited to different tasks, single person approaches the HPE problem by assuming that only one person is in the image frame at the time of inference. This leads to higher efficiency and speeds than it's counterpart,

but this approach fails to maintain a high accuracy when more people enter the frame since the network can't usually distinguish different subjects. This kind of approach can be easily combined with a people detector (an object detection model trained to estimate the bounding boxes of people [29]) to generate a top-down multi-person HPE pipeline. Today the research on this side of HPE is currently swaying towards lightning-fast networks designed to run on low-power devices with projects such as RTMPose [8] while research on multi-person HPE is focusing on more complex networks that exploit new architectures such as transformers [26].

Multi person DCNNs, on the other hand, are built for the purpose of detecting multiple people in one frame. This more difficult task is often archived at the expense of network complexity, real-time performance and accuracy when handling overlaps. The added complexity is often handled with two different approaches to the problem: top-down and bottom-up. While top-down methods prioritize detecting individual persons first and then estimating their joints' poses, bottom-up approaches invert this order by detecting all keypoints first and grouping them into individual poses afterward.

The top down methods discover the people in the image first, typically framing them in bounding boxes with specially trained object detectors [29] designed to detect people and passing images cropped accordingly as inputs for the HPE model that will estimate a set of keypoints in each bounding box. The main advantage of this approach is that, since bounding boxes are formed around people, it is possible to "zoom in" while evaluating the HPE inference since the DCNN input is usually of fixed size and fitting a smaller cropped area in the same number of pixels would for sure let the DCNN work with finer details. This opportunity raises the overall accuracy of this methods and grants them an advantage when dealing with overlapping individuals, while the nature of performing one HPE invocation for each person in the image makes the speed performance of this approach directly dependent on the number of people. In image 2.4 it is easy to see how, by combining the results of the inferences performed by the HPE network, the multi-person output is obtained.

The bottom-up approach, on the other hand, is composed of two main steps as shown in figure 2.4. The first part of the pipeline is a body joint detector and it's

purpose is to predict all the body joint candidates in the input image, while the second part is a body part association mechanism that assigns each predicted joint to it's person's set. While the first task is usually approached with DCNNs the second can be solved by using other algorithms or methodologies such as Integer Linear Programming for DeepCut [15]. This approach is much faster than it's counterpart (especially when a large number of people is involved in the detection), but it lacks in accuracy since the precision is directly tied to how well the persons in the frame are distinguishable in an usually down scaled version of the original input image frame.

Figure 2.4: Top-down and bottom-up approaches

2.2.2 Type of representation

The first thing that comes to mind when thinking about HPE is the fact that, despite having two dimensional data such as images as input, some algorithms and DCNNs are able to reconstruct the three dimensional pose of the humans in the camera frame while some others are not.

2D HPE focuses on retrieving the joints information with respect to the image frame. These methods are usually much faster than 3D ones and much less complex, but they lack usability in tasks requiring depth information. Sometimes these methods can be used as a stepping stone to get to the 3D poses with techniques such as 2D to 3D lifting or by leveraging multi-camera setups.

While most of the current approaches to HPE focus on single camera inference, some research revolves around new ways to improve multi-camera setups. This approach grants some robustness when dealing with occlusions and other possible challenges since even if, for example, a person is occluded in one camera's frame, it probably is visible enough in some others, so that it is possible to weight the single camera results and get an overall better HPE result. Multi-camera setup are commonly used to perform the 3D triangulation of the HPE results from 2D (or 3D) skeletons generated by multiple cameras and perspectives. Even though most multi-camera setups are just composed by a set of DCNNs that operates on a single image and some kind of algorithm that generates a final prediction by combining the single DCNNs' estimates, some other networks like VoxelPose [19] use directly the multiple views as their input and directly reasons in 3D space.

Single-image 3D HPE, on the other hand, is slower and more computationally demanding than it's counterparts since, on top of the 2D approach's workload, there must be mechanisms to extract the depth information from a 2D image in order to pinpoint 3D joint coordinates. These mechanism can be built by leveraging hardware such as depth cameras or by adding additional computational complexity to the underlying DCNN. Depth cameras are sometimes used in order to gain some additional information to generate a better 3D prediction and to better separate the people from the background, unfortunately this approach has some great limitations. Since depth cameras are usually very sensitive to sunlight, this approach is usually only used for indoor estimation and, even indoors, depth sensors produce lots of noise making them unreliable for critical applications. The main advantages of these methods is the real-world coordinates output. These are essential for tasks involving interaction between humans and machines and to find and suppress implausible poses when precision is needed.

2.2.3 Regression and heatmap methods

When dealing with joint representation, it is important to distinguish between the two main ways to actually extract keypoints from an image: regression and heatmap-based methods.

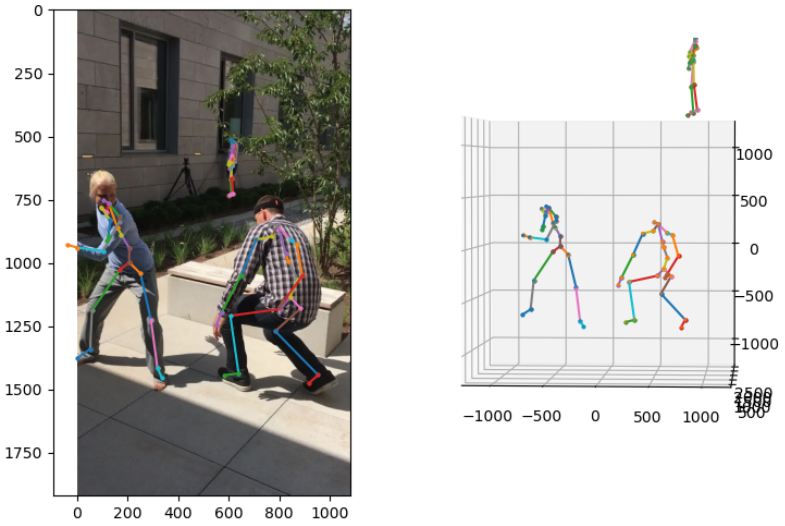


Figure 2.5: 2D and 3D example (made with MeTRAbs DCNN)

Regression-based methods are simpler than the heatmap-based ones as they are more straightforward. These methods directly predict the joint coordinates from images without intermediate steps. This usually means that they are faster than their counterparts and are better suited for real-time tasks. An example of this approach can be found in the DCNN used in the ROS2 network prototype that will be introduced in the following chapters.

Heatmap-based methods, on the other hand, add an additional step with respect to the regression-based ones. Instead of estimating directly the joint coordinates these methods add an intermediate step where 2D Gaussian kernels are generated on each join location. These heatmaps represent the probability that the corresponding keypoint lies in a certain position and their ground-truths are generated by centering 2D Gaussians at the keypoint ground truth location [28].

These methods are commonly used when performing multi-person bottom-up HPE in order to estimate the single joint coordinates before performing body part association like in MeTRAbs [17].

2.3 Challenges

This section will expose some of the most relevant challenges that arises when dealing with HPE and some possible ways used to overcome them.

2.3.1 Occlusion

Occlusion is one of the most well-known challenge in the field and it refers to the scenario where some parts of the person on which HPE needs to be performed is not entirely visible from one or more points of view (cameras). This has some notable impacts in single-camera scenarios as it is impossible to know for sure the positions of the occluded joints, but some networks like "HUPOR" [12] are built from the ground up with the intention of becoming resistant to occlusions by correctly guessing the poses of heavily occluded joints with specific mechanisms and training. In this case the network, after finding all the visible keypoints, performs an "occluded keypoints reasoning" step that infers the occluded keypoints' heatmap from visible cues. This step's training is done by utilizing another network to predict the occluded keypoints' map from the dataset's keypoints and using that heatmap as the ground truth for training this step. On the other hand, this phenomenon is more manageable in a multi-camera scenario where models can be trained to assign lower confidence scores to uncertain joints and a voting or weight system can be implemented in order to try and overcome the occlusions in some frames (note that if the same joint is occluded in every frame the problem still persist). This is how this particular problem was managed in this thesis work.

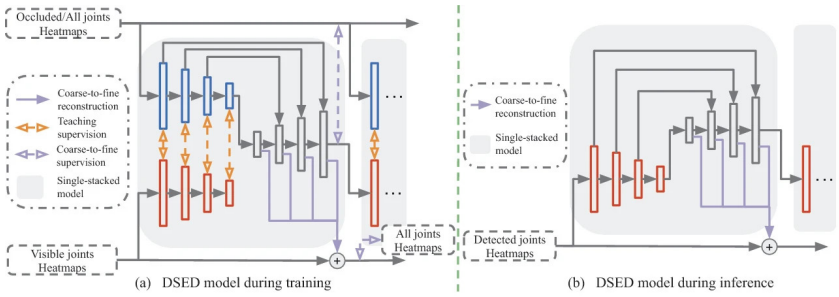


Figure 2.6: HUPOR occlusion training model

2.3.2 Unusual poses

Another great challenge when dealing with HPE is the difficulty for most models to correctly estimate "unusual" poses. These poses often include upside-down people or convoluted positions derived from yoga or martial arts. This difficulty is mostly due to the nature of the datasets used during training since most of the images and videos used contain people performing common actions (such as standing, sitting, walking ...) and lack some proper examples for other kinds of actions such as fitness related moves (handstands, yoga poses, etc...). Some particular networks are trained to cope with some unusual poses by modifying the dataset, for example the movenet network by google that was used in this work is trained with "fitness" related poses, making it less susceptible to this problem.

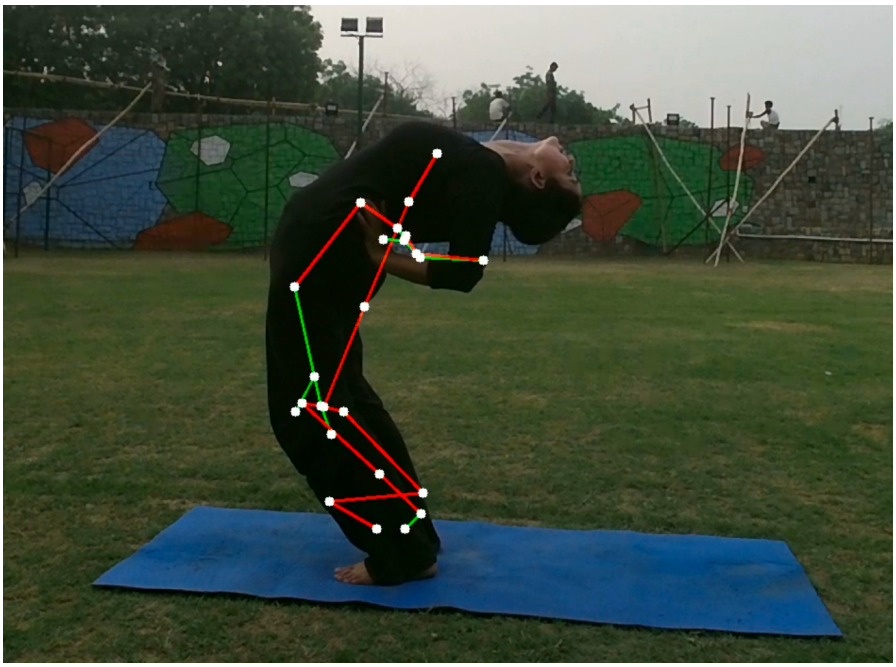


Figure 2.7: Unusual pose example

2.3.3 Depth ambiguities

In 3D HPE the image medium inherently lacks depth information and the same 2D pose can be represented by a set of possible 3D poses [25]. These conditions lead to a field of research in HPE that deals with finding the best methods to find the correct 3D interpretations of 2D poses. As of today various

of these methods are used for dealing with this particular problem, some of the most relevant are temporal and kinematic constraints.

Temporal Consistency Constraints: these methods involve the use of videos because of the additional time information. This additional information is useful to the DCNNs since the context information gained from close frames can give clues about the depth information.

Kinematic Constraints: these methods model the human body as a set of restrictions (bone lengths, joint angles, etc...) in order to easily suppress or penalize impossible or implausible poses.

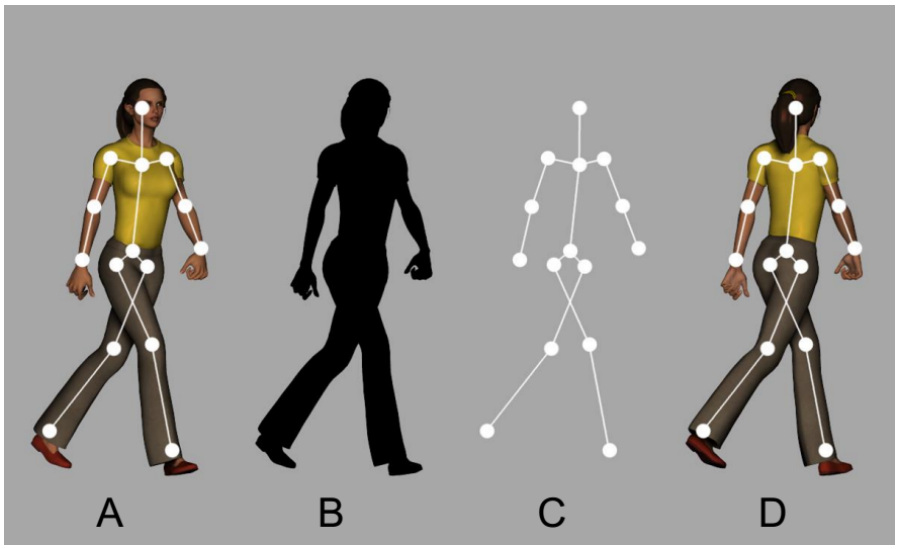


Figure 2.8: Depth ambiguity example

2.4 Metrics

In the deep learning metrics are essentials to quantify networks' performance, in the HPE field these are some of the most used metric used to evaluate DCNNs [2]:

Percentage of Correct Parts (PCP): this metric measures the localization accuracy for limbs. A limb is deemed correctly localized if it's two endpoints (joints) are within a certain threshold from the ground truth's endpoints. The most commonly used threshold is 50% of the total bone length, but other thresholds dependent on other variables (bounding boxes, pixel count, etc...) might be used.

Percentage of Correct Keypoints (PCK): this metric is very similar to PCP, but it measures the accuracy for joint estimates instead of the whole limb. In this case a joint estimation is deemed correct if it falls within a threshold from it's ground truth. This threshold can also be calculated in different ways based on the same variables used for PCP.

Mean Per Joint Position Error (MPJME): this is the most widely used metric to evaluate 3D HPE performance. It represents the average euclidean distance of a computed joint J_i to it's ground truth J_I^*.

$$MPJME = \frac{1}{N}\sum_{i=1}^{N} \| J_i - J_i^* \| \tag{2.1}$$

Average Precision (AP) and Average Recall (AR): when computing these common metrics, a joint prediction that is defined as "true positive" falls within a certain threshold from the ground truth (as in PCK), this means of course that a prediction that is outside this threshold is a "true negative" while a "false positive" is any unassigned prediction.

For evaluating the ROS2 network the PCK and MPJME metrics were used.

2.5 Recent advancements and trends

In the recent years many new innovative ideas and SOTA architectures that can benefit overall HPE were introduced. Among these, we can, for sure, see a rise in transformer-based architectures that can easily capture temporal relationships across the input frames as well as the actual human body joints, generating an overall smoother detection. This architecture was first applied in the PoseFormer [27] network and then expanded in it's second iteration [26].

With the current success of Large Language Model (LLM) architectures some researches have begun studying their application to HPE [14] by integrating their semantic understanding with geometric reasoning. An example of this new research field can be seen in SCALE-Pose [13], a model that leverages both the previously disused transformer based architecture and the newer LLM addition. Another interesting approach to LLM based estimation can be found in LocLLM [20]. This model employs the LLM to generate never-before-seen keypoints by using their description as input along as the image where to locate them.

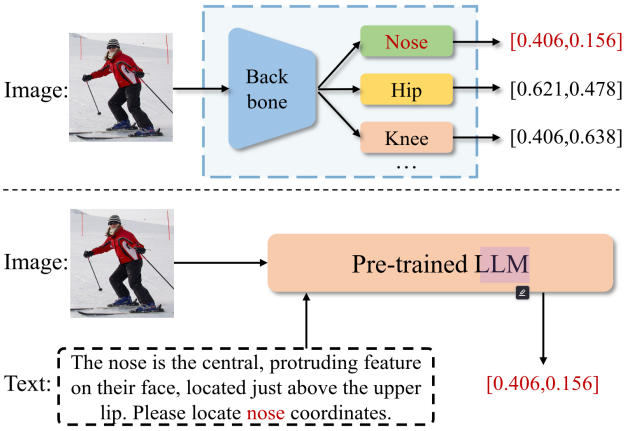


Figure 2.9: LocLLM example

2.6 Edge devices and real time HPE

SOTA architectures, exposed in the previous section are surely too computationally demanding for working on mobile or edge devices and, since as of today real-time mobile HPE is becoming increasingly important for consumer use, research on this topic is rapidly growing. The key challenge in this field is designing models that are both fast, accurate and light enough so that they are able to run with limited resources available maintaining a good frame rate.

An example of these novel architectures is MovePose [22], a MobileNet [5] based architecture developed specifically for edge devices. This particular DCNN produces results with accuracy comparable to current SOTA architectures in a fraction of the time. This feat is made possible by using DARK [23], a model-agnostic method that improves precision when extracting keypoints from a heatmap. Another interesting approach to creating DCNNs is the one proposed with MoVNect [6]. In this paper the researchers create a light and fast network by performing knowledge-distillation with a bigger pretrained network as teacher. In order to obtain this result researchers created a novel mimicry loss function that encapsulates both the heatmap and keypoints losses with respect to both teacher and ground-truth.

Another approach to this problem would be designing algorithms outside of the main HPE DCNNs and using them in order to boost performance. In [18] the authors explain how they achieved a real-time 3D HPE pipeline by utilizing TFlite optimizations and generative adversarial networks to train the 2D to 3D lifting network used. While the optimizations are pretty common in this field, the idea of utilizing generative adversarial networks to boost the performance of the lifting process is extremely innovative. This approach works by training a network to distinguish between real and generated 2D poses and by using it's output as the loss function for training the depth generator that generates z coordinates with respect to the camera frame for every 2D keypoint detected by the underlying HPE DCNN, since the "generated" that are fed to the discriminator are 2D projections of the rotated 3D poses generated by the depth generator. This approach reaches real time with up to 39 FPS on the Google Coral board for it's single-person single-image 3D HPE pipeline, unfortunately the paper's authors don't provide accuracy measures but they do provide some

impressive images taken from the pipeline output 2.10.

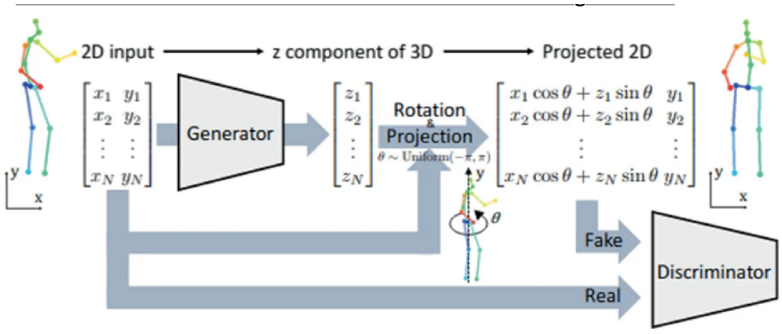


Figure 2.10: Generative adversarial networks for 3D lifting

Another interesting proposal would be to find ways to skip unnecessary HPE inferences in order to use less resources and gain better frame rates. Researchers that worked on MobiPose [24] proposed to use motion vectors to track persons across frames. This approach makes sure that the underlying DCNN inference isn't necessary for every frame, reducing the average time between frames by leveraging the faster nature of motion-vector-based tracking. This approach results in up to 80 FPS on mobile GPUs while maintaining over 80% accuracy.

3

Design choices for multi-camera HPE

3.1 Hardware choices

Selecting the right hardware when creating AI powered projects is a critical step in development, especially for single board computers and other edge devices. An uninformed choice could spell the end for a project if the maintainer is not adequately prepared, that is why, in order to build the edge network, these constraints were formulated for the hardware selection:

Budget: it was decided to focus on edge devices in order to provide some insights on performing HPE even on a tight budget, this is why it was decided to spend as little as possible when selecting the main computing units for this project.

No powerful GPU: the first constraint was to not use a device with a powerful dedicated GPU. Even if as of today GPUs are almost ubiquitous in modern devices it was decided not to use any in order to make this endeavor a true challenge and test the limits of CPU-only and low-power-GPU DCNN inference.

Multi core CPU: in order to try and archive real time with little amounts of GPU parallelization (or without it in it's entirety), it is necessary to be equipped with a multi core CPU, in order to be able to address communication between devices and inferences in parallel. Of course a high clock speed is a nice bonus

to consider when selecting a device for this type of projects and it was taken into account.

Maintainability: when selecting a device it was crucial to have an easily maintainable device, possibly with a known system on board in order to reduce the configuration time to the bare minimum and to quickly recover when some software inevitably fails.

The Raspberry platform was chosen as the core of this project because of it's low cost compared to it's computational power and because it runs on Linux (more specifically Ubuntu 24.04 since this summer). It was chosen to use a set made by Raspberry Pi 4 and Pi 5 in order to evaluate the differences in performance between the two different models. Both versions of the boards are equipped with 8 GB of RAM and the server version of Ubuntu 24.04 LTS (in order to reduce unnecessary software and try to improve performance) as well as 64 GB of SD memory and they are connected to a RealSense Stereo Camera D455. Unfortunately, despite our best efforts, it was not possible to use the dedicated GPU in order to speed up DCNNs computations on the boards for unknown reasons related to OpenCL and graphic drivers.

Another option that was discussed was the NVIDIA Jetson Nano board that is currently widely used for low-power AI applications since it possesses a very powerful on board GPU. This, of course makes this board and it's predecessors widely used in edge device AI research, but it falls out of the scope of this thesis project by having such powerful hardware. A similar product sometimes used in edge AI is google's Coral board, but for the same reasons as the Jetson it falls out of the scope of this work and was not used in the end. It is worth to know that the Coral USB accelerator is another powerful tool that can be used in order to perform DCNN inference on most edge devices. This could be explored in future works.

The last hardware piece that was considered in the final months of this work is the new Raspberry AI camera. Unfortunately it was not possible to use it in our tests, but it would for sure be a fine addition for future developments. This new piece of technology holds a powerful AI accelerator inside that directly runs

various DCNNs out of the box, such as one made for people detection and one for HPE, both of these would be very useful for the project.

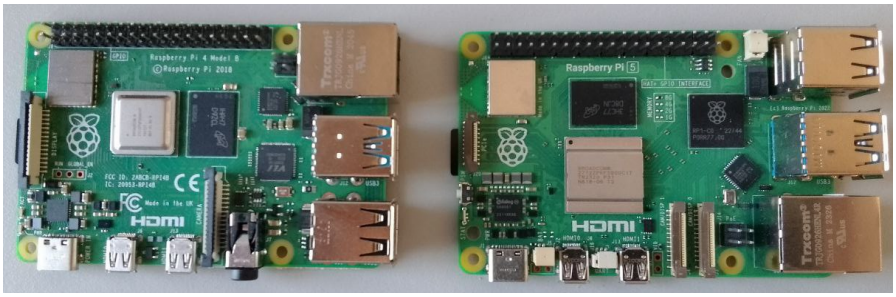


Figure 3.1: Raspberry pi 4 and 5

Figure 3.2: Nvidia Jetson Nano

Feature	Raspberry Pi 4	Raspberry Pi 5	NVIDIA Jetson Nano
GPU speed	32 GFLOPS	51 GFLOPS	512 GFLOPS
CPU cores	4	4	4
CPU speed	1.8 GHz	2.4 GHz	1.43 GHz
Price	80€	100€	250€ +

Table 3.1: Performance and cost of different boards

3.2 Frameworks choices

In this project we relied on the ROS2 framework (more specifically the jammy release) for handling the task scheduling and communication across various systems. This choice was made because of many factors:

Compatibility: this piece of software is out of the box compatible with Ubuntu 24.10 LTS and, thanks to it's nature, it provides easy scalability and cross-platform compatibility. The compatibility issue is mentioned because prior to the release of Ubuntu 24.10 using ROS2 on a Raspberry Pi 5 was a particularly "hacky" and tedious task since often critical required packages were missing from the Raspberry OS' packages list, while it was the only possible OS choice for the time and the only way to install it natively was a very slow build process that could take up to 3 or 4 hours and fail at any time for missing dependencies.

Real-time capability: ROS2 is often used in real-time applications, such as robot navigation and control, this is because it's communication systems (such as messages, services and actions) are fast and reliable and most of all highly configurable and adaptable to new situations. In this project, messages and asynchronous services played a crucial role in establishing a fast and efficient communication network.

Data acquisition and visualization: the rviz2 tool provide a great interface for visualizing 2D and 3D data, such as camera feed and 3D markers positions and, thanks to the bag package recording data for testing new program pieces becomes trivial and this greatly speeds up development and helps with qualitative and quantitative analysis.

Community and experience: of course previous experience with the ROS1 framework proved to be essential for swift development and the community forums are of great help, especially since the overall documentation is often lacking and overlooks some crucial details.

The TFlite [1] framework was used for running the DCNN inferences efficiently on the Raspberry Pi boards. This choice was made despite the current limitations on supported operations imposed by the TFlite library because of these advantages:

Edge optimizations: the shared C++ libraries can be compiled directly on ARM boards with a great deal of optimizations with respect to the base version of TensorFlow. Some of these optimizations are given by delegates are optimized for a different platform (or set of platforms), the GPU delegate on the other hand supports offloading computations to on board GPUs on most devices. TFlite differs from it's parent library TensorFlow also for it's model representation: as a matter of fact TFlite utilizes static graph execution, this feature reduces the overheads due to changes in the DCNN at runtime by sacrificing some versatility. In this work the XNNPACK delegate was used for CPU inferences, while the GPU delegate is available to be used on other machines it does not work with the current Raspberry setup as some drivers are not working as intended.

Model optimizations: the framework provides many different options for optimizing existing DCNNs with operations, such as quantization and pruning.

Compatibility: the TFlite framework supports converting models to the ".tflite" format from various different file types other than TensorFlow, such as PyTorch and the interpreter is available for multiple different embedded devices, such as single board Linux computers like the Raspberry Pi, but also microcontrollers such as Arduino.

[1] Please note that at the time of writing the TFlite project is migrating to LiteRT, this means that the installation guides provided in the project repository might not be up to date in the future.

3.3 Neural networks used

In this project two main networks were used: the first one handles single person 2D HPE and the other handles people detection. This section briefly discusses the two DCNNs.

3.3.1 Human pose estimation

The DCNNs that were chosen to perform 2D HPE from the worker node are derived from google's MobileNet [5] family and are called MoveNet. These models are built to be used with the TFlite framework for tasks on edge devices and are designed to be robust when detecting fast or difficult movements and to reject people that are closer to the frame border, making them perfect to use concurrently with the detector DCNN since it will better reject keypoints on the bounding boxes' edge that probably belong to another person.

Two versions of this network were used: thunder and lightning. Both versions are quantized and they differ in input size, lightning is the faster version of the model, built for performance-critical applications with a 192x192 input size, while thunder was made for obtaining higher accuracy at the cost of some speed with it's 224x244 input size. Both networks are built for single-person HPE, but a multi-person variant exists, even if it was not tested in this project.

3.3.2 People detector

The network used for people detection is based on the YOLO architecture and it was provided by as a pre-trained TFlite model by a GitHub repository *DoranLyong/yolov4-tiny-tflite-for-person-detection*. The YOLO architecture [1] was chosen as it is one of the fastest available for this kind of tasks for state of the art results. While the model utilized is based on YOLOv4, a possible improvement for this project would be building a people detector with a newer (and faster) YOLO version as the base for building the network.

3.4 The triangulation

In order to accurately estimate the real-world 3D joint coordinates from the 2D ones obtained by DCNN inference the triangulation algorithm proposed in [4] was used. This method consists on solving a simple weighted least square problem of form

$$x_j = [A_j^T W_j A_j]^{-1} A_j^T W_j B_j \tag{3.1}$$

for each joint in the HPE model's skeleton.

The A and B matrices in the previous equations are directly derived from the camera projection matrices

$$M_c = I_c {}^W T_{C_c}^{-1} \tag{3.2}$$

that are obtained by the calibration information given by the slave nodes' calibration service since I_c and ${}^W T_{C_c}^{-1}$ are respectively the intrinsic matrix and the extrinsic matrix of the c-th camera.

Assuming that the vector $p_{c,j}$ represents the coordinates of the j-th joint on the c-th camera's frame and that ${m_i}^{(k,h)}$ represents the element of matrix M_i at row k and column h, then the matrices A_j and B_j will be of form

$$A_j = \begin{bmatrix} {p_{1,j}}^{(1)}({m_1}^{(3,1)} - {m_1}^{(1,1)}) & {p_{1,j}}^{(1)}({m_1}^{(3,2)} - {m_1}^{(1,2)}) & {p_{1,j}}^{(1)}({m_1}^{(3,3)} - {m_1}^{(1,3)}) \\ {p_{1,j}}^{(2)}({m_1}^{(3,1)} - {m_1}^{(2,1)}) & {p_{1,j}}^{(2)}({m_1}^{(3,2)} - {m_1}^{(2,2)}) & {p_{1,j}}^{(2)}({m_1}^{(3,3)} - {m_1}^{(2,3)}) \\ \vdots & \vdots & \vdots \\ {p_{i,j}}^{()}({m_i}^{(3,1)} - {m_i}^{(1,1)}) & {p_{i,j}}^{(1)}({m_i}^{(3,2)} - {m_i}^{(1,2)}) & {p_{i,j}}^{(1)}({m_i}^{(3,3)} - {m_i}^{(1,3)}) \\ {p_{i,j}}^{(2)}({m_i}^{(3,1)} - {m_i}^{(2,1)}) & {p_{i,j}}^{(2)}({m_i}^{(3,2)} - {m_i}^{(2,2)}) & {p_{i,j}}^{(2)}({m_i}^{(3,3)} - {m_i}^{(2,3)}) \\ \vdots & \vdots & \vdots \\ {p_{c,j}}^{(1)}({m_c}^{(3,1)} - {m_c}^{(1,1)}) & {p_{c,j}}^{(1)}({m_c}^{(3,2)} - {m_c}^{(1,2)}) & {p_{c,j}}^{(1)}({m_c}^{(3,3)} - {m_c}^{(1,3)}) \\ {p_{c,j}}^{(2)}({m_c}^{(3,1)} - {m_c}^{(2,1)}) & {p_{c,j}}^{(2)}({m_c}^{(3,2)} - {m_c}^{(2,2)}) & {p_{c,j}}^{(2)}({m_c}^{(3,3)} - {m_c}^{(2,3)}) \end{bmatrix} \tag{3.3}$$

$$B_j = \begin{bmatrix} p_{1,j}^{(1)}(m_1^{(1,4)} - m_1(3,4)) \\ p_{1,j}^{(2)}(m_1^{(2,4)} - m_1(3,4)) \\ \vdots \\ p_{i,j}^{(1)}(m_i^{(1,4)} - m_i^{(3,4)}) \\ p_{i,j}^{(2)}(m_i^{(2,4)} - m_i^{(3,4)}) \\ \vdots \\ p_{c,j}^{(1)}(m_c^{(1,4)} - m_c^{(3,4)}) \\ p_{c,j}^{(2)}(m_c^{(2,4)} - m_c^{(3,4)}) \end{bmatrix} \tag{3.4}$$

The weight matrix W_j is instead a square diagonal matrix of form

$$W_j = diag\left(\begin{bmatrix} w_{1,j} & 0 \\ 0 & w_{1,j} \end{bmatrix}, \ldots, \begin{bmatrix} w_{i,j} & 0 \\ 0 & w_{i,j} \end{bmatrix}, \ldots, \begin{bmatrix} w_{c,j} & 0 \\ 0 & w_{c,j} \end{bmatrix}\right) \tag{3.5}$$

where $w_{c,j}$ is the weight associated to the camera c and joint j. This equation represents the way of calculating this value proposed by the [4] paper's authors

$$w_{c,j} = w_{c,j}^s \left(\frac{w_{c,j}^o + w_{c,j}^d}{2}\right) \tag{3.6}$$

In this equation $w_{c,j}^s$ represents the square of the considered joint's confidence score given with respect to camera c's perspective, while $w_{c,j}^d$ weights the distance of the joint's distance to the same camera. The last value is the orthogonality index $w_{c,j}^o$ and it is used to weight the orientation of the body with respect to the camera frame, giving a higher weight to 2D joints generated by cameras that the person is directly facing.

In our implementation we used a sightly different equation to generate weights:

$$w_{c,j} = w_{c,j}^s * w_{c,j}^c \tag{3.7}$$

The only variations is that the orthogonality index as proposed in the paper has been removed since, for the time being, records of the skeletons are not kept by the system. Following the insights proposed by the paper's authors low confidences (< 0.4) further penalize the weight assignments by squaring them, but not removing them totally as it was proposed.

3.5 Neural network optimizations

In order to try and build an overall better system, two main ways to generate light DCNNs were tried, in this section the attempts are exposed.

3.5.1 MeTRAbs lite

The first thing that was attempted when trying to create a better network was creating a TFlite compatible DCNN based on a SOTA one. The chosen network was MeTRAbs [17], unfortunately this endeavor proved to be too difficult to actively pursue since the network utilizes unsupported TensorFlow operations and other totally custom operations. In order to provide fast inference times the TFlite framework only officially supports a subset of all Tensor Flow operations and does not support any custom operation 3.3. These operations would have needed to be implemented in C and compiled with the TFlite library in order to work. After noticing that many other SOTA approaches needed this kind of support in order to be used it was decided to try and develop a custom network using knowledge distillation.

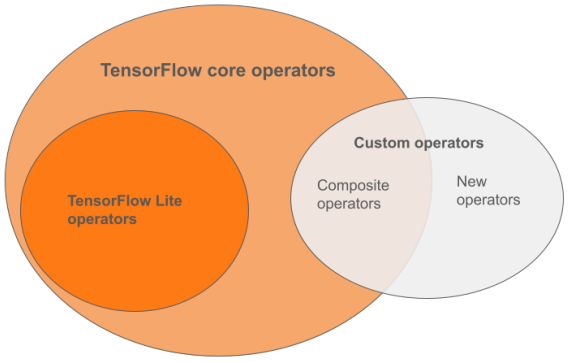


Figure 3.3: Supported TFlite operations diagram

3.5.2 Knowledge distillation

Following the paper "Lightweight 3D Human Pose Estimation Network Training Using Teacher-Student Learning"[6] a lightweight version of the MeTRAbs [17] network was built and trained using the method proposed in [6] by exploiting another sightly modified version of the MeTRAbs DCNN as the teacher

network in order to be able to access the heatmap data. The H36M dataset was used for this experiment, unfortunately, due to time constraints, it was not possible to use the whole dataset to train the model and a very small number of epochs was used as well. This endeavor proved to be unsuccessful, with a PCK value of under 0.2 when using a threshold of 300mm but might be a great starting point for expanding this work.

A big reason for this failure (other than the time and dataset constraints) might lie in the actual network architecture that was used for the student network since it was based on the paper's [6] network with some modifications to make it more similar to MeTRAbs.

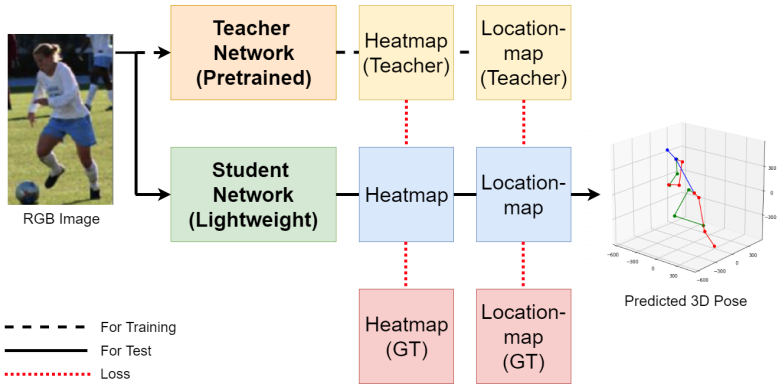


Figure 3.4: Knowledge Distillation example

4

The network architecture

4.1 General architecture

In this chapter the architecture of the network will be discussed and the reasons for the main choices will be explained as long as some of the iterations needed to reach the final product. The complete final repository can be found and tested here at: `https://github.com/TheSav1101/ros2-ws/`.

The network, from an hardware perspective, consists of four Raspberry Pi boards (divided in two Pi 4 boards and two Pi 5) and a master PC all connected to the same local network for communication. Each Raspberry is also connected to a calibrated RealSense camera and it's calibration information are either stored in a json file or sent as calibration messages. All the available DCNNs in ".tflite" format are stored on the devices and the information needed to configure the TFlite interpreters for different models are (for the time being) hard coded in the project.

From a ROS2 point of view a slave node (that will perform HPE inferences) is launched on each Raspberry Pi. Every slave node automatically generates new threads with other nodes such as worker, loop and webcam nodes as needed. On the other hand, a master node (that will obtain the 3D HPE result by merging results obtained by all slave nodes) is launched on the master PC and it will automatically start scanning the local ROS2 network for slave nodes and make subscriptions and calibration requests. Note that the RealSense nodes are not

spawned by the slaves, but need to be manually launched. The project provides an alternative webcam node by passing "NULL" as "raw_image_topic" parameter if the RealSense cameras are not used.

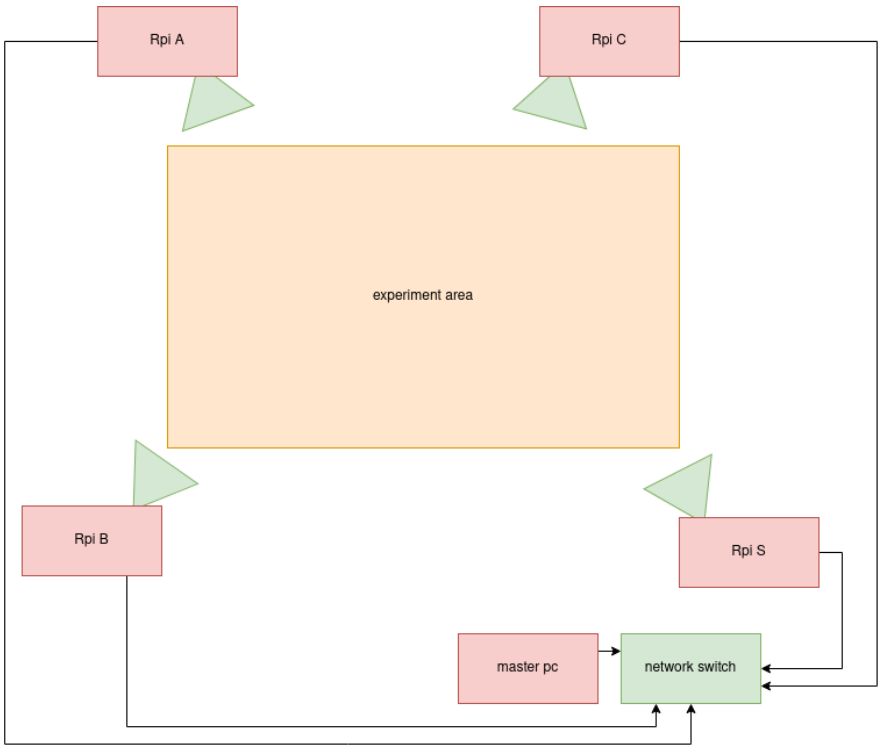


Figure 4.1: Hardware diagram

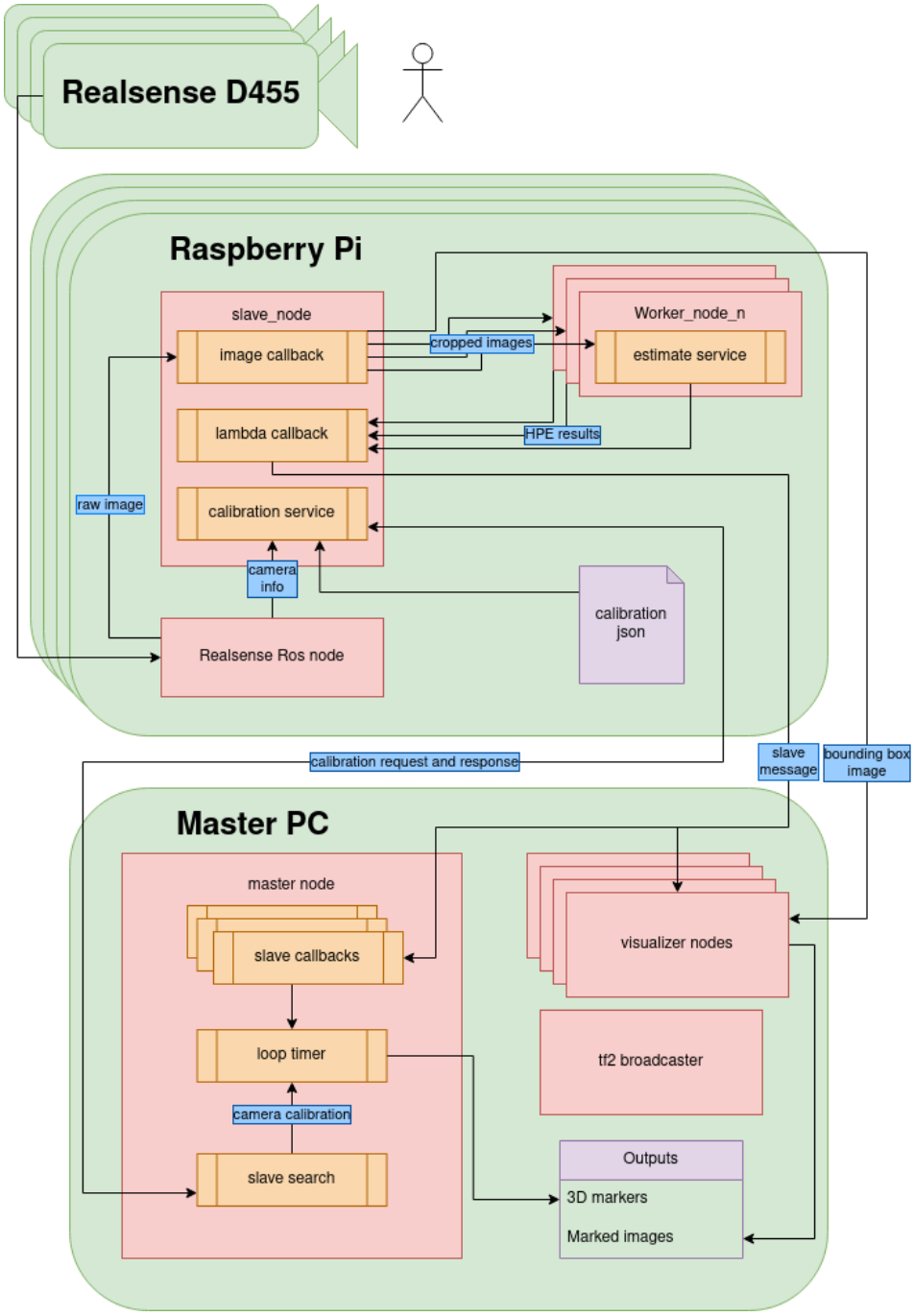


Figure 4.2: Software diagram

4.2 The slave node

This is the main node running on each Raspberry board, it is the one responsible for handling image data and executing the HPE on it. This is the only node that needs to be ran in order to setup all the logic related to the raspberry

computations.

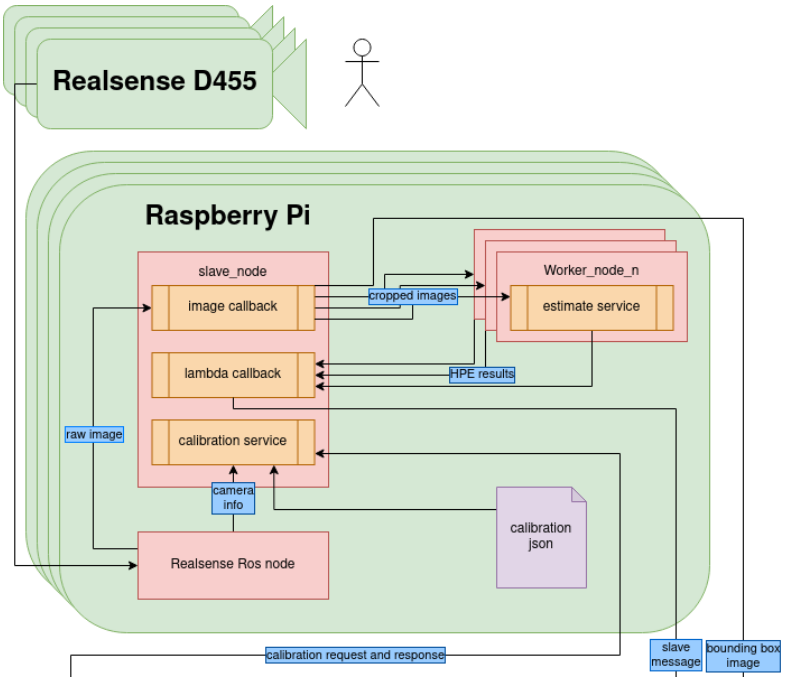


Figure 4.3: Slave node diagram

4.2.1 Startup

When a slave node is launched, it automatically generates a variable number of worker nodes (nodes responsible for the HPE inference) based on a command line parameter, then it creates a subscriber to the RealSense camera topic passed (or generates a webcam node that tries to get a video input from any webcam connected to the board if the user deems it necessary), if there is one, it subscribes to the *sensor_msgs::msg::CameraInfo* topic in order to get calibration information, it instantiate the TFlite tensors for the people detector and starts spinning.

4.2.2 Callback cycle

When a new image is received from the image topic and the callback function is called, the image message is used as input for the off-the-shelf people detector DCNN that returns the possible bounding boxes of all people in the image frame. After non maxima suppression is applied to the resulting bounding boxes array the best ones are used to resize the images and request a service to the worker nodes. In order to exploit all the resources available and to parallelize the

computation, each worker node only works on a single image at each cycle; if not enough workers are available at a certain cycle, new ones will be generated and the excess bounding boxes will be discarded. All the service requested to the workers are then tied to a lambda function, that will handle the delivery of the Slave message to the master node.

This node is capable of handling multiple people in the same image for 2D HPE, sadly the master node is not yet ready for this, as we will see in the next sections, but it wasn't always like this. In the first iterations of the project, the loop and worker nodes did not exist and the HPE computation that is now executed by the worker nodes was executed directly by the slave node, improving real time performance at the cost of accuracy, since the humans in the whole images are much smaller than the ones in the bounding boxes that the worker nodes now receive and the models were not able to correctly guess joint coordinates for such small targets. Currently this old version of the node still exists and is usable in the project and it is named "slave_single node".

Slave callback

Inputs: The image acquired by the corresponding camera.

Outputs: The callbacks on futures generated by the worker service calls.

Description: This callback runs the DCNN responsible for finding all bounding boxes that find people in the given image and request a HPE service to the worker nodes.

4.2.3 The worker node

Each worker node is directly created by a slave node and it holds the logic for executing a single person HPE on a bounding box each time a service is requested by the corresponding slave. A slave node can have as many active worker nodes as the maximum number of people found up to the relevant point in time. Since having more worker nodes in memory doesn't cause significant performance drops as long as they are not working, there is not any kind of "garbage collector" that kills inactive workers, but this would be a welcome addition if the project is picked up again later on.

Worker service

Inputs: The bounding box data resized to the HPE DCNN input size.

Outputs: The 2D skeleton detection corresponding to the person in the bounding box with respect to the camera frame coordinates.

Description: This service performs the single-person 2D HPE on the bounding box and sends back as response the skeleton obtained with an additional confidence vector.

4.2.4 The calibration service

Each slave node receives some calibration information when launched, either by a json file prepared in advance by the user (with the node name in the file's name) or a "sensor_msgs::msg::CameraInfo" subscriber if a publisher is passed as command line parameter by the user. In this second case, no extrinsic information is given in the info message and it is expected that the "frame_id" parameter in the info message header corresponds to a tf2 frame that is being actively published. The rotation and translation information is always gathered with respect to the "world" frame.

Calibration service

Inputs: Empty request.

Outputs: The Calibration message with the extrinsic parameters expressed as a tf2 frame and the intrinsics expressed as an IntrinsicParams message.

Description: This service finds and returns the information necessary for the master node triangulation process.

4.2.5 The loop node

This node was used in the old iterations, before the futures were handled by lambda callbacks. It ran on a separate thread with respect to the slave node and contained a loop that checked the pointer to the futures queue belonging to it's parent slave node for pending operations (while locking a mutex for thread safety) and if a futures vector was present the queue was popped and the loop

started waiting for the future results from the worker nodes in order to compile them into a "Slave" message that contained all information about detections at the point in time described in the header for the slave node.

Initially this loop was inside the main slave node, but there was an issue with the future handling (futures were never marked as SUCCESS) and it had to be moved to a separate node. Sadly the cause of the problem is not yet clear but it is suspected that it has to do with how the node multi threaded executor that was handling the slave node never stopped spinning, making impossible for other callbacks that have to do with future handling to be processed. This suspicion was raised when switching to a callback system instead of a loop, since the code:

```
executor.spin();
```

Code 4.1: Old slave spin

had to be changed with the code:

```
rclcpp::Rate rate(30);
while (keep_running.load() && rclcpp::ok()) {
    executor.spin_some();
    rate.sleep();
}
```

Code 4.2: New slave spin

in order to make callbacks work correctly.

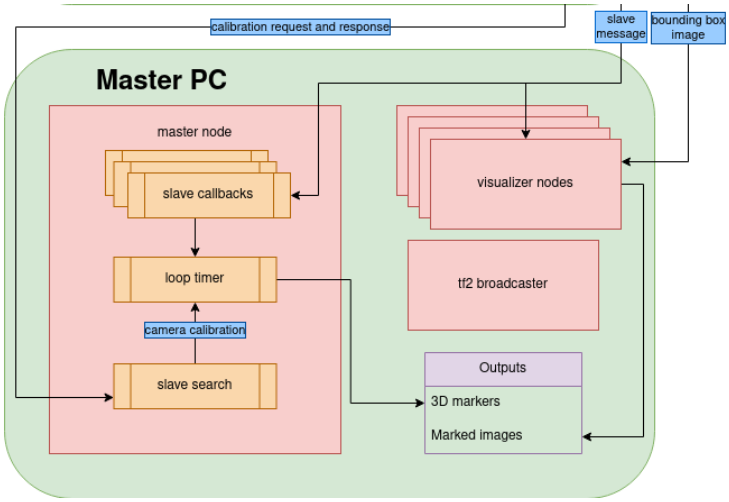


Figure 4.4: Master node diagram

4.3 The master node single

This multi-person master node is not usable at the moment, instead the "master_node_single" is compatible with the rest of the network and usable for single person 3D triangulation. This node is the only one that does not run on an edge device (even if it probably could since it is the node that uses the least computational power at the moment) and it's responsible for handling all Slave messages with 2D HPE generated by the slave nodes.

This node, when started, automatically starts scanning the ROS2 local network for nodes of type slave that are publishing every couple of seconds with a dedicated wall_timer. When one of these nodes is found a service request is made to the slave's calibration service and the data acquired is stored in an array. At the same time a subscription to the node's slave message publisher is generated and inserted in a second array at the same index occupied by the calibration information. When the node is created it also spawns a second thread called "loop thread".

4.3.1 Slave scanning and callbacks

There is not a direct way to find a node of a certain type in ROS2, this is why, in order to find active slave nodes, the master node takes the whole ROS2 topic list and matches the names against the "(/slave_)(.*)" regular ex-

pression. This is possible because all slave nodes publish their Slave messages on the "slave_nodeName" topic. From this knowledge, the master node keeps track of the new slave node name and, after adding a blank message in the slaves_feedback_ vector, it generates a lambda callback that will store the last message received from this particular new node at the correct index in the slaves_feedback_ vector. After this is done, the master node requests a calibration to the newfound node following the service naming convention "calibration_nodeName" and generates a new visualizer node, passing the same slave name as a constructor parameter and adding it to the multi threaded executor.

4.3.2 The visualizer node

This is a simple node that subscribes to a particular image topic that is published by a slave while executing the raw image callback (i.e. when performing the inference on the people detector DCNN). The image published by the topic is the same one that is published by the camera node, but with all of the considered bounding boxes drawn on it. This node simply reads the last Slave message and the last image from this topic and draws markers where the joints should be 4.5. Unfortunately, it does not store old images and, since the HPE inference and the people detector inference are running in parallel, the markers usually lag a bit behind the person in the frame.

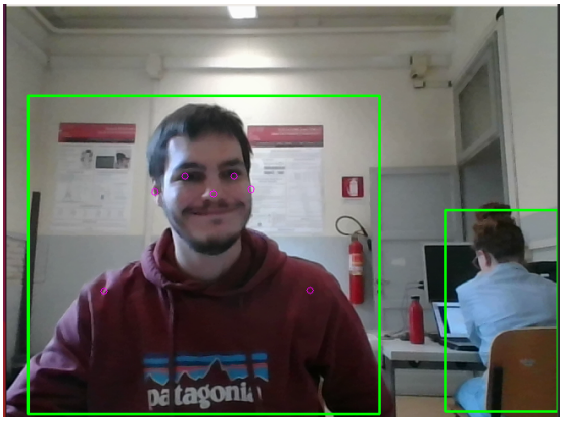


Figure 4.5: Visualizer output example

4.3.3 The master loop thread

This thread is the main thread of the node and it is used to compute the 3D HPE by triangulating the 2D joint information given by the slave nodes. At first the "filterFeedbacks()" function selects only the Slave messages from the slaves_feedback_ vector that were generated by an image acquired within a small time constant. This is possible because the time stamp in the Slave message header corresponds to the original image's. If fewer than two messages that satisfy this constraint are found in the vector, the thread prints a warning message and skips the triangulation; otherwise it performs a first unweighted triangulation as explained in chapter 3 and then a series of weighted ones for each joint in the given skeleton. After this process, the node clears old joint and edge markers in rviz2 and generates new ones that are color-coded based on the average confidence of the slaves responses for the given joint.

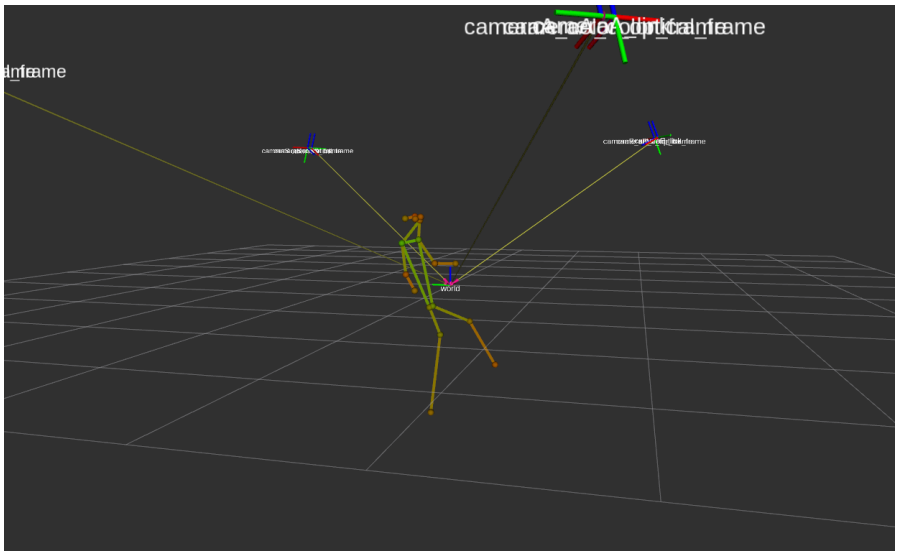


Figure 4.6: Example 3D skeleton obtained by the master node

4.4 Iterations

This project's architecture history can be divided in three major iterations for the time being. The first iteration is the simplest one: in this iteration the slave nodes simply received an image from the camera subscription and started a callback that performed the single person HPE, assuming that just one person was in the camera frame. This approach was the best for real-time performance,

but it lacks in accuracy when the person is far from the camera, this is why the second iteration of the project built.

In this second design phase, the slave node did not perform the HPE inference directly, but it used an off-the-shelf people detector in order to find the bounding boxes of all the people in the frame. When some people are found, the slave node sends the camera image cropped following the the bounding boxes to a set of worker nodes as a service request. The HPE computation is handled by the worker node on the zoomed in image and, at this point in time, the responses were handled by a separate node that was spawned by the slave: the loop node. This iteration leveraged the concept of futures, an object returned by an asynchronous request that is used to access the request's result when it terminates. The loop node was responsible for waiting for the asynchronous requests' futures and compiling a message with the multi-person 2D HPE result. Currently the triangulation process does not support multiple people and only the first skeleton returned by each slave node is actually used. This might create problems if the project is used inadvertently for multi-person HPE purposes.

The third phase is mainly about optimization. The loop node is no longer used and the futures are handled by callbacks in the slave node directly by the ROS2 scheduler. This phase also added some improvements in visualization and was used to test the best parameters for the network and the GPU delegate performance.

5 Experiments

5.1 Experiments

A total of 10 experiments were conducted in a laboratory with two Raspberry Pi 5 (RpiA, RpiS) and two Raspberry Pi 4 (RpiB, RpiC) connected via a network switch (1G) to the master computer that was running the master node. Each Raspberry was connected to a RealSense D455 camera and a RealSense-Ros node was launched on every board in order to acquire images and publish calibration information. The two Pi 5 boards were equipped with active cooling modules, while the others were running with no case and no cooling. This is mentioned because of overheating problems during the experiments that severely degraded the performance of two of the boards (RpiA, RpiC) but, even when using domestic fans for additional cooling, nothing could be done to improve them during these tests. In each experiment a different set of variables were used.

The table 5.1 reports the different parameter used for the experiment. The text (Sgl/Slv) indicates the type of node used for the inference. The "Sgl" node is the slave_single node, while the "Slv" is the final version of the slave node with people detection. On the other hand the numbers indicate the HPE model used for inference where 0 corresponds to the faster "lightning" model and 1 to the more accurate "thunder" model.

Test	Parameters pi 5	Parameters pi 4
0	Sgl 0	Sgl 0
1	Sgl 1	Sgl 0
2	Sgl 1	Sgl 1
3	Slv 0	Sgl 0
4	Slv 0	Sgl 1
5	Slv 1	Sgl 0
6	Slv 1	Sgl 1
7	Slv 0	Slv 0
8	Slv 1	Slv 1
9	Slv 1	Slv 1

Table 5.1: Experiment parameters

The metrics used for evaluating the networks are PCK, MPJME and HC PCK, where the HC PCK metric only keeps track of nodes with accuracy above the 0.6 threshold but otherwise works as a normal PCK. For both PCK metrics the chosen threshold was 250mm and the ground truth for every metric was obtained by using the MeTRAbs DCNN on the input image received from RpiS' camera.

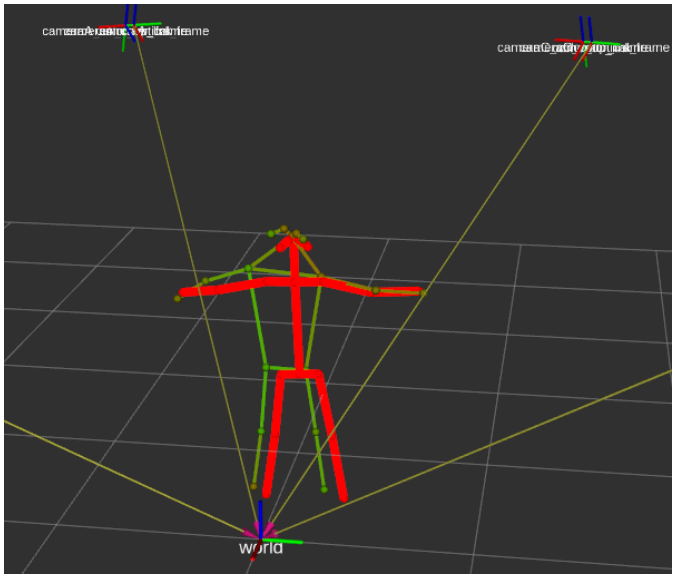


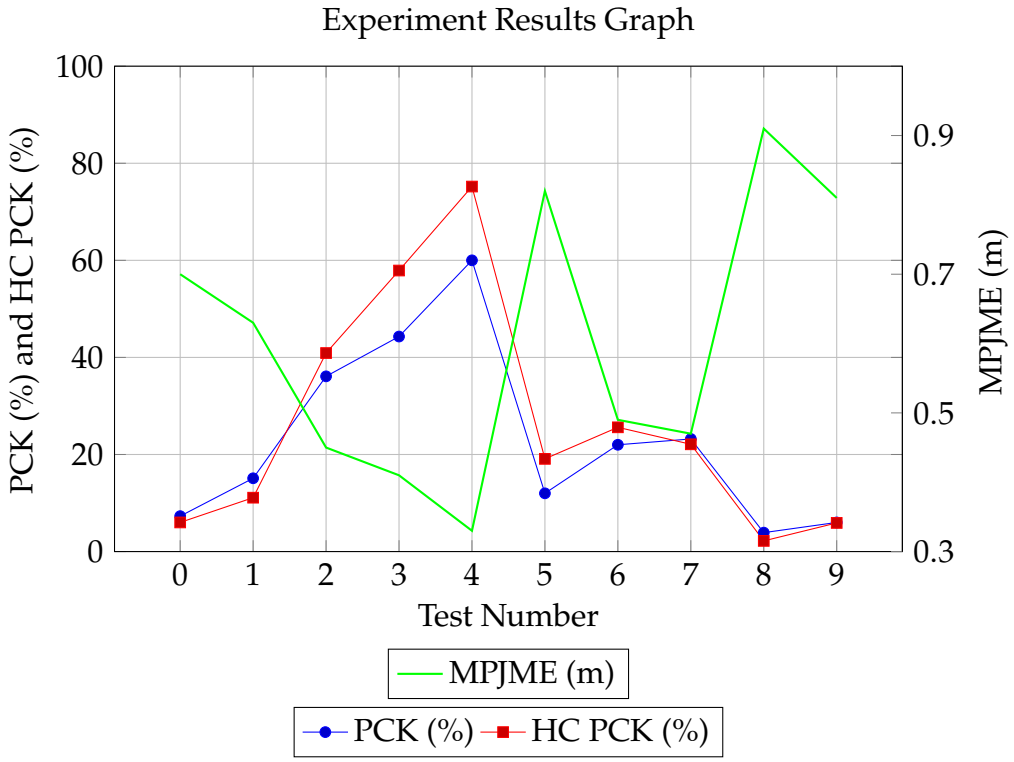
Figure 5.1: Skeleton from MeTRAbs(red) and our skeleton (green) during test 4

5.2 Results

Table 5.2 reports the accuracy results for the experiment that were conducted, while table 5.3 reports the performance of each Raspberry during the experiments. Note that the frame rates are calculated as $\frac{1}{delay}$, where $delay$ is the time that an image needs to be processed by the pipeline.

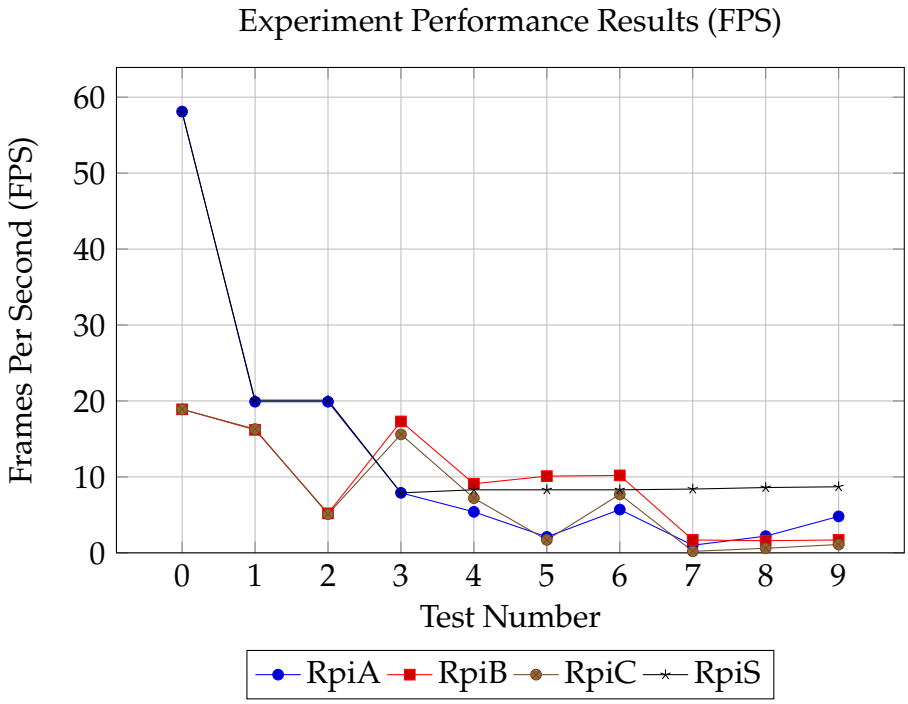
Test	**PCK** (%)	**HC PCK** (%)	**MPJME** (m)
0	07.3	06.0	0.70
1	15.1	11.1	0.63
2	36.1	40.9	0.45
3	44.3	57.9	0.41
4	60.0	75.2	0.33
5	12.0	19.1	0.82
6	22.0	25.6	0.49
7	23.2	22.1	0.47
8	03.9	02.2	0.91
9	06.0	05.9	0.81

Table 5.2: Experiment accuracy results



Test	RpiA	RpiB	RpiC	RpiS
0	58.1	18.9	18.9	58.0
1	19.9	16.2	16.3	20.1
2	19.9	05.2	05.1	20.1
3	07.9	17.3	15.6	07.9
4	05.4	09.1	07.2	08.3
5	02.1	10.1	01.7	08.3
6	05.7	10.2	07.7	08.3
7	01.0	01.7	00.2	08.4
8	02.2	01.6	00.6	08.6
9	04.8	01.7	01.1	08.7

Table 5.3: Experiment performance results (FPS)



When logging the results for "Slv" type tests the HPE inference delays were also logged, this fact generated a curious result: even if the Raspberry Pi 4 boards couldn't go above 19 FPS in the "Sgl" tests, the HPE inference frame rate was much higher in "Slv" tests, reaching up to 35.8 FPS when the "lightning" network was used and 14.4 when the "thunder" network was. This is probably due to how the ROS2 executors were used since that is the only relevant difference between the two node types. The same is true for the Raspberry Pi 5 boards that also saw increased HPE performance when ran with the different executor.

During the experiment it became evident that Raspberry boards with degraded performance (RpiA when overheating) and older Pi 4 boards when under heavy workloads couldn't keep up with sending messages across the local network. This fact severely degraded performance during more intensive tests even after reboots and efforts to cool them down. In the end the only boards that didn't suffer from overheating problems are the RpiB (that was without a case and positioned close to an open door that provided some air currents) and the RpiS (that had a custom perforated case on top of the stock Raspberry active cooling solution and was positioned next to an open window).

6

Conclusions and Future Works

6.1 Conclusions

From the results of the experiment it is clear to see that older Raspberry Pi 4 boards are inadequate for real-time HPE processing, a network made using this project should only be comprised of Raspberry Pi 5 boards. Furthermore the data acquired suggest that a great bottleneck is generated in the system when using the people detector, making the FPS count plummet. Since the best test results were obtained with the added precision granted by this last network though a solution to the problem might lie in utilizing a lighter network for that purpose as it produces a great boost in accuracy with respect to the faster methods that do not use it. Overall the system runs in real-time with poor accuracy, but could be made better (as seen from the experiments) by sacrificing some speed or by experimenting with new ways to reduce the overall CPU workload such as possibly using another lighter model when performing people detection.

6.2 Future works

Other than a new people detector network, some more topics require further investigation and possible work and adjustments. First of all, during the more intensive tests, some difficulties were encountered with the local network message delivery. The high CPU usage made it difficult for the older boards and the ones that were suffering from overheat to reliably send messages to the master PC on time, raising the overall latency. In order to relieve the CPU from some of it's workload alternative hardware could be used to boost the overall system performance. At first a test with the on-board GPU should be tried, this was not possible during this project because of driver issues, hopefully this problem will eventually be solved as some hobbyists claim to have done in Raspberry's official forums. Other tests could be conducted by using other hardware such as the Raspberry AI camera and the Coral USB accelerator that were presented in chapter 3.

Another thing that would need to be worked on is the master node. As of now it lacks an appropriate algorithm to be used for multi-person HPE. Such an algorithm would need to able to reliably discard false positive detections and would need to include skeleton tracking as the system is not synchronized as of now.

The last path that could be taken in future research would be to eliminate the people detector DCNN altogether and substitute it and the current single-person HPE DCNN with another network that directly performs multi-person inferences, if possible even with 3D outputs. 3D detections could for sure make the final detections overall more precise with the additional information, with that said it is probably too computationally intensive for this type of devices and some other steps would need to be taken in order to ensure the applicability of a 3D DCNN. In order to archive this result, a good starting point could be found in the knowledge distillation research [6] discussed in chapter 2, but a whole project should be dedicated to this research in order to get acceptable results.

7 Appendix

7.1 Messages and services

In this section the messages and services used in the project will be exposed to the viewer in order to try and give a better understanding of the architecture.

7.1.1 Estimate service

This service is the one used by the slave node to request a DCNN inference on the image data contained in a bounding box.

```
1 hpe_msgs/Detection detection
2 ---
3 hpe_msgs/Hpe2d hpe2d
```

Code 7.1: Estimate.srv

7.1.2 Detection

This message contains the cropped image defined by the corresponding bounding box and it's used in the Estimate service as a request parameter.

```
1 sensor_msgs/Image image
2 hpe_msgs/Box box
```

Code 7.2: Detection.msg

7.1.3 Box

This message is used to represent a bounding box in an image and hold some more useful image related information in order to convert the coordinates of joints found by the worker node from the bounding box frame to the original image frame's pixel coordinate system.

```
int32 x
int32 y
int32 width
int32 height
int32 img_width
int32 img_height
```

Code 7.3: Box.msg

7.1.4 Hpe2d

This message is used as response in the Estimate service and it contains an header for keeping track of the time at which the image was taken and the Joints2d message representing the coordinates of the joints found by the worker node performing HPE.

```
std_msgs/Header header
hpe_msgs/Joints2d joints
```

Code 7.4: Hpe2d.msg

7.1.5 Joints2d

This message holds all the information about the position and confidence for each joint in a detection, as well as the number of joints found by the worker node.

```
float32[] x
float32[] y
float32[] confidence
int32 dim
```

Code 7.5: Joints2d.msg

7.1.6 Slave

This message contains the aggregated result of the skeletons found by a slave node. This is the message that is sent to the master node after each loop iteration in the slave node.

```
std_msgs/Header header
hpe_msgs/Joints2d[] all_joints
int32 skeletons_n
```

Code 7.6: Slave.msg

7.1.7 Calibration service

This is a simple service that has no request parameters, the response is a custom message that holds intrinsics and extrinsic information about the camera related to the slave node acting as service server.

```
---
hpe_msgs/Calibration calibration
```

Code 7.7: Calibration.srv

```
geometry_msgs/TransformStamped frame
hpe_msgs/IntrinsicParams intrinsic_params
```

Code 7.8: Calibration.msg

7.1.8 IntrinsicParams

This message is used as part of the calibration message and holds all information about the intrinsic camera matrix and the distortion coefficients.

```
float64[] camera_matrix             # 3x3 intrinsic matrix
float64[] distortion_coefficients   #[k1, k2, p1, p2, k3]
```

Code 7.9: IntrinsicParams.msg

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV]. URL: https://arxiv.org/abs/2004.10934.

[2] Yucheng Chen, Yingli Tian, and Mingyi He. "Monocular human pose estimation: A survey of deep learning-based methods". In: *Computer Vision and Image Understanding* 192 (2020), p. 102897. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2019.102897. URL: https://www.sciencedirect.com/science/article/pii/S1077314219301778.

[3] M.A. Fischler and R.A. Elschlager. "The Representation and Matching of Pictorial Structures". In: *IEEE Transactions on Computers* C-22.1 (1973), pp. 67–92. DOI: 10.1109/T-C.1973.223602.

[4] Luca Fortini et al. *Markerless 3D human pose tracking through multiple cameras and AI: Enabling high accuracy, robustness, and real-time performance*. 2023. arXiv: 2303.18119 [cs.CV]. URL: https://arxiv.org/abs/2303.18119.

[5] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV]. URL: https://arxiv.org/abs/1704.04861.

[6] Dong-Hyun Hwang et al. "Lightweight 3D Human Pose Estimation Network Training Using Teacher-Student Learning". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Mar. 2020.

[7] Catalin Ionescu et al. "Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (July 2014), pp. 1325–1339.

[8] Tao Jiang et al. *RTMPose: Real-Time Multi-Person Pose Estimation based on MMPose*. 2023. arXiv: 2303.07399 [cs.CV]. URL: https://arxiv.org/abs/2303.07399.

[9] Rohit Josyula and Sarah Ostadabbas. *A Review on Human Pose Estimation*. 2021. arXiv: 2110.06877 [cs.CV]. URL: https://arxiv.org/abs/2110.06877.

[10] Naimat Ullah Khan and Wanggen Wan. "A Review of Human Pose Estimation from Single Image". In: *2018 International Conference on Audio, Language and Image Processing (ICALIP)*. 2018, pp. 230–236. DOI: 10.1109/ICALIP.2018.8455796.

[11] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312.

[12] Qihao Liu et al. "Explicit Occlusion Reasoning for Multi-person 3D Human Pose Estimation". In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, 2022, pp. 497–517. ISBN: 978-3-031-20065-6.

[13] Xinnan Ma et al. "SCALE-Pose: Skeletal Correction and Language Knowledge-assisted for 3D Human Pose Estimation". In: *Pattern Recognition and Computer Vision*. Ed. by Zhouchen Lin et al. Singapore: Springer Nature Singapore, 2025, pp. 578–592. ISBN: 978-981-97-8795-1.

[14] Rama Bastola Neupane, Kan Li, and Tesfaye Fenta Boka. "A survey on deep 3D human pose estimation". In: *Artificial Intelligence Review* (2024). ISSN: 1573-7462. DOI: 10.1007/s10462-024-11019-3. URL: https://doi.org/10.1007/s10462-024-11019-3.

[15] Leonid Pishchulin et al. *DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation*. 2016. arXiv: 1511.06645 [cs.CV]. URL: https://arxiv.org/abs/1511.06645.

[16] István Sárándi, Alexander Hermans, and Bastian Leibe. *Learning 3D Human Pose Estimation from Dozens of Datasets using a Geometry-Aware Autoencoder to Bridge Between Skeleton Formats*. 2022. arXiv: 2212.14474 [cs.CV]. URL: https://arxiv.org/abs/2212.14474.

[17] István Sárándi et al. "MeTRAbs: Metric-Scale Truncation-Robust Heatmaps for Absolute 3D Human Pose Estimation". In: *CoRR* abs/2007.07227 (2020). arXiv: 2007.07227. URL: https://arxiv.org/abs/2007.07227.

[18] Hai-Thien To, Trung-Kien Le, and Chi-Luan Le. "Real-Time End-to-End 3D Human Pose Prediction on AI Edge Devices". In: *Intelligent Systems and Networks*. Ed. by Duc-Tan Tran et al. Singapore: Springer Singapore, 2021, pp. 248–255. ISBN: 978-981-16-2094-2.

[19] Hanyue Tu, Chunyu Wang, and Wenjun Zeng. *VoxelPose: Towards Multi-Camera 3D Human Pose Estimation in Wild Environment*. 2020. arXiv: 2004.06239 [cs.CV]. URL: https://arxiv.org/abs/2004.06239.

[20] Dongkai Wang, Shiyu Xuan, and Shiliang Zhang. "LocLLM: Exploiting Generalizable Human Keypoint Localization via Large Language Model". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 614–623.

[21] Yi Yang and Deva Ramanan. "Articulated pose estimation with flexible mixtures-of-parts". In: *CVPR 2011*. 2011, pp. 1385–1392. DOI: 10.1109/CVPR.2011.5995741.

[22] Dongyang Yu et al. "MovePose: A High-Performance Human Pose Estimation Algorithm on Mobile and Edge Devices". In: *Artificial Neural Networks and Machine Learning – ICANN 2024*. Ed. by Michael Wand et al. Cham: Springer Nature Switzerland, 2024, pp. 144–158. ISBN: 978-3-031-72338-4.

[23] Feng Zhang et al. *Distribution-Aware Coordinate Representation for Human Pose Estimation*. 2019. arXiv: 1910.06278 [cs.CV]. URL: https://arxiv.org/abs/1910.06278.

[24] Jinrui Zhang et al. "MobiPose: real-time multi-person pose estimation on mobile devices". In: *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. SenSys '20. Virtual Event, Japan: Association for Computing Machinery, 2020, pp. 136–149. ISBN: 9781450375900. DOI: 10.1145/3384419.3430726. URL: https://doi.org/10.1145/3384419.3430726.

[25] Siqi Zhang et al. "A Survey on Depth Ambiguity of 3D Human Pose Estimation". In: *Applied Sciences* 12.20 (2022). ISSN: 2076-3417. DOI: 10.3390/app122010591. URL: https://www.mdpi.com/2076-3417/12/20/10591.

[26] Qitao Zhao et al. *PoseFormerV2: Exploring Frequency Domain for Efficient and Robust 3D Human Pose Estimation*. 2023. arXiv: 2303.17472 [cs.CV]. URL: https://arxiv.org/abs/2303.17472.

[27] Ce Zheng et al. *3D Human Pose Estimation with Spatial and Temporal Transformers*. 2021. arXiv: 2103.10455 [cs.CV]. URL: https://arxiv.org/abs/2103.10455.

[28] Ce Zheng et al. "Deep Learning-Based Human Pose Estimation: A Survey". In: *CoRR* abs/2012.13392 (2020). arXiv: 2012.13392. URL: https://arxiv.org/abs/2012.13392.

[29] Zhengxia Zou et al. "Object Detection in 20 Years: A Survey". In: *Proceedings of the IEEE* 111.3 (2023), pp. 257–276. DOI: 10.1109/JPROC.2023.3238524.