Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Dipartimento di Ingegneria dell'Informazione

Master's Degree in

Computer Engineering

# Uncertainty-aware human pose estimation for collaborative aerial robots

Supervisor: **Prof. Stefano Ghidoni**
Co-supervisor: **Dr. Matteo Terreran**
External supervisor: **Prof. Antonio Franchi** (University of Twente)
Candidate: **Filippo Boldrin**

Academic Year 2022-2023

Graduation date: December 5th, 2023

# Abstract

Human pose estimation (HPE) algorithms based on deep learning allow for extraction of pose information (skeletal keypoints) from images or videos. While the pixels-to-keypoint correspondence in images may be determined with reasonable confidence and performance, applications of HPE algorithms to robotics are much more challenging, because they require estimation of human poses in three-dimensional coordinates (3D HPE). Moreover, collaborative robots using HPE for localizing humans in the workspace may need a measure of the quality of the estimation, to avoid taking decisions based on wrong detections of humans in the camera data.

This experimental thesis focuses on the design and implementation of a "Quality of Estimation" (QoE) heuristic, giving a measure of the uncertainty in the estimation of the 3D human pose. The QoE heuristic is calculated using heatmaps from the pose estimator and simple geometric priors on the shape of the human body, and it can be computed in real time from a live camera feed.

Considering a human-collaborative UAV handover setting, the proposed heuristic has been validated on a real scenario including a stereo camera and a hexacopter, showing a favorable performance to power ratio. Moreover, the heuristic was used in a simulated and simplified human-robot collaboration setting to develop exploration policies based on reinforcement learning. Experimental results show that the heuristic allows to learn an exploration policy minimizing the uncertainty on the estimated 3D human pose.

All the software was developed and tested on an actual embedded platform (Nvidia Jetson TX2) and a hexacopter equipped with a RGB-D camera from the University of Twente.

# Sommario

Gli algoritmi di Human Pose Estimation (HPE) sono basati su tecniche di Deep Learning e permettono di estrarre informazioni sulla posizione di alcuni giunti scheletrici umani all'interno di immagini e video. E' possibile ottenere la corripondenza tra giunti scheletrici e pixel con buone prestazioni, sia in termini di efficienza computazionale che di accuratezza; d'altra parte però le applicazioni della HPE al dominio della robotica sono meno immediate, perché richiedono di stimare i giunti scheletrici nello spazio tridimensionale. Inoltre, un robot collaborativo che utilizzi algoritmi di HPE per localizzare gli umani all'interno del suo workspace potrebbe necessitare di euristiche in grado di quantificare la precisione dell'output: lo scopo è quello di evitare decisioni basate su un output errato dell'algoritmo di HPE, che potrebbero mettere a rischio la sicurezza dell'operatore.

Questo lavoro sperimentale si concentra sulla progettazione e implementazione di una euristica chiamata "Quality of Estimation" (QoE), che fornisce in tempo reale una misura dell'incertezza nella stima della posa 3D. La QoE viene calcolata usando le heatmap del pose estimator e semplici assunzioni sulla geometria del corpo umano.

In uno scenario di handover tra una persona e un UAV collaborativo, la suddetta euristica è stata validata in uno scenario reale, che include una stereocamera e un esacottero, dimostrando un rapporto favorevole tra accuratezza e tempo di elaborazione. Oltre a ciò, l'euristica è stat usata in un ambiente di simulazione che riproduce in maniera semplificata uno scenario di collaborazione tra operatore umano e robot, allo scopo di sviluppare una policy esplorativa basata per mezzo del reinforcement learning.

I risultati sperimentali mostrano che l'euristica permette di apprendere una policy di esplorazione che minimizza l'incertezza dei keypoint sulla posa 3D stimata.

Il software è stato sviluppato e testato su un sistema embedded reale (Nvidia Jetson) e un esacottero dell'Università del Twente.

# Contents

**1 Introduction** **1**
- 1.1 Scope of the work . . . 2
- 1.2 Addressed challenges . . . 3
- 1.3 Structure of the thesis . . . 4

**2 Literature review** **5**
- 2.1 2D human pose estimation . . . 5
  - 2.1.1 Top-down (two-stage) approach . . . 5
  - 2.1.2 Bottom-up approach: OpenPose . . . 6
  - 2.1.3 Visual Transformers . . . 9
- 2.2 3D human pose estimation . . . 10
  - 2.2.1 The challenges of 3D HPE . . . 10
  - 2.2.2 Pose lifting vs end-to-end training . . . 10
  - 2.2.3 Capsule Neural Networks for 3D HPE . . . 11
- 2.3 Uncertainty in human pose estimation . . . 13
- 2.4 Practical literature review . . . 14
  - 2.4.1 3D HPE . . . 15
  - 2.4.2 2D pose estimation . . . 16

**3 Pose estimator and QoE heuristic** **19**
- 3.1 2D Human pose estimator . . . 19
- 3.2 2.5D to 3D augmentation . . . 19
- 3.3 Quality of Estimation heuristics . . . 21
  - 3.3.1 Heatmap-based QoE heuristics . . . 22
  - 3.3.2 Distance QoE heuristics . . . 25
  - 3.3.3 Combining the QoE heuristics . . . 29
  - 3.3.4 Temporal correlation and intermittent measures . . . 29
- 3.4 Hardware configuration . . . 30
  - 3.4.1 Hexarotor . . . 30
  - 3.4.2 Onboard embedded computer . . . 30
  - 3.4.3 RGB-D camera . . . 30
- 3.5 Experimental evaluation . . . 31

3.5.1 Limitations of the 3D pose estimator . . . . . . . . . . . . . . . . 31
3.5.2 Evaluation of the QoE heuristics . . . . . . . . . . . . . . . . . . 32
3.5.3 Run-time analysis . . . . . . . . . . . . . . . . . . . . . . . . . . 32

**4 Exploration policy learning** **34**
4.1 Limitations of the uncertainty estimation heuristic . . . . . . . . . . . 34
4.1.1 Using the uncertainty heuristic for exploration . . . . . . . . . . 34
4.2 State-action predictor . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.3 Reinforcement Learning . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.3.1 Continuous state and action spaces . . . . . . . . . . . . . . . . 37
4.3.2 System setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
4.3.3 Discretized states and actions . . . . . . . . . . . . . . . . . . . 41
4.3.4 Environments . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
4.3.5 Environment design . . . . . . . . . . . . . . . . . . . . . . . . . 43
4.3.6 Learning algorithm and network architecture . . . . . . . . . . . 44
4.4 Training details . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
4.5 Reward design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
4.5.1 Binary reward . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
4.5.2 Binary reward with path length penalty . . . . . . . . . . . . . 47
4.5.3 Variable reward based on QoE value . . . . . . . . . . . . . . . 47
4.6 Evaluation and results . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.6.1 Quantitative results . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.6.2 Qualitative analysis and failure modes . . . . . . . . . . . . . . 48

**5 Conclusions and future work** **50**

**A Reinforcement learning primer** **52**
A.1 Deep Reinforcement Learning . . . . . . . . . . . . . . . . . . . . . . 53
A.2 Proximal Policy Optimization . . . . . . . . . . . . . . . . . . . . . . 54

**Bibliography** **55**

**List of figures** **62**

# Acknowledgements

This thesis was developed at the University of Twente (Netherlands), between March and August 2023. I wish to thank first and foremost Prof. Ghidoni, who spent many hours in the effort to make this exchange happen. My mobility period was made possible also thanks to the Erasmus+ EU programme and the Italian Youth Fund (Fondo Giovani Nazionale), which provided funding, and to the RaM department at the University of Twente, which provided the facilities I used for research and experiments. I finally wish to thank Amr Afifi, Nicolò Botteghi and Antonio Franchi for hosting me and giving me supervision during my stay.

# 1 | Introduction

The research work presented in this thesis explores the topic of visual perception in autonomous aerial robots.

*Autonomous mobile robots* (robots that act adaptively according to the conditions of the environment they are in, which they can perceive and explore) have acquired industrial significance in the last two decades[1]. While non-autonomous robots have very limited perceptual needs, autonomous robots need many sensors and sophisticated *perception* algorithms, *i.e.* algorithms used to obtain structured information from the raw sensor readings; this information may then be used to guide the decision-making of the robot, integrating information from the environment the robot is working in. When the sensor being considered is a camera, as it is in the case study presented in the following, the algorithms belong to the varied and complex realm of *computer vision*.

Autonomous robots may be designed to interact with humans: for instance, in industrial settings, a robot may be used to lift heavy loads that a human will work on. Whenever an autonomous robot is designed to interact more or less naturally with a human, it is called a *collaborative robot*. Collaborative robots (or *cobots*) are considered essential tools to enable the so-called *fourth industrial revolution*: while non-autonomous robots (*e.g.* robot arms used for factory automation) have very limited flexibility with regard to the setting they are operated in, autonomous robots deliver a much higher degree of robustness and flexibility, enabling the manufacturing industry to develop innovative processes. Additionally, collaborative robotics acknowledges that humans and robots are not interchangeable, and that industrial processes would benefit from integrating the skills of the two in the same workspace, relying on the sensing capabilities of the cobots to ensure safety of the human operator.

Aerial robots (often referred to as UAVs -Unmanned Aerial Vehicles-, or *drones*) have also rapidly increased their applicative significance in the past years, especially in the form of VTOL (Vertical TakeOff and Landing vehicles). Because of their structural simplicity and maneuverability, they are used for inspection of hard-to-reach parts of

[1]The first example of autonomous mobile robot is arguably Shakey The Robot, created in Stanford in the late 1960s [1], but it is in the 1990s that autonomous robots start to gain significance thanks to technological and theoretical advancements, leading to the commercialization of autonomous lawn-mowers and vacuum cleaners in the early 2010s. See [2, chapter 2] for a brief history of autonomous robotics.

Figure 1.1: Typical powerline maintenance scenario

industrial complexes, or to capture a bird's eye view of the environment they are deployed in.

## 1.1 Scope of the work

*Collaborative aerial robots* lie at the intersection of research on intelligent robotics and unmanned aerial vehicles. A collaborative UAV is a tool that may prove useful whenever a human operator is working at heights, both in indoor industrial complexes or outdoors (*e.g.* for aerial powerline maintenance, assembly of large contrivances, construction works, tree pruning...)[3]. Research on collaborative aerial robots has been promoted in recent years by the EU-funded project AERIAL-CORE[2]. The project focuses on the applications of autonomous and collaborative UAVs for inspection and maintenance of high-rise infrastructures, with a special focus on powerlines, as high-voltage pylons are usually out of the reach of aerial work platforms: maintenance workers of powerlines usually need to climb on the pylons and work at height while perched on the pylons and secured with ropes.

This thesis work starts by envisioning the following scenario for human-robot collaboration: a human operator is working at height on a powerline. A collaborative UAV allows the operator not to carry a bag with heavy tools, as the robot will carry tools and other objects on command from the ground to the position of the worker, performing a *handover*. This task may be separated into several subtasks:

1. Localizing and picking the correct object from the ground
2. Gathering altitude until the robot is in the general vicinity of the worker
3. Localization of the human (to make sure they are in the field of view of the camera)

[2]`https://aerial-core.eu`

4. Hovering around the human, searching for a suitable position for performing the handover

5. Flying close to the human and releasing the tool once the human has grasped it

The focus of the thesis was on subtask 4 of the above list, and in particular on using perceptual information from an onboard camera to guide the environment exploration (this technique is known among roboticists with the name of *active perception*). Three building blocks revolve around the solution of the task outlined in step 4:

1. A human pose estimation algorithm, which takes an image as input and gives localization of some keypoints of the human skeleton (such as shoulders, elbows, wrists) in space as output.[3]

2. An uncertainty estimation algorithm, which takes the output of step 1 as input and returns a score of the prediction, with a high score assigned to predictions that are likely to be correct.

3. An exploration policy: a policy for deciding how to explore the environment around the human worker in order to reach a position where the score computed in step 2 is the lowest (or at least low enough to guarantee a successful handover).

In the following of this thesis, algorithms for solving all of the above steps will be proposed. The main contribution of this thesis is the proposed algorithm for solving step 2: in fact, there seems to be no algorithm in the scientific literature which tackles the problem of real-time uncertainty quantification for human pose estimation algorithms, and in the following, the QoE (Quality of Estimation) heuristic computation will be presented.

Furthermore, the QoE heuristic will be used also to provide a partial solution to problem number 3: it will be shown that the the heuristic conveys enough information to train an agent in a reinforcement learning setting, to learn an exploration policy which reaches a point in which the QoE uncertainty is minimized.

## 1.2 Addressed challenges

**Perception in unstructured environment** Previous works, like [4], have focused on the human-robot handover problem within a structured environment, this work strives to do the same in an environment that is as general as possible. In particular, while it is standard for robotics research to use *fiducial markers* such as AprilTag [5] to aid the localization of relevant locations in the environment, in the setting being considered,

[3]Identifying the location of the parts of the body in the image is not enough: in fact, planning the movement of the robot requires real-world (three-dimensional) coordinates, and the robot needs to navigate to *e.g.* 1 m to the shoulder of the human to solve the problem successfully.

the only information the robot is allowed to use for localizing the human is a raw RGB-D (RGB + Depth) stream.

**Safety and uncertainty quantification** Human pose estimation is not guaranteed to give correct results, and as with most computer visions tasks, it may fail unexpectedly. Computer vision algorithms (especially when based on deep neural networks, as is the case with Human Pose Estimation) are usually not endowed with self-reflection properties, and are not able to effectively estimate their own confidence. In order to do so, an algorithm to quantify the uncertainty of estimation (the Quality of Estimation heuristic) will be proposed in the following.

**Onboard computation** Considering that a collaborative UAV might be operated outdoors, minimizing the number of different components to be set-up and connected is of the utmost importance. The ideal collaborative robot would be a self-contained system, allowing a single person to transport it and make it operational without using any external compute nodes (such as a notebook communicating with the drone). Because of this, one of the objectives pursued was making sure all the required components (controller, perception and elaboration) could run on an energy efficient compute device, which could, at least in theory, be mounted on the drone itself. As detailed in the following, some design choices have been dictated by the desire to produce a realistic prototype within reasonable computation power and energy efficiency constraints. This is also the reason why most of the software has been written in plain C++, avoiding modular frameworks such as ROS and focusing on maximizing the computational efficiency.

## 1.3 Structure of the thesis

In Chapter 2, a theoretical and practical review of previous literature is presented, mostly on the topic of Human Pose Estimation. In Chapter 3, the Quality of Estimation heuristic is exposed, together with a naive 2D-to-3D pose estimator used in experiments, motivated by the hardware limitations of the setup being used. Finally, in Chapter 4, the learned exploration policy based on the QoE heuristic is presented. Appendix A contains basic Reinforcement Learning concepts that may prove useful to read chapter 4 more easily.

# 2 | Literature review

Human pose estimation is the task of extracting the position of *skeletal keypoints* (usually corresponding to skeletal joints such as elbows) from a picture of sequence of pictures depicting one or more humans. 2D pose estimation gives as output pixel locations, while 3D pose estimation gives as output positions in space.

## 2.1 2D human pose estimation

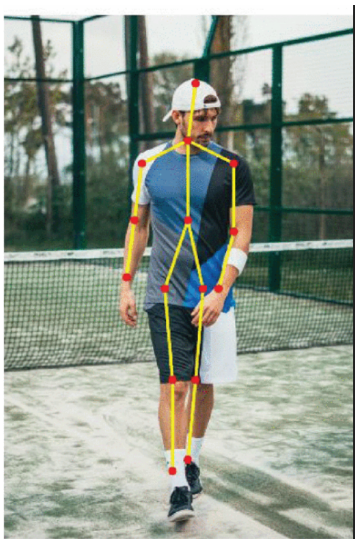
Human pose estimation algorithms, similarly to most high-level Computer Vision tasks, had been unable to obtain good flexibility and generalization power until the advent of Deep Learning in the early 2010s. Many classical algorithms had been proposed in the years, especially modeling body parts with graph-like data structures.

The first (and simplest) application of deep learning to the problem was DeepPose [6], proposed in 2014 by Toshev and Szegedy. DeepPose uses a cascade of CNN models to directly regress the coordinate of each joint in the image space. In particular, given an image $I$, DeepPose computes $y = \psi(I; \theta)$ where $\psi$ is the convolutional neural network using the parameters $\theta$ learned during training. $y \in \mathbb{R}^{2k}$ is the vector of $(x, y)$ coordinates of each of the $k$ keypoints of the skeletal model.

### 2.1.1 Top-down (two-stage) approach

Simple models such as DeepPose (and many subsequent refinements) cannot be applied to images where there is more than one person or in which people are not immediately recognizable (e.g. in a landscape picture featuring one single person in the background), so they may only be reasonably applied to images featuring one single human as the most prominent object inside the picture. One simple workaround may be the so-called *top-down approach*: initially, the whole image is processed using a generic object detector network such as Mask R-CNN[7] or YOLO[8], fine-tuned to recognize humans, thus obtaining bounding boxes for the humans in the image. Subsequently, a single-person pose estimator is run individually on image crops containing the bounding box areas. The two stages of the prediction (human detection and pose estimation) give the name to this technique.

Figure 2.1: Typical output of a 2D pose estimator model. The keypoints given as output are red points in the image, and the linking between them are decided a priori by the skeletal model being considered. Image by [9] Licensed under CC BY 4.0



Though simple, this technique introduces significant problems for the overall system performance: if the object detector fails, then failure of the pose estimator is inevitable; moreover, even a slight misalignment of the bounding boxes (which is explicitly considered acceptable by the loss functions used to train the object detector) could cause performance degradation in the pose estimator. Some workarounds to solve the issue of top-down human pose estimation were proposed, for instance in [10] [11] [12], but other solutions have become more popular, mostly because of their better scalability[1] and because the complexity of algorithms employing two-stage pose estimation is significant.

### 2.1.2 Bottom-up approach: OpenPose

A parallel line of research to the one exposed in the previous paragraph regarded instead the possibility of detecting *single body parts* and then perform a *parsing step* to link them together correctly. This technique is also called *bottom-up* human pose estimation.

**Heatmaps** Bottom-up HPE heavily relies on the concept of *heatmap*s, a per-pixel confidence map that evaluates the likelihood of a given skeletal joint being present in each pixel. The application of this concept to HPE, and, most importantly, the idea of obtaining such heatmaps as CNN outputs, can be ascribed to Tompson et al., who in 2014 proposed a hybrid neural-Markov Random Field model for human

[1]A nontrivial disadvantage of using top-down human pose estimator is that runtime of the algorithm is linear in the number of detected bounding boxes: this poses significant limitations to applications such as video streams, where runtime should not vary significantly.

pose estimation [13]. The CNN they propose is trained on images labeled with joint positions of the humans, which are however smoothed out with a Gaussian kernel in the preprocessing. This step allows the model to output heatmaps that do not assign a single pixel to every skeletal keypoint, producing a twofold benefit: on the one hand, it naturally produces pseudo-probability distributions centered on the ground truth position of the joint, on the other hand, since no constraint is posed on the minimum distance between detected keypoints, the smoothing of several detected joints close to one another allows for implicit modeling of the spatial extension of keypoints in images, which is often greater than one pixel [2].

Given that no specific normalization procedures are generally applied to the CNN output, heatmaps do not represent proper probability distributions, since every pixel generally contains values between zero and one. Moreover, negative values may also be outputted for certain pixels. See Figure 2.2 for a visualization of the typical heatmap output of a pose estimator.

**Part Affinity Fields** One of the most successful approaches to multi-person pose estimation was given by OpenPose [14][15], which integrates and modifies the architecture proposed by [16]. The innovation of OpenPose mostly consisted in the introduction of a new output of the pose estimator, the *Part Affinity Fields* (PAF). While heatmaps may be interpreted as scalar fields over the domain of image pixels, a PAF is instead a 2D vector field, in which nonzero vectors are unit vectors laying on the pixels depicting the limbs, pointing from the starting point to the endpoint of such limb, according to a predefined skeleton topology.

**Parsing** Human parsing (*i.e.* arranging the detected keypoints into skeletons) is made easier and faster by the use of PAFs. In general, for skeletons featuring $K$ keypoints, deciding which couples of keypoints represent different ends of the same limb, is an instance of the K-dimensional matching problem[3], which is NP-Hard. The authors of OpenPose propose a greedy relaxation of the problem that obtains good results thanks to the definition of PAFs: since the PAFs are learned through a CNN architecture featuring a large receptive field, the authors speculated that it could be possible to solve the matching problem as a set of independent bipartite matching problems, *i.e.* by computing the score of each candidate limb independently from all other limbs in

[2] A keypoint such as an elbow or a shoulder may not have a unique pixel location, in fact, depending on the scale of the subject, it may be totally arbitrary which pixel to choose as "keypoint location".

[3] One must keep in mind that every detected keypoint corresponds to a peak in the heatmap corresponding to a given type of joint, so each detection is implicitly labeled with the type of node it represents. Thanks to this categorization, the problem simply becomes a matching problem.

Figure 2.2: Fictitious representation of a heatmap given as output by a pose estimator. This representation wishes to highlight several characteristics of a heatmap: *non-normalization* (the sum of all values is not 1), *noisy values* (especially far from the peaks, where negative values are common), *multimodality* (it is possible to have several peaks in a heatmap, if the same type of skeletal keypoint has been detected in multiple locations). Multiple detections of the same keypoint may merge togther to create a single peak with wider support.

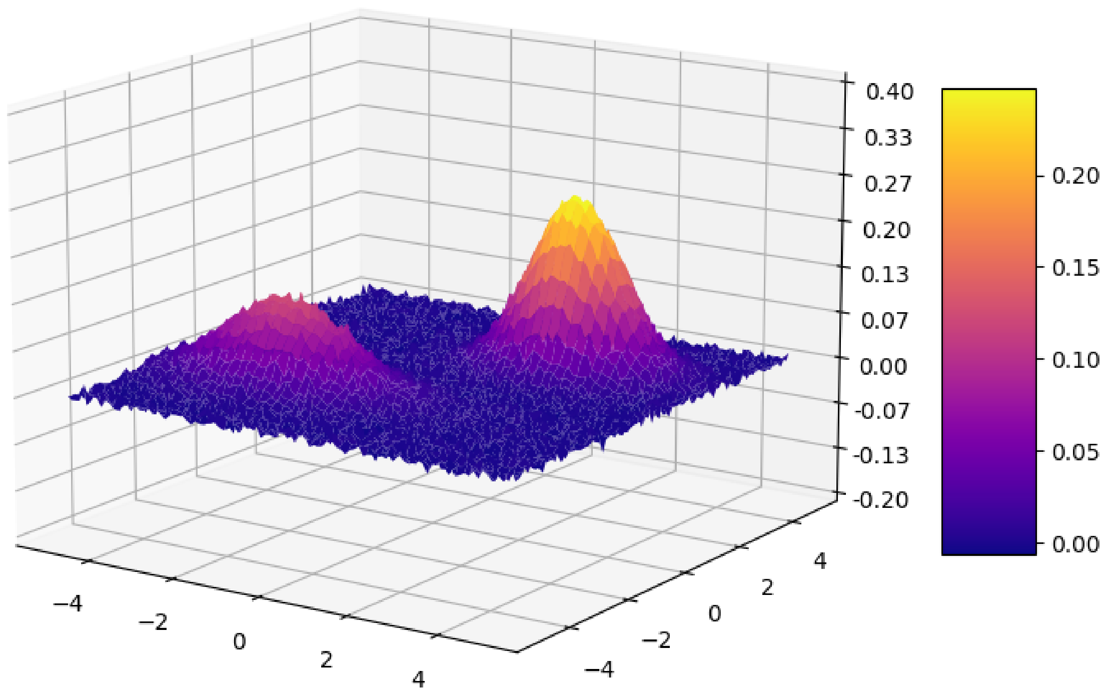
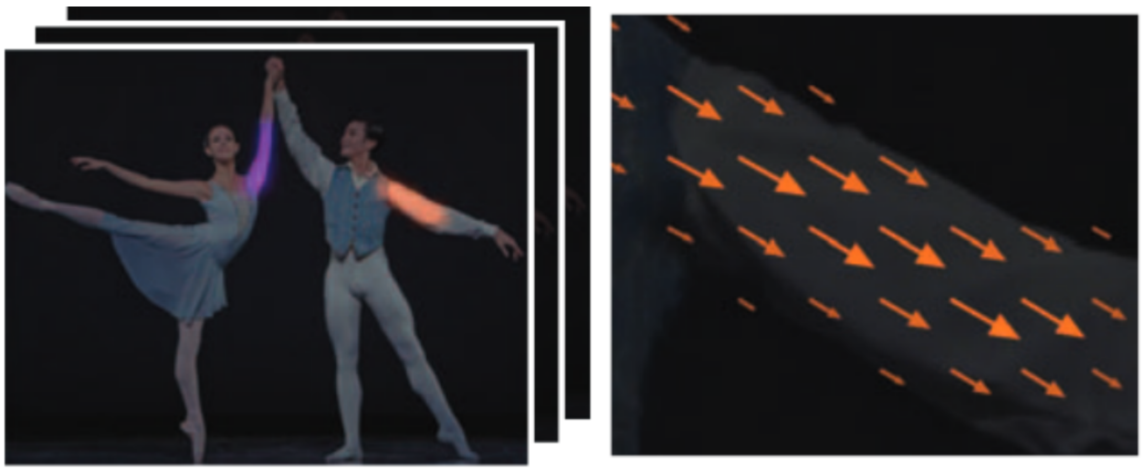


Figure 2.3: The PAF between the shoulder and elbow keypoints is designed to contain vectors that lay on the arm and point from the shoulder to the elbow. Taken with permission from [14] © 2017 IEEE



the same skeleton, and choosing the maximum among all candidate scores.[4].

### 2.1.3 Visual Transformers

While pose estimation based on CNN-extracted heatmaps and part affinity fields has been proven effective, (especially after incremental improvements), other researchers have explored different architectures, often motivated by the need to either increase accuracy or to simplify the post-processing steps needed for human parsing and reduce the number of parameters in the CNNs.

Transformers are a Deep Learning architecture different from CNNs, originally built for Natural Language Processing applications [17]. The transformer architecture has the ability to model long-term dependencies in sequential data (such as natural language sentences), overcoming the limitations of recurrent neural networks in this regard, using a building block called *attention head*, able to handle complex dependencies within input data (based both on the information contained in the data and on the *position* such information occupies within the input). Similarly to how CNNs are usually pre-trained on classification tasks and then fine-tuned to the specific application they are being built for, Transformers may be pre-trained also on *reconstruction tasks*, such as cloze (fill-in-the-blank) tasks, called *Masked Language Modeling* in the case of Natural Language modeling.[5]. A notable advantage of this kind of task is that it does not need annotated data, making the gathering of training data extremely easy and accessible.

Visual transformers, *i.e.* the application of the Transformer architecture to Computer Vision tasks, was initiated around 2020 by seminal papers [18] and [19]. A comprehensive survey of existing Visual Transformer models has been conducted by Liu et al. [20]

As shown in ViTPose [21], training a Visual Transformer for human pose estimation allows for usage of a model with better performance and throughput when compared with top-down CNNs. ViTPose authors also focus part of their work on scalability, since they build four different models with a number of parameters varying between 100 millon and 1 billion, with accuracy and inference time varying proportionally.

In [22], the authors propose instead PETR, a so-called end-to-end trainable human pose estimator, which uses the Transformer architecture on image features extracted

[4]The score of a candidate limb having starting point $\mathbf{j_1}$ and endpoint $\mathbf{j_2}$ is computed as the line integral of the PAF along the straight path from $\mathbf{j_1}$ to $\mathbf{j_2}$

$$\int_{\mathbf{j_2}-\mathbf{j_1}} \mathbf{PAF(s)} \cdot d\mathbf{s}$$

[5]This is due to the fact that the Transformer architecture is intrinsically an encoder-decoder network: the encoder module is used to extract *embeddings* which incorporate contextual features from the whole input sequence, while the decoder exploits mutual dependencies within the extracted features.

by CNNs, leveraging two custom decoders in the transformer network to parse human poses. The improvements in performance with respect to top-down methods are significative, but so is the number of parameters, and the reported frame processing time is still significantly longer than optimized bottom up implementations (the reported speed on high-end hardware -Nvidia Tesla V100- is 65 ms per frame (around 15 FPS), while bottom-up implementations were found to reach 30 FPS, while [23] report 133 FPS on a 2080Ti).

## 2.2 3D human pose estimation

### 2.2.1 The challenges of 3D HPE

3D human pose estimation is a different and more complex problem when compared to 2D HPE: in 2D HPE, skeletal joints must be detected only from pixel locations in the image, and may be disregarded if outside of the picture; in 3D HPE, on the other hand, the possible joint locations may belong to a 3D (thus higher-dimensional) search space, and it is often not possible to disregard the occluded joints. Furthermore, the 2D-to-3D joint detection is an ill-posed regression problem, since many possible 3D poses may be associated to the same 2D projection.

Another pressing issue of 3D HPE is the lack of diverse training data with high-quality ground truth information. The main source of training data for 3D HPE is Human3.6M [24], a dataset of videos featuring 11 professional actors (each recorded in isolation) performing a variety of actions in an indoor setting. The ground truth values of 3D skeletons were collected using a motion capture system. Human3.6M remains one of the largest datasets for 3D HPE to date, despite having been released in 2013, because data collection using motion capture remains expensive and time-consuming. Other datasets for 3D HPE have been proposed, but they are either small-scale datasets or large-scale datasets with lower quality annotations.

A workaround solution for solving the issue with the lack of data is using synthetic data, *i.e.* human figures generated with computer graphics. This is the case for the SURREAL dataset [25]. Many commercial solutions for generation of synthetic data have been proposed in recent years (for instance, the Unity game engine provides packages for synthetic generation of humans [26] [27], and Nvidia offers a software called Omniverse Replicator).

### 2.2.2 Pose lifting vs end-to-end training

3D HPE models may be divided into two categories: pose-lifting networks and end-to-end trainable models. *Pose lifting* or *two-stage* models learn to predict 3D poses from 2D poses. While the learning process may be arguably facilitated by having a 2D skeleton, this approach entails one obvious disadvantage: errors in 2D pose estimation will be compounded with the ones from the pose-lifting module. On the other hand, pose-lifting modules are often lightweight and offer much faster inference, even when combined with a 2D pose estimator (this is the case for [28]). Some recently proposed two-stage models such as [29], which uses a diffusion decoder regularized by means of the 2D skeleton, are not, strictly speaking, pose-lifting models, but share the same weakness.

Other researchers have instead investigated *end-to-end* 3D HPE, in which the raw image input is used to directly extract 3D skeletons. This approach suffers instead from difficulty of training, since the direct extraction of 3D skeletons is usually made by means of volumetric information (*e.g.* Luvizon et al. [30] propose to use volumetric heatmaps, generalizing the heatmap concept from 2D pose estimation): processing higher-dimensional data greatly increases the number of trainable parameters in the network[6].

### 2.2.3 Capsule Neural Networks for 3D HPE

**An overview of Capsule Neural Networks** Capsule neural networks were originally proposed by Geoffrey Hinton et al. as an alternative to CNNs, in the seminal paper [31]. Capsules are groups of neurons (they may also be called ”vector neurons”) that store structured information about a generically defined ”feature” in an image[7]. This vector or matrix of information is generally interpreted as a transformation matrix representing the pose of the detected feature in the image.

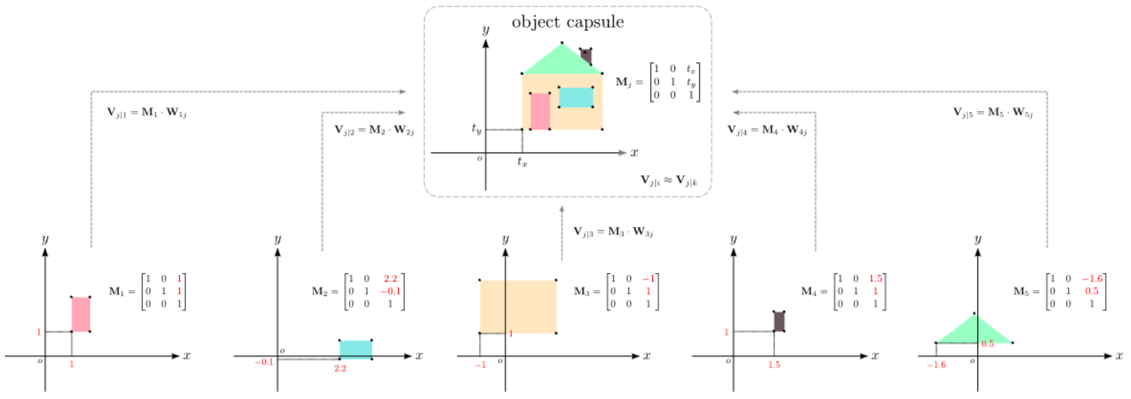
Capsules were proposed to overcome the limitations of CNNs, which are not intrinsically trainable to recognize the single parts of complex objects, and, unlike humans, base their object recognition properties on texture, not on part-whole hierarchies[32]. In particular, [31] criticizes the CNN max-pooling operator as the main culprit, since it foregoes spatial information within the input, but at the same time recognizes there are not many other ways of reducing dimensionality, given that neurons in standard neural networks can only have a scalar activation value.

The fundamental idea introduced by capsule networks is using the capsules to learn a *voting matrix* between consecutive layers of the neural network, which explicitly models the part-whole relationships between consecutive layers (from pixel-level features (edges, corners) up to complex objects). The voting matrix is then multiplied by the capsule matrix in a very similar way to how weights of subsequent layers are computed in standard neural networks. The capsules and voting matrices may be interpreted as encoding, respectively, the absolute pose (position and rotation) of the feature and the relative pose of the feature within the object detected by the capsule in the following

[6]this phenomenon is also known with the colloquial name *curse of dimensionality*

[7]The similarity with classical features (*e.g.* SIFT) is striking, as each feature is typically coupled with a vector that acts as a descriptor, and codifies information about the feature keypoint.

Figure 2.4: The single parts of the object are recognized by lower capsules, while the whole object is recognized by the higher capsule. Viewpoint invariance is achieved by recognizing the single parts of the object and abstracting away the different instances of the same object: in fact, the pose matrix of the single parts is absolute, but the voting matrix only includes pose information of the single part relative to the object. Image from [33]


object capsule
$\mathbf{M}_j = \begin{bmatrix}1 & 0 & t_x\\0 & 1 & t_y\\0 & 0 & 1\end{bmatrix}$
$t_y$
$t_x$
$o$
$x$
$y$
$\mathbf{V}_{j|i} \approx \mathbf{V}_{j|k}$
$\mathbf{V}_{j|1} = \mathbf{M}_1 \cdot \mathbf{W}_{1j}$
$\mathbf{V}_{j|2} = \mathbf{M}_2 \cdot \mathbf{W}_{2j}$
$\mathbf{V}_{j|3} = \mathbf{M}_3 \cdot \mathbf{W}_{3j}$
$\mathbf{V}_{j|4} = \mathbf{M}_4 \cdot \mathbf{W}_{4j}$
$\mathbf{V}_{j|5} = \mathbf{M}_5 \cdot \mathbf{W}_{5j}$
$\mathbf{M}_1 = \begin{bmatrix}1 & 0 & 1\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_2 = \begin{bmatrix}1 & 0 & 2.2\\0 & 1 & -0.1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_3 = \begin{bmatrix}1 & 0 & -1\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_4 = \begin{bmatrix}1 & 0 & 1.5\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix}$
$\mathbf{M}_5 = \begin{bmatrix}1 & 0 & -1.6\\0 & 1 & 0.5\\0 & 0 & 1\end{bmatrix}$
1
-0.1
2.2
-1
1.5
-1.6
0.5

layer.

If many capsules from the lower layer (*i.e.* the layer closer to the input of the network) give similar outputs when multiplied by the voting matrix of a capsule in the higher layer, then the capsules in the lower layer "agree", and the higher layer capsule is likely to represent an object composed of parts encoded by the capsules in the lower layer. The procedure for learning the voting matrices, called *capsule routing*, computes the "agreement" between each pair of capsules in adjacent layers with a clustering-like procedure and then modifies the weights in the voting matrix in order to disregard the input of the lower capsules which do not identify parts of the higher capsule.

Despite many recent advances [34], [35], [36] the research on capsule neural networks has been severely hindered by the absence of efficient algorithms for capsule routing. Training time of capsule networks is not comparable with the training time of convolutional neural networks, even though the number of parameters in capsule neural networks is significantly smaller, both because of the time complexity of such algorithms and because specialized hardware and software for neural networks does not include standard implementations of functions used in capsule networks, (Barham and Isard in [37] additionally argue that specialized kernels for GPUs and TPUs are hard to design and take a very long time to compile, thus resulting in lower overall productivity when compared to CPU kernels).

**Capsule neural networks as 3D human pose estimators** The ability of capsule neural networks to learn viewpoint-equivariant representation of complex objects make them a good candidate for human pose estimation: in fact, one of the biggest challenges of 3D HPE has been the lack of viewpoint-independent training sets.

Garau and Conci [38], building on the work by Ramirez et al. [39] propose a capsule neural network for human pose estimation, using matrix capsules and a capsule routing algorithm based on a variational Bayesian approach, as proposed by [40]. In particular, their architecture is composed of a CNN encoder followed by a capsule neural network further refining the encoded data. The learned latent representation of the input is then fed into three independent decoders, used to jointly predict 3D keypoints, 2D keypoints and heatmaps from the capsule-level encoded features. The loss function used for training the network is a multi-task loss based on the three decoding tasks being used.

The authors report interesting generalization properties of the network, especially when evaluating the 3D pose estimation from unseen points of view, and especially considering the low amount of training data needed to reach qualitatively acceptable result. However, the performance of this network remains lower than the state of the art (the reported mean per-joint position error on Human3.6M is of about 86mm with a standard deviation of 15.86, while MeTRAbs [41], one of the popular CNN-based methods for 3D HPE, is around 50mm with a much smaller standard deviation of 0.7 mm).

## 2.3 Uncertainty in human pose estimation

When considering the concept of ”uncertainty” for a human pose estimator, many different concepts may assume this same name. For instance, deep learning models with a classifier head feature one neuron for each class in their output layer. The activation values of the neurons (”logits”) are usually interpreted as ”unnormalized probability values” for each class, however this interpretation is misleading, as deep learning models tend to be overconfident and unaware of out-of-distribution inputs [42]. From a theoretical standpoint, given an image, a predicted 3D human pose $\mathbf{x}$ will in general be different from the real 3D pose $\mathbf{x}^*$. A definition of ”uncertainty” when it comes to human pose estimation could be intuitively defined as $||\mathbf{x} - \mathbf{x}^*||$, where the ”minus” sign is shorthand for any evaluation metric that measures the distance between two 3D skeletons (for instance, it could be per MPJPE, mean per joint position error, *i.e.* the mean value of the distances between ground truth and predicted joints). So, quantifying uncertainty in human pose estimation requires a measure that takes into account not only the predicted keypoints in isolation, but also take their mutual relationship into account.

Several previous works have tried to estimate the uncertainty in human pose estimation; however, none of the approaches investigated focuses on real-time uncertainty estimation: most of the papers either focus on the issue from a theoretically sound standpoint or propose purely learning-based approaches (with a considerable number of parameters), tackling the issue without paying much attention to the complexity and execution time of the proposed algorithms, which are mostly meant to be run in a off-line fashion.

Gundavarapu et al. [43] propose a 2D HPE approach for uncertainty estimation using a Gaussian Neural Network, *i.e.* a CNN with three fully connected prediction heads, one for the mean and two for the covariance matrix, having the task of predicting a 2D Gaussian distribution over the keypoints of the human body. The mean branch is first trained on ground truth joints of the 2D split of Human3.6M. The covariance heads are subsequently trained by performing multiple predictions of the joint positions using the mean prediction head and then using a maximum-likelihood approach to fit the Gaussian covariance matrix over the predicted points.

Another learning-based approach was proposed by Kundu et al. [44], who propose a multi-head predictor that evaluates 3D HPE uncertainty by evaluating the agreement among predictions using an ensemble-like architecture.

Liu et al. [45] propose instead to evaluate the uncertainty of human pose estimators by computing the mutual information among adjacent frames after an automated alignment procedure among them. While the system exhibits good performance qualitatively, the alignment process and mutual information calculation are CPU-intensive and introduce a considerable delay in the processing.

Finally, Loquercio et al. [46] propose a general framework for uncertainty estimation in neural networks that implicitly creates an ensemble of predictors by applying dropout at test time. This method is both theoretically interesting and practically motivated, especially considering it does not require retraining of existing models and does not significantly impact the inference time; however, the technique does not apply directly to Human Pose Estimation models, which apply considerable post-processing after the pure neural network, whose output should also taken into consideration.

## 2.4 Practical literature review

Onboard computation in robots is by nature resource-constrained, and the issue becomes even more pressing in the case of UAVs, which are only able to carry a limited battery payload. Nevertheless, while a large number of pose estimation models have been proposed throughout the years, many of them were not developed with inference speed in mind, even less so on low-power devices, and powerful hardware is needed by the overwhelming majority of HPE models in order for them to reach real-time performance (targeting 30 FPS).

The theoretical literature review exposed in the previous sections was complemented by a practical literature review, with the main aim of answering the questions: *will it run on an embedded device? Will it have real-time performance?*

At the University of Twente, one Nvidia Jetson[8] TX2 was available for practical experimentation in this regard, so it was adopted as the baseline for the evaluation of the available models.

The default system configuration, with Jetpack 4.6.1, CUDA 10.2, Python 3.6, PyTorch 1.11, is pretty inflexible to guarantee system stability and compatibility with the custom hardware of the Jetson platform. This fact, however, resulted in severe compatibility issues with most of the software being experimented with, even when dockerized.

### 2.4.1 3D HPE

None of the tested models was found to suit the needs of the project, below are reported some details about each model and the reason for its rejection.

**MeTRAbs** MeTRAbs [41] is an end-to-end trainable 3D HPE algorithm that predicts 2.5D heatmaps in a metric space that is in no way constrained by the image size or by the visible joints in the image. This technique results in predictions robust against truncation. Qualitative results on in-the-wild recorded camera data were surprisingly good, especially when taking into account cropped frames that are typical in human-robot collaboration. The runtime of the model during inference was however unsatisfying (0.17 FPS) and unapt to real-time applications

The performance of the 3D end-to-end model focused the search on pose-lifting models, which have considerably fewer parameters and should then have better performance.

**MixSTE** The authors of MixSTE [47] propose a model exploiting temporal correlations (which are present in the live camera feed of a drone) to lift 2D skeletons to 3D with a high-FPS solution. The network architecture is based on transformers and exploits both inter-joint correlations and temporal correlation by means of specialized attention blocks. MixSTE was considered promising because of its high theoretical framerate, however the dependencies of the open source release of the project were not satisfiable on the Jetson platform. In particular, some parts of the source code needed to be compiled using Numba [48], a just-in-time compiler based on LLVM. LLVM is not available on ARM platforms (attempts to compile it directly on the Jetson failed).

**Gast-Net** Graph Attention Spatio-Temporal convolutional Network [49] uses as input a video sequence, and models the 2D-to-3D problem as a graph inference problem in

[8]Energy-efficient GPU-accelerated devices for deep learning applications are produced by Nvidia under the commercial name Jetson. Jetsons are single-board computers with an Nvidia Gpu and an ARM processor, equipped with a customized version of Ubuntu called Jetpack, containing drivers and software dependencies for the deployment of deep learning workloads.

the graph defined by the 2D pose estimator output, using self-attention within the graph and then performing a convolution (called ”temporal convolution”) between the outputs of subsequent frames. Deployment of Gast-Net was hindered by unavailability of Numba, as for MixSTE. Additionally, support for 2D batch normalization in the ONNX interpreter shipped with JetPack was not present, so attempts to package the model into an ONNX and then import it for inference have failed.

**Single-shot multi-person human pose estimation** Mehta et al. [50] propose a representation of the CNN output called Occlusion-Robust Pose Maps, which include redundant information about 3D positions of joints at different levels of details (from a generic torso + limbs models down to single joints). The NN architecture is trained to output such representation during inference, and the representation is then deterministically parsed using 2D heatmaps and PAFs from a separate 2D pose estimator, trained separately from the 3D branch. An open-source implementation of this method by Mariia Ageeva[9], optimized for inference on Nvidia Embedded devices, offered qualitatively good performance but low inference speed (no more than 6 fps).

**VideoPose3D** [51] is a CNN-based pose lifter which uses dilated temporal convolutions to perform inference among subsequent 2D skeletons. In particular, to reduce the number of parameters, 2D poses are simply encoded as pixel coordinates, and treated as a time series, which is then processed by a 1D convolutional block. The standard setup of VideoPose3D is not strictly speaking real-time, because temporal convolutions also take into account future data (causal convolutions would be needed in this case). The introduced delay is of 13 frames, since the receptive field for the CNN is set to 27 frames. The runtime performance of the pose lifter was assessed to be very good (around 80 FPS), thus making the 2D pose estimator the real bottleneck. Qualitatively, results were however not as good, especially in frames where the human body is cropped or otherwise occluded, which are unfortunately quite common for a collaborative robot. In particular, it seems that the model is overfitting on characteristics of the training set (Human3.6M): in fact, in videos where the human limbs are completely visible, performance seemed to be good or very good, while upon occlusions or cropping, or in unusual points of view, there was a substantial performance drop, with 3D skeletons exhibiting jitter and unnatural or impossible poses.

### 2.4.2 2D pose estimation

**OpenPose and jetson-inference (TRT-pose)** The model of choice for 2D HPE on Jetson platforms is provided directly by Nvidia and is based on the OpenPose architecture, using a different backbone for CNN feature extraction (VGG14 [52]) with a

[9]`https://github.com/Daniil-Osokin/lightweight-human-pose-estimation-3d-demo.pytorch`

reduced input size (54x54) and a parallelized algorithm for heatmap peak search. This model yields the best speed-accuracy tradeoff among all the models considered in the practical literature review, reaching 40 FPS at the expense of inaccurate pose estimation in certain contexts. On the contrary, OpenPose uses a Caffe[53]-based backbone; installing Caffe in modern Linux distributions (even more so in the case of JetPack) is a long process requiring a lot of trial-and-error steps. A performance test of OpenPose performed on a machine equipped with a desktop GPU (Nvidia GTX 1080Ti) showed a model speed of around 8 FPS using the default (higher-accuracy) settings, and went no further than 18 FPS when tuning the heatmap sizes before severe performance degradation.

**ViTPose** The scalability of ViTPose [21] allowed for testing of a small scale model (ViTPose-small), having performance around 10 FPS but much worse accuracy when compared to the OpenPose-based models. Despite being advertised in the paper as a scalable and lightweight model, this visual transformer model does not seem to be mature enough yet for deployment to edge devices: in fact, conversion of the model to TensorRT to guarantee faster inference and lower memory footprint failed in the JetPack 4.6.1 setup because of unsupported operators. Moreover, upon closer inspection, the (much better) FPS values reported in the paper leveraged batched inference, which was instead disabled in the experiments, since it introduces undesired delay and is not suitable to live camera streams.

| Model name | FPS | Model type | Notes |
|---|---|---|---|
| MeTRAbs | 0.17 | 3D | |
| MixSTE | ? | 2D-to-3D | could not run on Jetson |
| Gast-Net | ? | 2D-to-3D | could not run on Jetson |
| Single-shot multi-person HPE | 6 | 2D-to-3D | |
| VideoPose3D | 80 | 2D-to-3D | Performance refers to pose lifter alone (without the 2D module) |
| OpenPose | ? | 2D | Could not run on Jetson. Achieved 18 FPS on a different system with Desktop gpus |
| Nvidia TRT pose / Jetson inference | 40 | 2D | |
| ViTPose | 10 | 2D | |

Table 2.1: Results of the comparison between different models. FPS as evaluated on Nvidia Jetson TX2

As will be explained in the next chapter, the practical literature review led to choosing a high-framerate 2D pose estimator for a Jetson TX2, and subsequently adapt it for 3D detection by means of a depth sensor. In fact, while 3D HPE models would be more helpful for the purposes of a robot-mounted human pose estimator, the high complexity of the 3D models leads to unacceptable performance for real time operation. 2D-to-3D lifting modules could be coupled with the TRT pose estimator, however none of the models tested achieved good performance. It is worthwhile to note that developing a pose lifter especially targeted at Jetson platforms could be a valid solution, especially if it is possible to run the inference with TensorRT.

# 3 | Pose estimator and QoE heuristic

In this chapter, the pose estimator and its integration with the QoE heuristic computation will be explained in detail. Then, the experimental setup for validation of the system will be introduced and some results will be presented.

## 3.1 2D Human pose estimator

As outlined in Chapter 2, a prolonged testing of available 3D pose estimators showed that real-time performance ($\geq$ 30 Hz) was unattainable on the Jetson, because of limitations in computing power. The 2D pose estimator was based on an off-the-shelf 2D pose estimator provided by Nvidia in the jetson-inference software package.[1]

This pose estimator performs inference on one frame at a time, not exploiting the temporal correlation between frames in a video stream, and makes use of heatmaps and part affinity fields, as first suggested in [14][15]. The model is pretrained and ready for deployment on the Jetson platform, making it much easier to work with on an aarch64 platform such as the Jetson TX2. The model is written using the C++ PyTorch API for performance reasons and has been exported to ONNX format and optimized for inference using TensorRT[2].

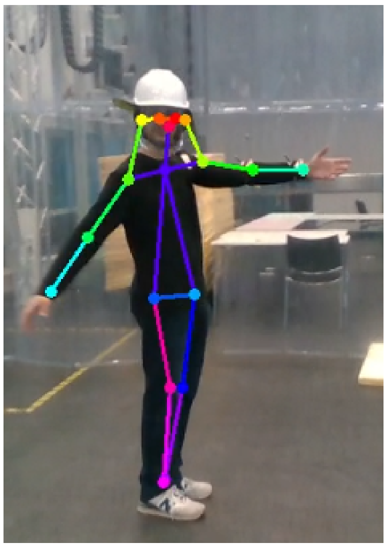
## 3.2 2.5D to 3D augmentation

For the purposes of human-robot handover, the positions of human joints must be expressed in a 3D reference system, while the output of the pose estimator, combined with depth information coming from the stereo camera gives us a 2.5D representation of the keypoints (pixel coordinates and orthographic distance of every pixel from the image plane). Reprojecting 2.5D data to 3D data was performed on undistorted images, using standard extrinsical calibration equations. The equation used for deprojection

[1] `https://github.com/dusty-nv/jetson-inference/releases`

[2] a proprietary framework by Nvidia, which allows users to quantize the network weights (e.g. from float32 to float16), effectively reducing the memory footprint of the processed model while keeping similar performance. Other CUDA-specific optimizations are also applied, such as layer fusion (to offer a speedup during inference) and dynamic tensor memory (a speedup technique that avoids or defers tensor memory allocation and deallocation if memory reuse is possible)

Figure 3.1: The COCO-18 skeletal model used by the pose estimator. The 18 keypoints are: nose, eyes, ears, shoulders, elbows, wrists, hips, knees, feet, neck.



is simply derived from the pinhole camera model without distortion, in homogenous coordinates, as follows:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} u' \\ v' \\ z \end{pmatrix} = \mathbf{K} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} u' \\ v' \\ z \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

So:

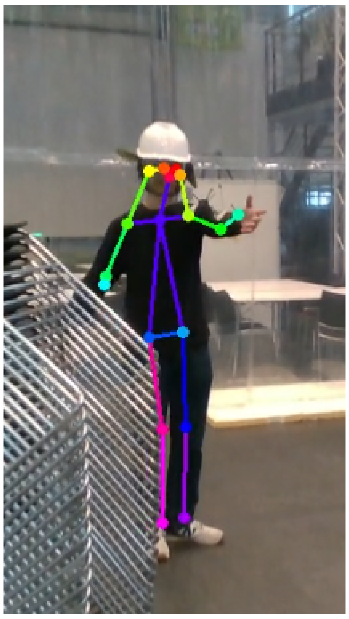
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{u' - zc_x}{f_x} \\ \frac{v' - zc_y}{f_y} \\ z \end{pmatrix}$$

And finally, noting that $u' = uz$ and $v' = vz$ (since we wish to forego the multiplicative constant of the homogenous coordinates, given that $(u, v)$ are not scaled in the input)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{z(u - c_x)}{f_x} \\ \frac{z(v - c_y)}{f_y} \\ z \end{pmatrix}$$

where measured quantities are $u$ and $v$ (the pixel coordinates of the point being deprojected), and $z$ (the depth value of the required pixel in the RGBD frame). $\mathbf{K}$ is the

Figure 3.2: In this frame, it can be noted that the pixels of the occluded joints (right knee, ankle and hip) actually correspond to occluded locations, and their depth measure will therefore be severely skewed.



intrisic camera matrix, provided by the Intel RealSense self-calibration process. All code was written in C++, using the RealSense API and the image processing library OpenCV.

To improve the model's robustness to noise in the depthmap, the $z$ value is not computed from a single pixel, but it is instead calculated as the average value in a 7x7 neighborhood of the pixel, after removing outliers (*i.e.* values outside of the range $[1.5, 15]$ meters, corresponding to the manufacturer-approved operating range for the stereo camera, with the lower bound incremented from 0.3 to 1.5 meters because for the application considered, the drone may never be too close to the human).

It must be noted in the above equations that the depth measure given as output by the RGBD sensor was assumed to be the true value. This is a simplifying assumption that is often incorrect: in fact, even if we assume the depth measures to have no noise, occlusions and (most importantly) self-occlusions will give rise to systematic errors, especially considering that the 2D pose estimator will often assume the presence of a joint even when the location of the joint is slightly occluded (see Figure 3.2).

## 3.3 Quality of Estimation heuristics

Several sources of error impact the estimation quality of the pose estimation pipeline exposed so far.

The 2.5D to 3D lifting of the poses suffers from the lack of joint training on RGB and depth data: a relatively small error in terms of pixel coordinates can translate to a very large error in metric coordinates because the deep learning model has not been trained to choose either side of sharp edges in the depth maps, such as the ones that could be found in Figure 3.2. In the second place, depth maps collected by the RGBD cameras are often noisy. Finally, a disadvantage of both the pose estimator and the depth camera is that inference and stereo matching are performed on single frames, disregarding the temporal correlation between frames.

One of the main goals of this thesis work has been finding a heuristic to find out on a joint-by-joint basis whether the estimated 3D position of the keypoint is accurate (so it can be relied on), while also keeping the resource demand as low as possible for computing the heuristic in real time on resource-constrained devices.

Three main sources of information for evaluating the uncertainty quality have been devised:

- Heatmaps in the pose estimation model: this is because heatmaps represent all the locations in which a given joint has been detected in the image, so, under the assumption that only one person will be present in the frame at all times, multiple detections mean that the model is more uncertain on which parts to choose. Moreover, the ”height” of the peaks of the heatmap represents a measure of the ”confidence” of the model, because the heatmap is not normalized.
- Distance-based priors: they are essentially priors on body part length, devised because it is reasonable to assume some limits to the size of the body parts of the humans (*e.g.* an arm cannot be 3 meters long).
- Temporal correlation between frames: since data is captured from a camera feed, temporal correlation information may be used to detect intermittent or jittery (and thus more uncertain) measures.

The uncertainty measure for a given skeletal keypoint in a given frame is thus computed as the mean of three components, which will be detailed in the following.

### 3.3.1 Heatmap-based QoE heuristics

Since the chosen pose estimator used a heatmap-based method, heatmaps were the natural candidate for evaluating the model confidence. It must be noted that the area of the support of the peaks in the heatmap, which might be erroneously interpreted as the ”variance” of a Gaussian, does not always have a direct connection with the confidence of the estimation, since the spatial extension of keypoints varies significantly in the input images.

Extracting the heatmaps from GPU memory and exposing them for processing was performed by trial-and-error, modifying the source code of the pose estimator model, since no development documentation was provided with the model.

In the following, the assumption that the size of the heatmap is equal to the size of the image will be made for simplifying the exposition; nevertheless, the size of the

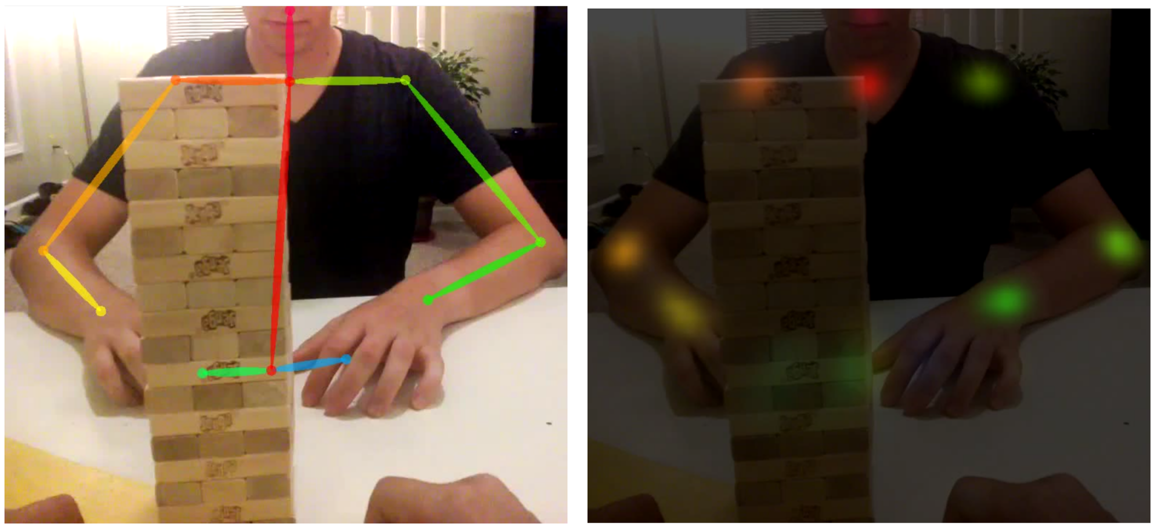


Figure 3.3: Pose estimation output (left) and heatmaps (right). The intensity of the colors in the heatmap representation is proportional to the value in the heatmap, *i.e.* to the unnormalized probability of finding the joint in the given pixel. The heatmaps have higher peaks where the keypoint is detected with higher confidence.

heatmaps in the pose estimator used did not match the size nor the aspect ratio of the input (during the experiments, the heatmap was a square matrix of size 28x28). In such cases, mapping the bin $(\tilde{u}, \tilde{v})$ to pixel $(u, v)$ is possible: if an image of size $\mathrm{rows}, \mathrm{cols}$ has been mapped into a heatmap of size $(\mathrm{binrows}, \mathrm{bincols})$, then

$$(\tilde{u}, \tilde{v}) = \left( \left\lfloor \frac{u \cdot \mathrm{binrows}}{\mathrm{rows}} \right\rfloor, \left\lfloor \frac{v \cdot \mathrm{bincols}}{\mathrm{cols}} \right\rfloor \right)$$

**Peak mass ratio** Each heatmap may approximate a multimodal distribution, because the same joint could be detected in multiple positions in the image (either because of multiple people being present in the input, or because of false positives). Furthermore, selecting the skeleton with the most keypoints, only one joint per type will be present in the output, and the pixel location of such joint will correspond to the peak of one of the modes in the heatmap. Because of this, one heuristic for quantifying the estimation quality could be intuitively expressed as

$$\frac{\text{mass of the mode}}{\text{total mass of the heatmap}}$$

"Mass" is here a term improperly used as though the heatmaps were real probability distributions: the "probability mass" of a heatmap is the sum over the whole image, while the "mass of a peak" is the sum over the support of the peak. The formal computation of such quantity proceeds as follows: let

$$H_i(u, v) : \mathbb{N}^2 \ni (u, v) \mapsto q \in \mathbb{R}$$

be the heatmap corresponding to the $i$-th skeletal keypoint, mapping pixel coordinates $(u, v)$ into the unnormalized joint probability $q$. Note that $q$ is in general unbounded (the preprocessed data used for training prescribes $q \in [0, 1]$ but the actual value may slightly exceed these bounds because of numerical errors and because no mechanism is implemented in the loss function to force the weights to output values comprised in this range[3]).

A definition of ”mass of the peak” in terms of connected components of the image is particularly convenient, since OpenCV provides a fast and parallelized [4] implementation of the Spaghetti algorithm for connected component labeling [54], [55].

To improve separation between the connected components, and avoid the artifacts due to negative values, the *masked heatmap* $M_i$ is computed as follows:

$$M_i(u, v) := \begin{cases} H_i(u, v) & \text{if } H_i(u, v) \geq (\overline{q_i})_+ \\ 0 & \text{otherwise} \end{cases}$$

where $\overline{q_i}$ is the average value of the heatmap, computed as

$$\overline{q_i} := \frac{1}{\text{rows} \cdot \text{cols}} \sum_{u=0}^{\text{rows}-1} \sum_{v=0}^{\text{cols}-1} H_i(u, v)$$

Note that $\overline{q_i}$ might be negative: to avoid inserting negative values in $M_i$, $\overline{q_i}$ is clipped to zero with the $\cdot_+$ operator.

Finally, let $C_i(u, v)$ be the output of the connected component labeler run on $M_i$, where every pixel $(u, v)$ is associated with an integer associated with the connected component it belongs to. Then, if the pixel output of the pose estimator for the i-th keypoint is $(u^*, v^*)$, the peak mass ratio heuristic amounts to

$$QoE_i^{Mass} = \frac{\displaystyle\sum_{(u,v):\, C_i(u,v)=C_i(u*,v*)} M_i(u, v)}{\displaystyle\sum_{(u,v)\in H_i} M_i(u, v)} \in ]0, 1]$$

where values close to 1 represent high quality of estimation.

**Peak height** Since the probability maps are an output of the neural network, and no normalization process takes place on such output, the height of the peak corresponding to the detected keypoint location correlates with the confidence of the prediction.

[3]in practice, values near 1 almost never occur for any input, while slightly negative values were extremely common occurrences.

[4]The parallelization happens through both vectorized CPU instructions and multithreading (OpenCV 4.7.0 was custom built on the Jetson especially for this purpose, to enable support for the NEON instruction set and the OpenMP parallelization framework). CUDA acceleration is also implemented for whole-image computations (*e.g.* for summing the pixel values of connected components), but the overhead of switching to the CUDA accelerator proved to be overall counterproductive.

Qualitatively speaking, it is possible to observe shorter peaks associated with uncertain keypoints (*e.g.* if they are occluded). For this reason, the height of the peak corresponding to the detected keypoint is an interesting measure.

Using the same conventions expressed in the previous paragraph, we may define the peak height heuristic as:

$$QoE_i^{Height} = M_i(u*, v*) \ \in ]0, 1[^5$$

where values close to 1 represent high quality of estimation.

There are no theoretical guarantees on the range of this heuristic, but several experiments showed that the peak height never came close to the bounds of the range.

### 3.3.2 Distance QoE heuristics

**Keypoints too far from the body** If a keypoint is too far from the average of all other detected keypoints, it is likely that an error in the depth estimation has occurred. In particular, considering the 3D location $\mathbf{k_i} \in \mathbb{R}^3$ of the $i$-th keypoint, it may be useful for uncertainty estimation to compute

$$d_i = ||\mathbf{k_i} - \frac{1}{18}\sum_{j \neq i} \mathbf{k_j}||$$

which represents the Euclidean distance between the $i$-th keypoint and the average of all other keypoints, or "centroid" of the skeleton. For real keypoints, it is not possible to have a value $d_i$ exceeding a certain threshold $T$ (empirically, $T = 1.5$ meters is a value granting acceptable performance).

To compute the heuristic uncertainty based on this measure, function $w$ is used to guarantee some fuzziness and flexibility with respect to the threshold value chosen.

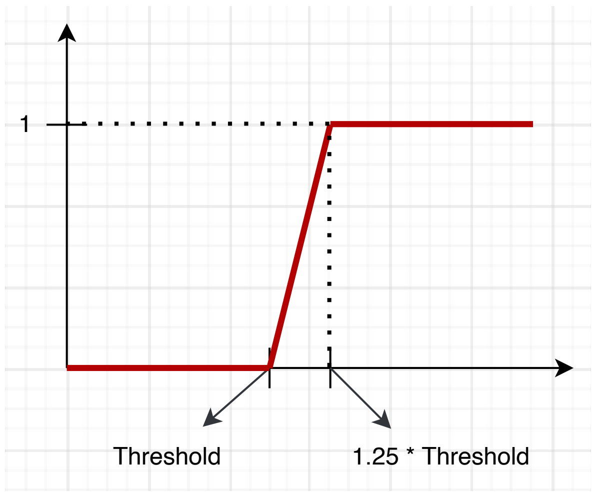
$$w_T(x) = \begin{cases} 0 & \text{if } x \leq T \\ \frac{4}{T}(x - T) & \text{if } T < x < 1.25T \\ 1 & \text{if } x \geq 1.25T \end{cases}$$

$$QoE_i^{Distance} = 1 - w_T(d_i)$$

**Bone length priors** Instead of using the general heuristic described in the previous paragraph, it is possible to leverage some constraints on the shape of the human body for certain keypoints. In particular, some of the links between joints of the human skeletal model being considered are rigid, because they are supported by bones. The

[5]The bounds were determined empirically

Figure 3.4: Graph of the function $w_T$



upper body kinematic chain (from wrist to wrist) is a good candidate for adding constraints to the positions of 3D keypoints in space. Another good reason for paying special attention to the estimation quality of the keypoints of the upper body is that the UAV should need to estimate especially well the position of the arm keypoints in performing a handover.

The $QoE^{Distance}$ heuristic for the keypoints in Figure 3.5 is computed as described in the following.

The range of bone lengths considered acceptable has been chosen arbitrarily to be quite flexible and applicable to adults of different height:

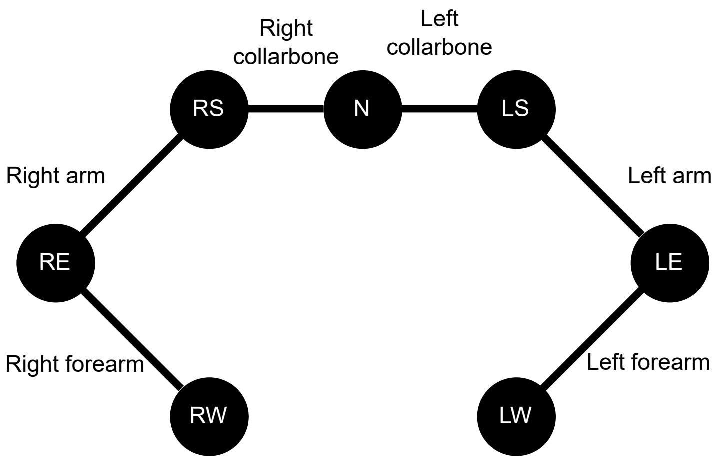
- Forearm length (wrist to elbow): 0.25 to 0.6 m
- Arm length (elbow to shoulder): 0.25 to 0.6 m
- Collarbone length (shoulder to neck): 0.15 to 0.45 m

Given that multiple joints in the kinematic chain may be uncertain, and that the bone length depends on the position of two joints, extra care was taken in avoiding the undue propagation of uncertainty constraints to joints that may not, in principle, be out of the range of plausible positions.

The procedure for assigning a QoE value to each and every keypoint in the kinematic chain follows these rules:

1. If a keypoint was not detected in the current image, then both the bones originating from it are considered to be outside of the acceptable range
2. A keypoint is incorrect if and only if both the bones originating from it are outside the expected range
3. In the case of the wrist, since only one bone originates from it, it is considered correct only if the following two conditions are true at the same time:

Figure 3.5: Diagram representing the upper body kinematic chain, using the joints detected by the pose estimator: wrists, elbows, shoulders and ”neck” (actually *manubrium sterni*).



- The elbow keypoint adjacent to it is considered correct
- The length of the bone originating from it is within the acceptable range

If the elbow keypoint adjacent to it is uncertain, instead, the wrist keypoint will be simply removed from the kinematic chain and its $QoE^{Distance}$ will be computed using the general rule applied to all the keypoints not belonging to the upper body kinematic chain.

An example of the application of the rules specified above is given in Figure 3.6.

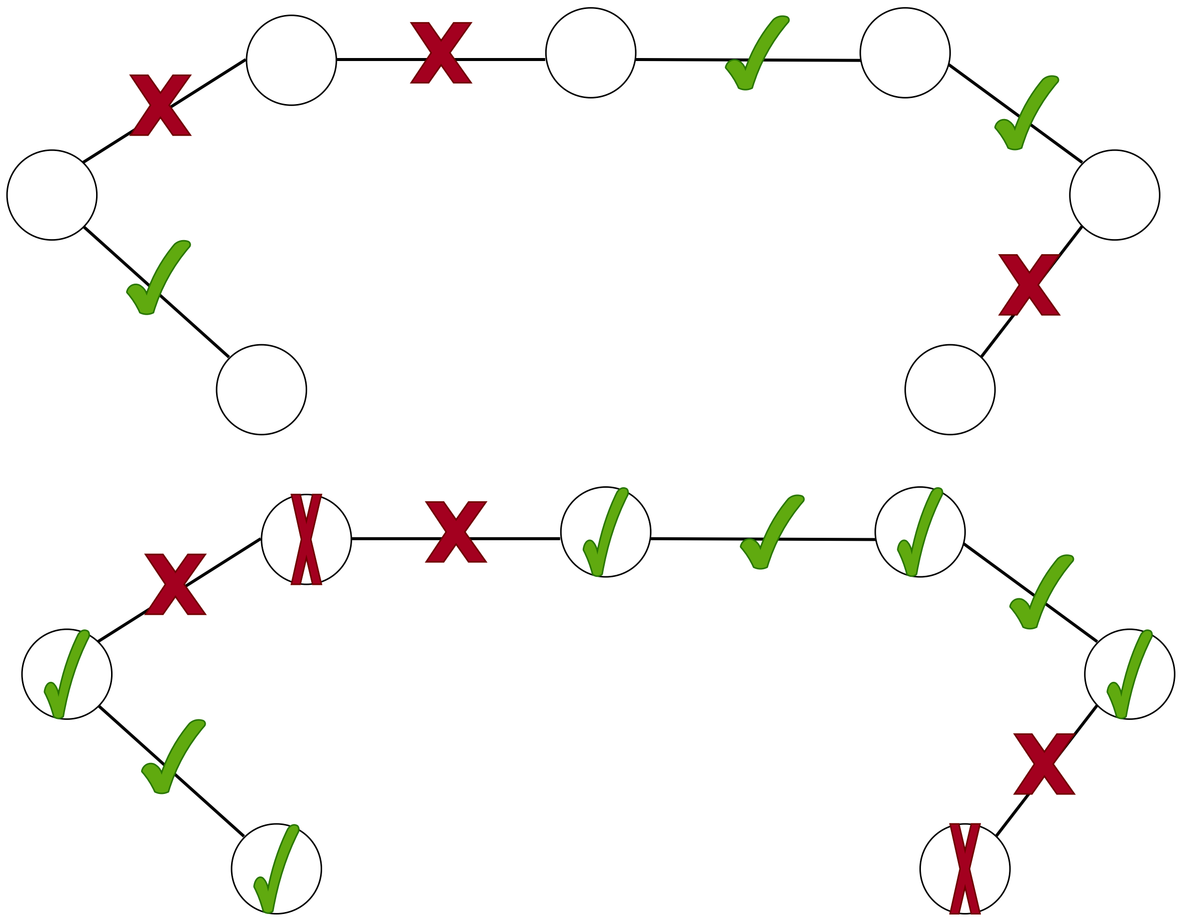
Having labeled every keypoint as ”correct” or ”incorrect”, we proceed to computing the $QoE^{Distance}$ value for each of them. For ”correct” keypoints, $QoE^{Distance} = 1$ (minimum uncertainty). For ”incorrect” keypoints, instead, the following algorithm is used:

- If either bone length is below the minimum allowed length, set $QoE^{Distance=0}$ (maximum uncertainty) to the incorrect keypoint
- If both bone lengths are above the maximum allowed length, compute $QoE^{Distance}$ as $1 - w_T(\frac{e_1 + e_2}{2})$, where the *excess* of bone $i$ $e_i$ is computed as

$$e_i = (\text{length of bone } i) - (\text{max allowed length for bone } i)$$

and $w_T$ is the weighing function described in Section 3.3.2 and used to compute the distance uncertainty heuristic for points not belonging to the upper body kinematic chain.

Figure 3.6: Within-range bone lengths are marked with a green tick, while bone lengths outside of the acceptable range are marked with a red X. The combination of detected bone lengths detected above results in the combination of correct/incorrect keypoints reported below.



### 3.3.3 Combining the QoE heuristics

The three heuristics presented so far are combined with a simple average:

$$QoE_j = \frac{1}{3}(QoE_j^{Mass} + QoE_j^{Height} + QoE_j^{Distance})$$

### 3.3.4 Temporal correlation and intermittent measures

The chosen pose estimator does not exploit temporal correlation among frames, since the keypoints are calculated independently for each frame. As a result, the estimated keypoints exhibit noticeable jitter. To introduce temporal correlation in the estimation, a Kalman filter was implemented, with the ultimate goal of stabilizing the pose estimator output.

**State, observation and transition model** The model used for the Kalman filter was a zero-velocity model, assuming that the human operator will be mostly static during the handover process (*e.g.* waiting for the tool with a stretched arm). The state $\mathbf{x} \in \mathbb{R}^{18 \cdot 3 = 54}$ is thus a vector containing the 3D position of the 18 keypoints being tracked, but no velocities. The state transition model $\mathbf{F} = \mathbf{I}_{54\times54}$ simply states the equation $\mathbf{x_k} = \mathbf{x_{k-1}}$. Since joints positions are observable, the observation model $\mathbf{H}$ was set to be a time-dependent matrix, multiplying by 0 the positions of the joints not detected in the current timestep.

**Process noise and observation noise** Process noise was set to a diagonal matrix with all entries equal to 0.01, *i.e.*$\mathbf{Q} = 0.01 \cdot \mathbf{I}_{54\times54}$. Moreover, whenever an observation is not available, the relevant entries in the process noise matrix are incremented in 0.001 steps. This design aims to model the fact that whenever a joints becomes unobservable for a long period of time, its position cannot be assumed to be simply the same as the one in the previous steps. Observation noise, on the other hand, was set to be time-dependent. Matrix $\mathbf{R}$ was considered a diagonal matrix, but the entries of the matrix were updated at each step with the values of the QoE heuristic.

Using the Kalman filter allowed for filtering of noisy measurements and also to use the values from the prediction step to fill in the blanks of missing measurements. Moreover, the *a posteriori* covariance matrix $\mathbf{P}$ could be theoretically used for uncertainty quantification whenever taking into account the current state of the filtered estimation, and not a single frame. This method, however, was not tested extensively (as the unfiltered heuristic was preferred over the filtered one for testing) and thus its effectiveness remains unknown.

## 3.4 Hardware configuration

The configuration for all experiments was chosen since the beginning to comply with real-world computational and physical constraints based on the realistic scenario of a tilted-propeller hexarotor with a camera mount, able to perform all required computation onboard.

### 3.4.1 Hexarotor

The drone considered is a prototype based on incremental refinements of the Tilt-Hex platform [56], built and operated at the University of Twente. The UAV platform features non-collinear fixedly-tilted propellers, and is thus fully actuated [6]. The drone platform features an additional controller interface, a microprocessor deputed to low-level control of the motors and data passing to and from an IMU module and an external computer for high-level control.

The high-level controller of the drone is a nonlinear MPC originally developed by Martin Jacquet et al. at LAAS-CNRS [58], compilable into one of several middlewares (in our case, POCOLibs, making use of shared memory for interprocess communication). To avoid integration testing and its subsequent challenges, the high-level controller was not run on the onboard embedded computer but on a laptop, making use of Matlab bindings of the code for simulation and path planning. It must be noted that there are no evident obstacles to running the high-level controller on the onboard computer.

### 3.4.2 Onboard embedded computer

The compute module for human pose estimation was chosen to be an Nvidia Jetson TX2, a single-board computer which provides four Cortex-A57 cores and two Denver2 cores for CPU computation, a CUDA-enabled accelerator based on the Pascal microarchitecture, and 8GB of shared memory, completely addressable by both CPU and CUDA. Energy consumption is of particular concern, because the computer should be mounted on battery-powered drones: the TX2 module's achieves a peak energy consumption of 15W, which can be further reduced up to 7.5W by switching off the Denver cores and reducing the clock speed of the CUDA accelerator.

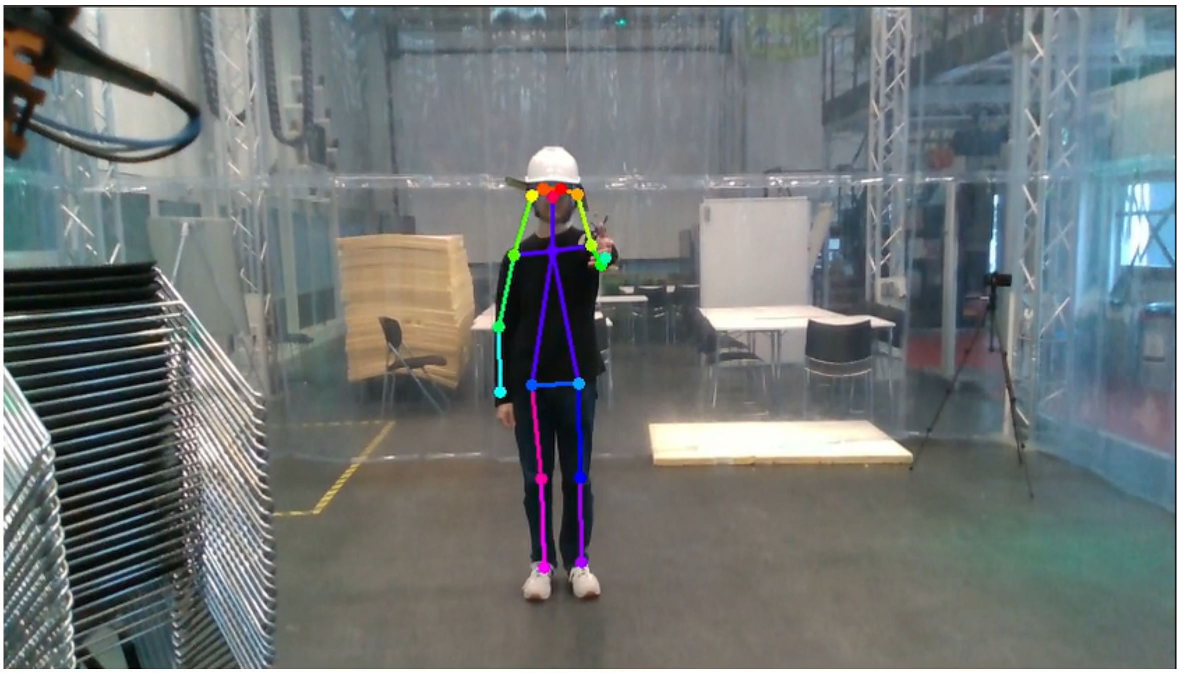
### 3.4.3 RGB-D camera

For the camera module, an Intel Realsense D435i was used. It is a soft-synchronized stereo[7] RGB camera. The camera was connected to the onboard computer by means

[6]general properties of non-collinear tilted-propeller UAVs are investigated in [57].

[7]The two camera shutters are not hardware synchronized, but the onboard ASIC provides a best-effort match based on hardware clock timestamps for both frames. The Realsense API then allows

Figure 3.7: Camera view for the drone-mounted RealSense. The only occlusion is part of the propeller in the top left corner.



of a USB 3.0 connector and then secured under the main body of the drone using a 3D printed camera mount. The position of the mount ensured minimal occlusions by the robot body (less than 3% of the image surface was occluded).

The choice of a stereo camera makes the whole application more flexible, since stereo matching can be performed both indoors and outdoors, while other distance sensors only work properly indoors. The price to pay for this added flexibility is the likelihood of having a noisier depth map and ”shaded areas” in the depthmap (when one of the two cameras is occluded and thus stereo matching cannot be performed).

## 3.5 Experimental evaluation

### 3.5.1 Limitations of the 3D pose estimator

The main weakness of the proposed 3D pose estimator is that it relies on accurate depth information in order to obtain correct 3D positions from 2D locations in the image. The RealSense camera being used provides accurate depthmaps up to 4 meters. Making use of the available hardware, the pose estimator was tested in a qualitative way with in-the-wild data within the operating range of the camera, however experiments with the camera equipped on the hexarotor did not give the expected results, since depth was systematically overestimated whenever objects were further away than 4 meters. The

the application programmer to transparently read undistorted and aligned RGB-D images, computing depth maps from disparity maps and aligning them in software. Since the frame acquisition speed can go up to 60 FPS at 848x480 resolution, the depth quality is adequate for slow-moving objects problem may be solved either with a 3D stereo camera rated for larger distances or by using a more powerful computer to and deploy one of the existing 3D HPE algorithms. While subpar for real-life applications because of hardware constraints, the setup was anyway tested by using a mix of synthetic data to evaluate the quality of the proposed QoE heuristic.

### 3.5.2 Evaluation of the QoE heuristics

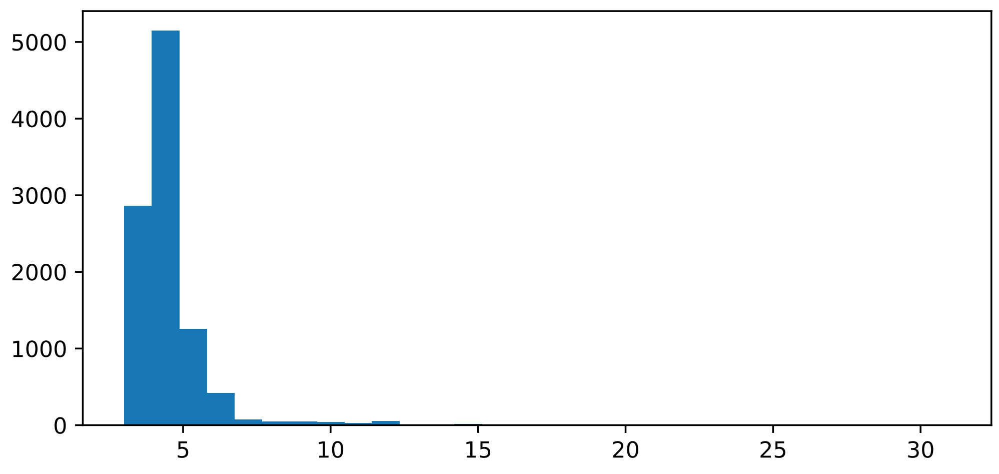
In order to quantitatively assess the performance of the QoE heuristic, a synthetic RGBD dataset, annotated with ground-truth 3D joint positions, was generated using Unity and the Synthetic Humans package (see Figure 4.1 for examples of images in the dataset). The pose estimator was run on the synthetic dataset, obtaining predictions for the 3D skeletons. Then, the Pearson correlation coefficient between the QoE heuristics and the MPJPE (mean per-joint position error) was computed.

Correlation between the complemented QoE heuristic (so the "uncertainty" of the estimation) was found to be $\rho \approx 0.64$ on a sample of 5000 synthetic generated frames. While ideally the two quantities should have a correlation coefficient closer to 1, it is nevertheless possible to conclude that the QoE heuristic proposed has a significant statistical relationship with the actual estimation error.

### 3.5.3 Run-time analysis

The whole 2D pose estimator + 3D augmentation + uncertainty estimator has the capability to guarantee at least 30 FPS while running on the Jetson board in the most power-hungry configuration (MAXN, requiring up to 25 W of power), and 17 FPS in the most power-efficient configuration (MAXQ). An experiment was performed to assess the performance of the pose estimator and guarantee it was not posing any bottleneck on computation time of the system as a whole. An initial design of the uncertainty estimation included CUDA-accelerated code, but upon testing, it was found that using CPU-only code for elaboration of the heatmaps is much more efficient. The next design iteration was then tested with the results visible in Figure 3.8. The runtime of the uncertainty estimator was within 5 milliseconds in almost 80% of the test runs. The highest recorded runtime was of 31 ms. The extraction of connected components is the most resource-intensive computation within the heuristic computation, so variability in the run time of the uncertainty is estimator may be explained with the intrinsical variability of the heatmaps, which may require more or less time for connected-components analysis.

Figure 3.8: Histogram of the running time needed for the $QoE$ computation (milliseconds) in 10000 runs. The first three columns account for over 90% of the trials.



# 4 | Exploration policy learning

## 4.1 Limitations of the uncertainty estimation heuristic

Since the heuristic computation happens in an online fashion and is entirely based on the output of the pose estimator (which is obviously dependent on the camera output), the uncertainty heuristic provides information about the *current state* of the estimation, but provides no directly usable information to perform *predictions*, or, in other words, it is not immediately clear how to use the heuristic measure to perform *trajectory planning.*

Given the safety constraints entailed by a collaborative setting, it might be advisable to separate the environment exploration from the actual handover, separating the robot operation into two different states, *exploration* and *handover*[1]. During the exploration phase, the robot would keep a safe distance from the human while trying to find a pose that guarantees a low uncertainty in the human pose estimation. Once the robot has found such a pose, the handover routine would take over, aiming at reaching the handover location (a point in front of the human operator) quickly and safely, for instance using pose-based visual servoing, or maybe even assuming the output of the pose estimator to be entirely correct and planning a straight-line path to the handover location.

### 4.1.1 Using the uncertainty heuristic for exploration

The uncertainty heuristic described in the previous chapter may have different applications, varying in complexity and flexibility. One immediately applicable idea, assuming the human operator will be inside a fixed-size workspace whose position is known, is to have the robot follow a circular trajectory around the human workspace, while measuring the uncertainty in each position. Once a sufficiently satisfying uncertainty is achieved, the handover process may begin.

On the other hand, it could be more useful to *learn* an exploration policy that aims

[1]Such separation might be formalized as a very simple State Machine or Behavior Tree, in which the state transition is triggered when a sufficiently low-uncertainty pose has been reached. Behavior Trees have been proposed as a general paradigm for designing autonomous UAV behaviors, for instance in [59]. It must be noted that the proposed ”exploration behavior” is in fact a very complex task that requires substantial computational power.

at minimizing the uncertainty based on prior knowledge of the setting: for instance, auto-occlusions will negatively influence the depth estimate of the shoulders when the camera is looking at the human from the side. This will result in a higher uncertainty for the occluded shoulder, since its predicted depth will be the same as the neck and shoulder keypoints: this will make the three keypoints appear to have almost coincident positions, resulting in a higher uncertainty value of the bone length uncertainty component. A smart exploration policy may be able to detect this situation and translate it into an appropriate action, such as impressing a lateral velocity to the drone (*i.e.* moving to the front or to the back of the person), to make sure to get rid of the auto-occlusion.

## 4.2 State-action predictor

The first idea that was explored was training a predictor that, when provided with information about the current frame (*e.g.* any combination of: frame features extracted by a CNN backbone, extracted keypoints from the pose estimator, per-joint QoE heuristic), could be able to predict the QoE heuristic in other positions around the current one. Ideally, the output of such a predictor would be the action to undertake to improve the QoE heuristic.

Several experiments were conducted to validate this idea, by training classifiers with different architectures on the following task: given an image and 8 possible choices of movement (up, up-left, left, down-left, down *etc.*), predict the direction in which the uncertainty will decrease.

To achieve this, a synthetic multi-view dataset comprised of 1024 scenes was created using the Unity game engine. For each scene, a human was placed in the center of the scene and 9 pictures were saved, taken at the same time with 9 virtual cameras, spatially arranged in a 3x3 array (one above the center camera, one above-left, one to the left, one below and to the left, *etc.*), all of them facing the human.

Every picture was labeled with its QoE heuristic (computed offline), and the image in the center of the array was given as input, while the remaining 8 pictures were used as "ground truth" values to be predicted.

The following architectures were tested (in the first column, netowrk architecture is represented as a Python list: every element represents a layer of neurons, and the number itself represents the number of neurons in that layer):

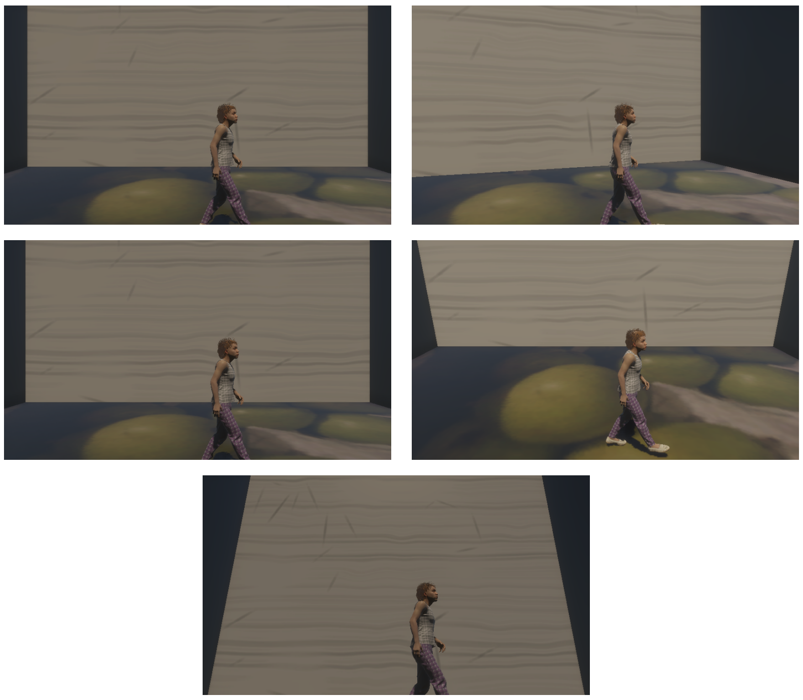


Figure 4.1: Examples of the multiview dataset generated with Unity. The background textures are randomized.

| Network architecture | Features being considered | Training error | Generalization error |
|---|---|---|---|
| [584, 128, 16, 8] | VGG16[52] features + joints + QoE | 0.13 | 0.82 |
| [72, 64, 8] | joints + QoE | 0.07 | 0.86 |
| [72, 32, 8] | joints + QoE | 0.10 | 0.76 |
| [72, 32, 32, 8] | joints + QoE | 0.18 | 0.88 |

It should be noted that the best results for generalization error (computed on a separate test set of 256 scenes) were all quite close to 0.875 which is the average error rate of a naive predictor (classes were almost balanced in the synthetic dataset). This means that the learning process has not achieved any meaningful result. The results of these experiments suggest that information from isolated frames may not be enough to learn a state-action predictor, so context information gathered from surrounding positions is needed to translate the current state of the pose estimator and uncertainty quality into information about the action needed to reduce uncertainty.

Such information might be provided by replacing the MLP with an RNN, in which the agent follows predefined trajectories. The predefined trajectories could be used to gather sequential information for training of the RNN. This idea, however, was not explored further: the most evident flaw is that, while the sequential information will certainly prove useful in removing ambiguities among similar states and granting learnability of a state-action pair, sequences are manually decided a priori, without taking into account the output of the state-action predictor.

Furthermore, a state-action model is inherently greedy, and does not explicitly take into account the difference between *reward* and *value*. In fact, simply estimating the best possible action to undertake in a given state would easily get the robot stuck in a local minimum, preventing it from reaching a pose in which the estimation uncertainty is low enough to perform a handover.

## 4.3 Reinforcement Learning

Reinforcement Learning algorithms explicitly model the problem as the choice of a *policy*, which contributes to solving the aforementioned problems with state-action networks (the most important of which being the "near-sightedness" of a policy that does not take into account past *nor* future steps): in fact, while a RNN solution might conceivably be able to use information about the visited states to integrate information about the current state, it would still lack the structure to avoid "dead-ends", *i.e.* actions that hinder the ability of the agent to reach its goal in the next moves. This last concept is tackled in reinforcement learning with *value estimation*, that is, by training a predictor that, when given a state as input, outputs the *expected return*[2] for the rest of the episode.

RL provides a natural and intuitive representation of the problem being considered. The pose estimator neural network provides a low-dimensional representation of the image data generated by the virtual camera in the 3D simulation, and thus helps speeding up the learning process.

### 4.3.1 Continuous state and action spaces

The RL model that was initially chosen for the task entails a continuous action and state space, closely resembling the states and actions available to a real UAV. In particular, the hexacopter being used at the University of Twente can be controlled by means of a high-level controller impressing velocity commands in Euclidean space [3]: such commands translate directly into velocity commands from a continuous action space that could be the output of a RL model such as the one being considered in the following.

The RL task was specified as follows:

[2]Return may be loosely defined as the sum of future rewards, giving more importance to the steps closest to the current one than the steps far in the future. See Appendix A for details.

[3]The high level controller takes care of translating velocity commands into torques for the propellers, so that the three-dimensional velocity vector is impressed to the center of mass of the drone.

**State:** filtered positions of the joints (current state $\mathbf{x}$ of the Kalman filter, together with uncertainties for all joints $diag(P_{post})$ and current position of the robot $\mathbf{x_R} \in \mathbb{R}^3$

**Action:** $\mathbf{v} \in \mathbb{R}^3$ linear velocities to be impressed to the robot, in the range (-1, +1) [m/s]. [4]

**Reward:** The reward was chosen to be dense and dynamic and initially specified as the sum of the uncertainties of all the keypoints being considered (trace of the covariance matrix $\mathbf{P_{post}}$ of the Kalman filter). This is however far from ideal, since for each episode the keypoints have initially very bad (random) estimates and the uncertainty will decrease dramatically regardless of which action is undertaken. This is why in the initial frames of each episode, reward is almost completely dependent on the distance from a point in front of the human.

The interpolation between the two is controlled by one parameter $\beta \in (1, \infty)$ in the following way:

$$R = -(\alpha R_1 + (1 - \alpha) R_2)$$

$$\alpha = (1 - \frac{1}{\beta}) + \frac{n\_steps}{\beta max\_n\_steps}$$

where $R_1$ is the reward component coming from the uncertainty component of the Kalman Filter and $R_2$ is instead the distance from one reference keypoint of the human skeleton (*e.g.* the shoulder). It follows from the above equations that $\alpha$ takes values in $(1 - \frac{1}{\beta}, 1)$, so tuning the parameter $\beta$ allows for different tradeoffs between the two components of the reward.

A very negative reward is given in case the robot moves too close to the human or out of the workspace, also causing truncation of the episode.

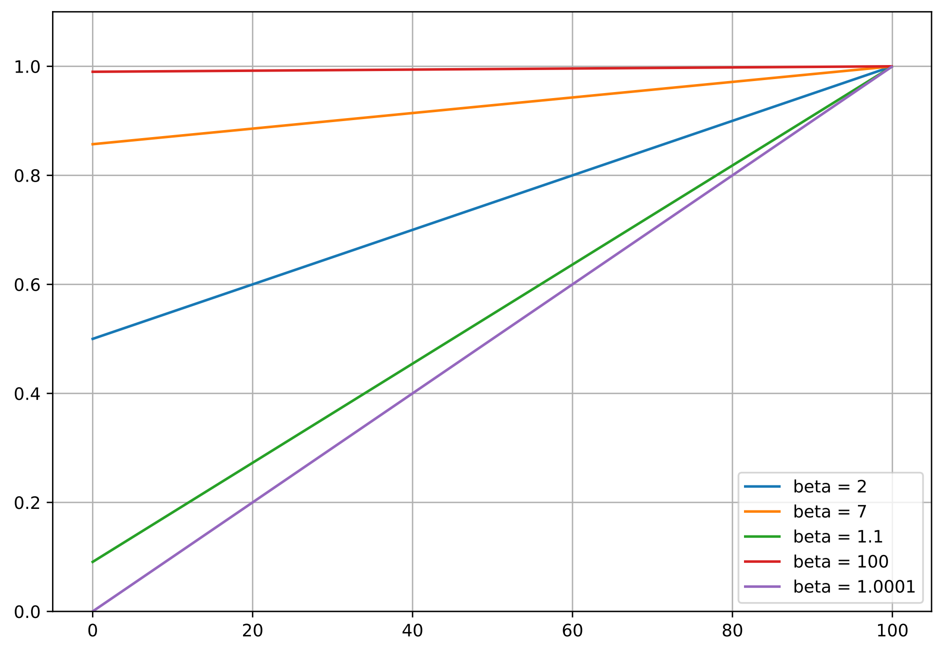
### 4.3.2 System setup

The processing pipeline for the exploration policy relies on three different processes:

1. 3D simulation, capable of generating photorealistic RGB-D renders of a 3D model of a human from arbitrary camera poses, and of exporting the camera position.
2. Pose estimation: using the RGB-D and camera pose data generated by the 3D simulation, estimate the 3D pose of the human in a fixed reference frame. The Kalman filtering of the pose estimator outputs takes places in this process.
3. Reinforcement learning: a policy network is trained on the intermediate representation provided by the (filtered) pose estimator output. The RL algorithm

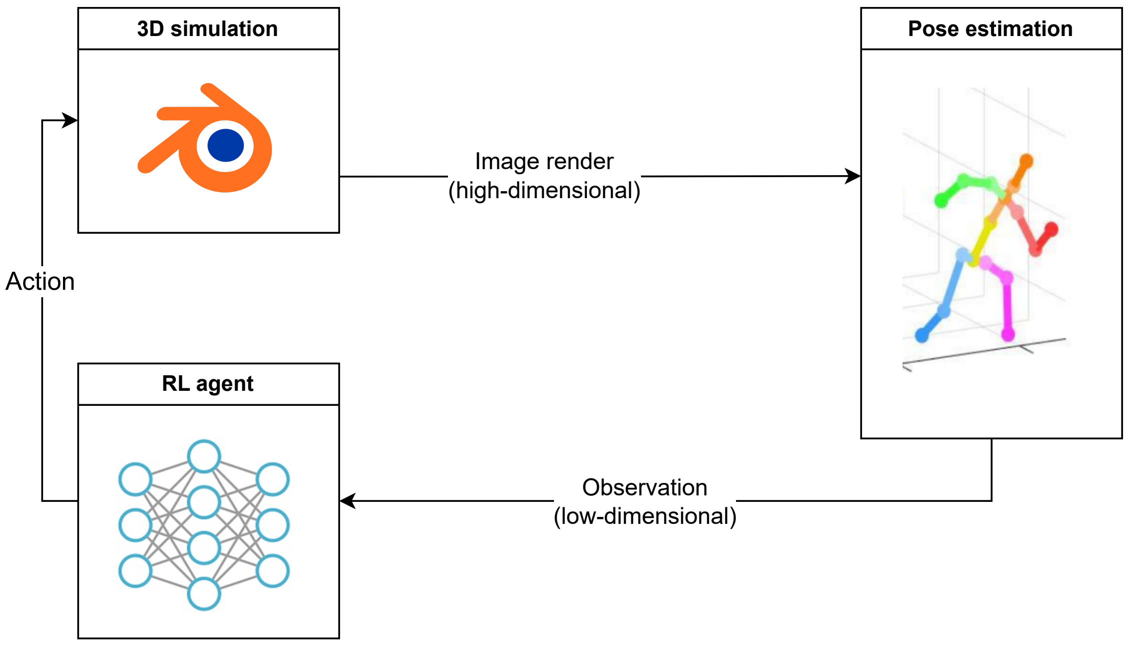
[4]For the sake of simplicity, angular velocities are not considered in this setup: a constraint on the robot's yaw was added in the simulation to make sure the human stays in the field of view of the camera. In a real-life setting, the angular velocity $\omega_z$, would be the fourth action output.

Figure 4.2: The profile of $\alpha$ for different values of $\beta$, assuming a maximum episode length of 100 steps. On the x axis: the step number; on the y axis: the value of $\alpha$. Note how the parameter choice gives more or less influence to the ”distance from target” reward component.



will interact with the 3D simulation (environment) by changing the camera position (*action*) and reading the pose estimator output of the newly rendered frame (*observation*).

Figure 4.3: High level schematic of the three processes involved in the reinforcement learning pipeline. The arrow direction indicates the order of execution of the processes, not the direction of communication,, which is bidirectional between the RL agent and each of the other two modules.



**Graphical 3D simulation** The 3D graphics program Blender was used for simulating the camera output of a drone-mounted camera. A 3D model of a human was created using the MakeHuman plugin. To improve generalization, the position, body pose and appearance of human models were randomized for each run.

**Reinforcement Learning agent** The reinforcement learning agent uses the Soft Actor Critic algorithm [60]. To speed up convergence, the experience buffer was filled with episodes where the actions were pre-defined to move closer to the human. The implementation of SAC was obtained from the framework skrl [61], wrapping the pose estimation + 3D simulation pipeline in a custom Gymnasium[5] environment.

[5]Gymnasium is a fork of OpenAI Gym [62], a Python package that allows to easily define Reinforcement Learning environments and tasks. Gym is regarded as the de-facto standard for Reinforcement Learning in industry and academia, and all the most common libraries implementing RL algorithms support the Gym interface for specification of environments.

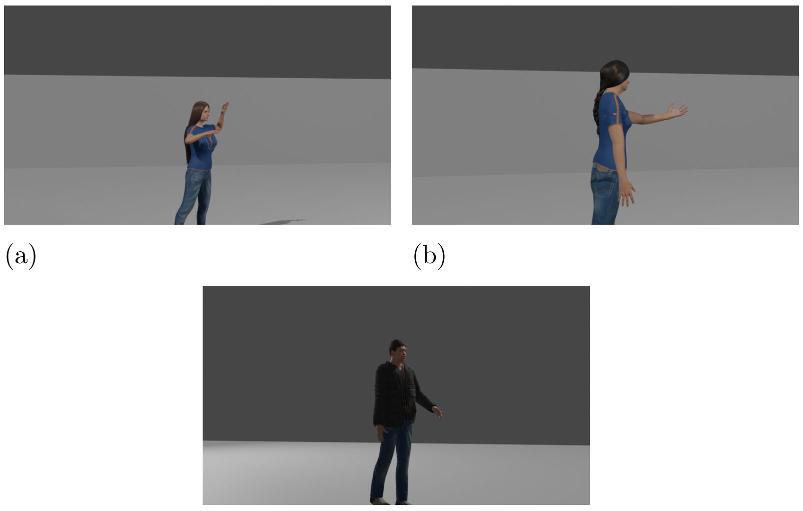

(a) (b) (c)

Figure 4.4: Examples of frames from the Blender 3D simulation wiht synthetic humans

**Inter-process communication** Execution of the three processes needs to be sequential, so it is necessary to synchronize them using appropriate IPC techniques. Both the pose estimator and the RL agent run on the Jetson TX2. Synchronization and message passing between these two happens using Unix named (FIFO) pipes. On the other hand, communication between the RL agent and Blender is achieved with TCP sockets and a custom protocol based on simple byte-length messages (e.g. the byte corresponding to the ASCII representation of the letter `N` is sent to signify a `New` episode has started), together with a 24-byte array representing the action the agent should undertake. Similarly, RGB and Depth images are passed to the pose estimator by means of a simple HTTP server. This architecture was in part mandated by the fact that the Blender renders needed to take place on a different computer. [6]

The Gymnasium environment acts as a master process in that it manages synchronization and file copy from the remote HTTP server (the render server) and sends appropriate signals to the pose estimator to indicate that the copy is terminated and processing can begin.

### 4.3.3 Discretized states and actions

The system described above had very poor performance, mostly because the time required for rendering and elaborating one frame (*i.e.* the time required for evaluating one single action of the agent) was averaging 1 second and sometimes exceeding 2 seconds. Moreover, given the continuous action space, the sample complexity of the algorithms was very high, and the multi-process architecture proved to be very fragile, since frequent crashes prevented the training from running long enough to converge

[6]Even if the performance of the Jetson allowed for rendering at the same time as pose estimation and RL agent training, this would bring no benefit to the overall system evaluation, because in real life conditions data is provided by a camera stream.

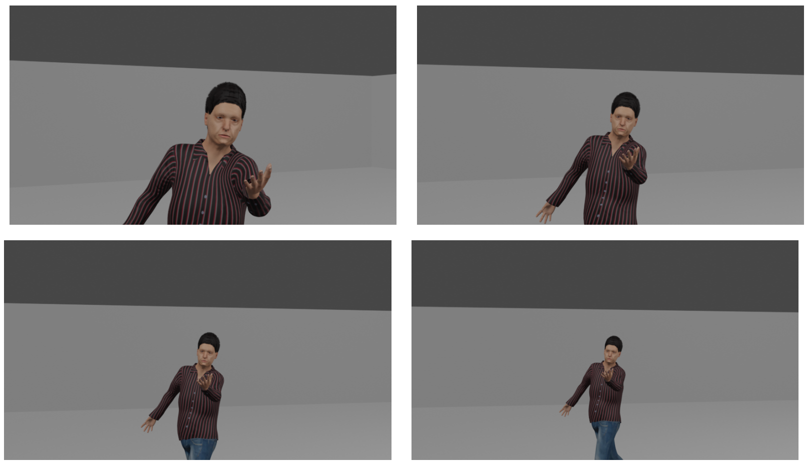


Figure 4.5: Discretization of the Blender 3D environment: in this case the x coordinate is increasing. Note how the camera is always facing the human model.

(even though after tens of thousand of training steps, the training did not yet converge to any reasonable policy).

For all these reasons, the experiment was then restructured and drastically simplified, keeping in mind the original target: evaluating whether the QoE heuristics were sufficient or even useful for learning a meaningful exploration policy for a UAV.

### 4.3.4 Environments

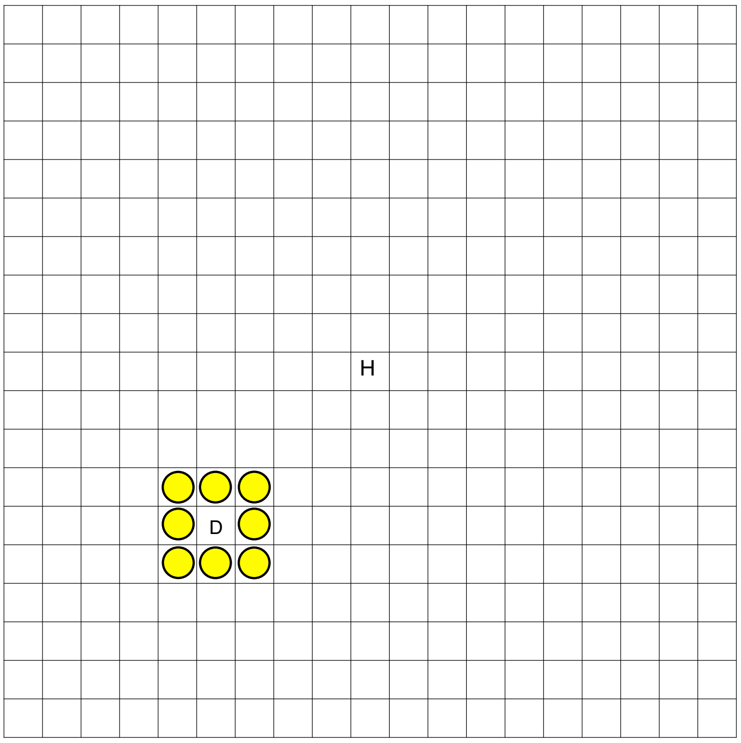
In order to increase the sample efficiency of the algorithm, allowing the learning to last for hundreds of thousands of episodes, a fixed set of environments was created in Blender, selecting human models from a set of 60 (40 for training, 20 for testing) randomized synthetic humans generated using MakeHuman. Each human had a pose chosen from a set of 8 poses (6 for training, 8 for testing, so test models included two poses unseen at training time). Humans were always placed in the $(0, 0)$ coordinate of the render space, but their rotation was randomized.

Each environment was then divided into a 19x19 grid, composed of 361 squares having a side of 1 meter, with $x$ and $y$ values spanning from -9m to +9m. After choosing a fixed altitude value, a view of the human from each of the grid squares was rendered and saved to disk, and then fed into the pose estimator. For each of the frames, the QoE heuristic was computed.

Finally, to discretize the action space of the agent, it was assumed that the robot could move to the 8-connected cells around it for each step in each episode.

These tricks turn every instance of the original, very complex enviroment into a gridworld-like environment, in which the reward of an action depends exclusively on the new state, without any dependency on previous states. If the environment was limited to a single instance (always the same human model, in the same pose, with the same orientation in space) learning an optimal policy in such an environment could be very

Figure 4.6: In a 19x19 grid, the drone (letter D) appears in a random position for each episode and can move to the 8 adjacent cells at each step (the ones with a yellow circle). The human (letter H) instead is always in a fixed position in the center of the grid.



easy even with classical Reinforcement Learning algorithms such as Q-learning [63], however it must be noted that the policy we would like to learn must be independent on the instance being considered.

### 4.3.5 Environment design

In the Gymnasium environment constructed for the experiments, the state representation is a vector

$$\mathbf{s} = [p_x, p_y, q_0, \cdots, q_{17}]$$

where the first two components are integers representing the $x$ and $y$ coordinates of the agent, while the $q$ components are the QoE estimation values for each joint in the skeleton.

For reward and episode termination, a weighted sum of the heuristic uncertainty for the upper body was used:

$$q_w = \mathbf{w}^\top \mathbf{q} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 1 \end{pmatrix}^\top \begin{pmatrix} q_{nose} \\ q_{l.eye} \\ q_{r.eye} \\ q_{l.shoulder} \\ q_{r.shoulder} \\ q_{l.elbow} \\ q_{r.elbow} \\ q_{l.wrist} \\ q_{r.wrist} \\ q_{neck} \end{pmatrix}$$

The weight vector $\mathbf{w}$ was chosen considering that a good handover position will need to be in front of the human, thus showing their face features. Giving a high weight to face keypoints is a trick for forcing the policy to move away from poses in which the face keypoints are not detected (thus their uncertainty is 1). Ears were not considered because highly susceptible to depth errors which result in 3D localization errors. Note that $||\mathbf{w}||_1 = \dim \mathbf{q}$ so the weighted sum has the same variation space as the naive sum of all uncertainty parameters for the upper body.

Episodes are considered successfully concluded whenever the agent reaches a position where $q_w$ is "small enough", *i.e.* when it is below the threshold value of 1.1, empirically determined as the lowest threshold below which the majority of training episodes began to fail. As desired, this formulation does not allow for any of the face keypoints to go undetected in the final position, since in that case the keypoint uncertainty would be calculated as 1.

### 4.3.6 Learning algorithm and network architecture

The learning algorithm chosen is Proximal Policy Optimization [64] (PPO). PPO is a policy gradient method introduced in Section A.2. PPO uses two neural networks to estimate the policy (*Actor network*) and the value (*Critic network*) function. For both neural networks, a recurrent architecture was chosen, and the implementation was provided by the Stable Baselines 3 "Recurrent PPO" package [65]. The recurrent architecture was considered necessary in order to incorporate temporal information from previous steps in the state vector: in fact, given the form of the state vector, the recurrent network may help in (1) modeling trajectories that do not repeatedly visit the same cells and (2) help in differentiating states where QoE is poor because of occlusions from states where the same occurs due to an unfavorable orientation of the human.

Both the actor and critic network were chosen to have the same internal structure, a 2-layer LSTM network having a hidden layer size of 256. Assuming the network input to be already in homogenous form (*i.e.* after having added a "1" component to the $\mathbf{x}$ vector to include the bias), the network architecture is described by the following equations[7]

$$\mathbf{l_{out}} = LSTM(\mathbf{s})$$

$$\mathbf{z} = \tanh(\mathbf{Z}\mathbf{l_{out}})$$

Where

- $\mathbf{s} \in \mathbb{R}^{20}$ is the observable state representation
- $\mathbf{l_{out}} \in \mathbb{R}^{256}$ is the LSTM output of a standard 2-layer LSTM network having hidden size 256
- $\mathbf{Z} \in \mathbb{R}^{128\times 256}$ is learnable

while the two networks have the same architecture, the weights are not shared, and their outputs are computed in a different way.

$$y_{value} = \mathbf{W_{value}}\mathbf{z}$$

$$y_{action} \sim \text{softmax}(\mathbf{W_{action}}\mathbf{z})$$

where

- $\mathbf{W_{value}} \in \mathbb{R}^{1\times 128}$ is learnable
- $\mathbf{W_{action}} \in \mathbb{R}^{8\times 128}$ is learnable
- $y_{action}$ is sampled according to the distribution given as output by the softmax function

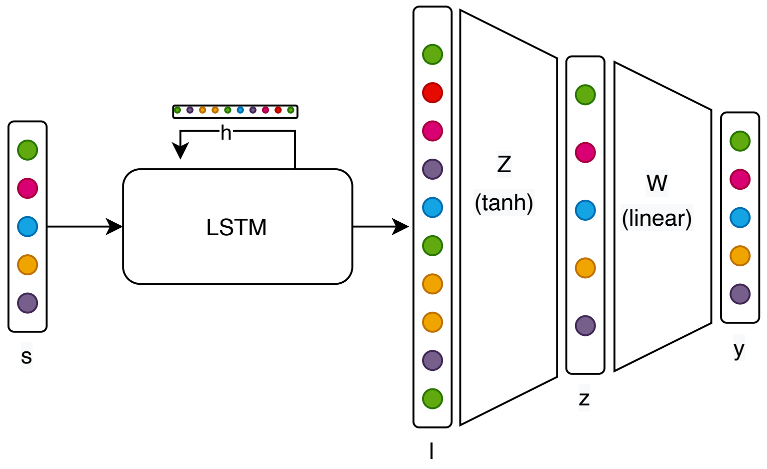
For the actor network, the scalar output is an integer which represents the index of the next action to undertake (for instance, 0 means "moving to the cell in front and to the left", 1 means "moving to the cell in front" and so on). For the critic network instead, the output represents the approximated value of the state (which is then used by PPO to estimate the advantage and optimize the actor network).

## 4.4 Training details

Training was performed using the curriculum learning technique [66], gradually lowering the QoE threshold for an episode to be considered successful, starting from 1.6 and

[7]Please note that bias is disregarded in the vector and matrix dimension list, but it was indeed present in the network.

Figure 4.7: Schematic representation of the basic Network architecture used for learning an exploration policy. The LSTM internal structure is not explicitly drawn, as it is simply the standard implementation of a 2-layer LSTM. The hidden state vector **h** is meant to show the recurrent computation that allows to keep track of a sequence of inputs.



lowering it by 0.05 for each continuation run. Performance was evaluated by qualitative inspection of the generated policy after each continuation rerun, and good performance was assessed after 12 reruns. For each rerun, the actor performed about 125 000 actions (allowing episode termination after the fixed number of steps was reached), so the total number of training steps was 1.5 million.

During training, in order to encourage exploration at the expense of exploitation (cfr. [67, § 1.1]), a gradually decreasing entropy bonus (suggested in the original PPO paper), was added to the loss function of PPO, with an initial coefficient of 0.3. The entropy bonus was removed at test time.

The optimizer of choice was Adam and the learning rate was fixed at $3 \times 10^{-4}$

## 4.5 Reward design

Three different experiments were performed with regard to the reward form, in order to be able to appreciate the contribution provided by the QoE uncertainty when used in the setting of exploration policy learning.

### 4.5.1 Binary reward

This is a sparse reward

- Episode terminated: +1
- Crashed into the human: -1
- None of the above: 0

It should be noted that the sparse reward provides little information to the value network, so it is expected that its performance will be worse.

### 4.5.2 Binary reward with path length penalty

- Episode terminated: +1
- Crashed into the human: -1
- None of the above: -0.1

The penalty is introduced to encourage the agent to follow short paths.

### 4.5.3 Variable reward based on QoE value

- Episode terminated: +10
- Crashed into the human: -10
- None of the above: QoE value of the current state

## 4.6 Evaluation and results

### 4.6.1 Quantitative results

| Model (best per class) | Training steps | Episode length (avg ± std dev) | Success rate | Crash rate |
|---|---|---|---|---|
| Binary reward | 625k | 35.49 ± 26.73 | 0.817 | 0.02 |
| Binary reward with path length penalty | 1.25M | **10.62 ± 19.17** | 0.865 | 0.04 |
| Variable reward based on QoE value | 1M | 12.13 ± 17.73 | **0.925** | **0.003** |

Note that the best model was selected based on test data, so models trained for over 1.5M steps may not necessarily be the best ones.

### 4.6.2 Qualitative analysis and failure modes

In the following, a *righteous failure* is defined as a situation in which the agent keeps exploring without ever succeeding[8] because, even after reaching a pose minimizing the estimation uncertainty, the uncertainty in that pose is above the specified threshold (*i.e.* $q_w > 1.1$).

**Binary reward** For most of the training runs with the reward calculated as in Section 4.5.1, the agent tended to circle around the human, always in the same direction. This not surprising, since reducing the number of possible actions simplifies the task. This, however, resulted in longer paths on average to reach the low-uncertainty configurations. Upon righteous failures, the agent keeps on circling around the human until the timeout. A smaller number of training runs exhibited a different behavior, following a semicircular path from the front to the back of the human, and thus always keeping to the left or right side, without ever exploring the other side.

**Binary reward with path length penalty** For training runs using the reward as calculated in Section 4.5.2, the behavior is less predictable, even though the oscillatory behavior described above was also present in some cases. The path length is significantly smaller, but it may be noticed how sometimes value estimation is very poor, for instance when the actor chooses to approach the human directly from the side, without first moving to the front, where the QoE uncertainty is lower. Upon righteous failures, the behavior is arguably more reasonable than in the previous case: the agent simply moves backward and forward between the same few positions, without ever exploring more.

**Variable reward based on QoE value** Finally, for training runs using the reward as calculated in Section 4.5.3, righteous failure behavior was very similar to the previous case. The preference for this last kind of reward is due to better quantitative results and faster training time. It may be argued that the QoE heuristic effectively provides reward shaping for the environment being considered.

**Other considerations** One obvious limitation of the training setup exposed so far is that the human is always in cell (0,0) in the environment, so while the policy network is forced to learn a way to turn around the human, it will not learn how to approach the human, and avoid crashes, but it will rather move ”close enough” to cell (0,0) and then avoid stepping into it. More experiments are needed to assess whether the agent would be able to perform well in this setting, too[9], however it should be noted that in

[8]eventually, the episode is terminated by a timeout after 100 steps

[9]in such a setting, the policy learning would certainly benefit from the entropy bonus being higher than 0, as exploration is always preliminarly needed to localize the human. Moreover, in this setting

many industrial settings humans are constrained to work within confined spaces, which could serve as "(0,0) cell" for the UAV exploration.

it would not be possible to precompute the rotation to keep facing the human, but yaw rotation would also need to be available as an action to the agent.

# 5 | Conclusions and future work

In this thesis, design and implementation of one of the building blocks for payload-carrying collaborative UAVs was proposed and discussed. This building block takes care of a ”hovering routine”, that takes place once the UAV already picked the object which the human operator needs, has localized the operator and is exploring the environment, looking for an apt position to start the handover.

In Chapter 3, it was shown how intuitive and computationally lightweight heuristics may provide sufficiently reliable information about whether a human pose estimator is in a failure mode, roughly quantifying the probability of a prediction being wrong. This information may be provided individually for each joint, and is based both on constraints posed by the structure of the human body and on intrinsic and intuitive features of 3D human pose estimators. The heuristics, use information coming directly from the pose estimation model, through the heatmaps, and also some distance priors, to further constrain the acceptable poses. A time consistency prior based on a Kalman filter, aiming to detect inconsistent measurements by the pose estimator, was also proposed but not exploited.

Furthermore, in Chapter 4, the heuristic proposed was shown to provide enough information for learning an exploration policy, when provided as a reward value in a very simple and controlled Reinforcement Learning environment.

On the one hand, the learned policy could in principle be used by a real drone as a kind of meta-controller, since a high level controller translating the discrete states into trajectories would be needed: the pose detections and associated QoE values could be filtered through the Kalman Filter described in Chapter 3, and the reference QoE value used for computing the reward could be either the Kalman filter uncertainty or an average filtered value using a moving window average.

On the other hand, though, mostly due to the limitations of the naive 2D-to-3D pose lifting model based on raw depth data, the proposed system is mostly of theoretical interest. The software could not be run successfully on the real setup, as experiments on recorded camera data from a flying drone clearly highlighted, since noisy depth measurements interfered with the 3D lifting of the 2D poses, hindering

As future works, it would be the next logical step to make the overall system deployable in real applications with real-time performance. In particular, two main research directions will be investigated: the use of 3D pose estimators and sim2real techniques.

**Training of a 3D HPE** To overcome the limitations of the 2D pose estimator, a 2D-to-3D pose lifter network could be trained with additional depth input (ie data from a real depth sensor could be used to augment the training data). The richer input should help in reducing ambiguities in the training data, even if the depth measure is not very accurate, and thus help in driving down the number of parameters and custom decoding architectures.

Additionally, in case an encoder-decoder architecture similar to [29] were to be used, the proposed QoE heuristic might be used as part of the reconstruction loss, during inference, to favor 3D reconstructions that have a higher QoE value.

**Learning a real controller and sim2real transfer** In presence of a high-level controller for the drone, able to control the drone in velocity, it is conceivable to conclude the experiment mentioned in Section 4.3.1, or otherwise to directly learn a high-level controller using Reinforcement Learning. The ability to control torque speeds of the propellers using based on camera data would constitute an advantage for the simplicity of the system, even though it would not guarantee that safety constraints are obeyed at all times. Moreover, learned controllers may sometimes feature instabilities that result in catastrophic failures (free fall) for aerial vehicles. A less drastic measure for sim2real transfer would be to use the MPC detailed in [68] and simply treat the RL output as one of the many inputs to the high-level controller.

In conclusion, trying to reach a synthesis between learning-based, fail-prone algorithms and the delicate equilibrium and resource constraints of UAVs has been challenging, and more experiments are definitely needed to assess the validity of the proposed algorithm. Anyhow, the QoE heuristic tries to keep together the "best of both worlds", by giving an evaluation measure of a learning-based HPE, with the stability, speed and determinism of classical image processing.

Machine learning is today (and will be increasingly more in the future) a fundamental tool for building collaborative robots; at the same time, managing the complexity and uncertainty entailed by deep learning models in computationally efficient and interpretable ways is an interesting topic that could help in bridging the gap between complex models and their deployment on real robots, and the contribution exposed so far moves a little step in this direction.

# A | Reinforcement learning primer

In this small chapter, tools that have been investigated and subsequently used by the candidate in the experimental parts of his work will be introduced, as he wishes to provide references and a short primer to the reader who may not be familiar with the topic of reinforcement learning, not unlike him at the beginning of the thesis work.

Reinforcement learning (RL) is a branch of machine learning concerned with learning *policies* through exploration of an *environment* performed by an agent. As with many branches of Machine Learning, RL makes use of concepts ispired by studies on human and animal learning, namely the *reward* process involved in operant conditioning which was extensively studied in behavioral psychology and animal behavior [69]. In RL, an agent is able to explore its environment (defined as a set of *states*) by means of a fixed set of *actions*. Every action, when taken in in a given state, gives rise to a reward, which could be positive or negative, depending on whether the agent is acting towards achieving its goal or not. The exploration of the environment occurs in *episodes* featuring a finite number of timesteps. At each timestep, the agent chooses an action which may or may not change the state it is in.

Broadly speaking, the goal of RL is learning a policy $\pi(a|s;\theta)$, parametric in parameters $\theta$, which assigns an optimal probability of undertaking action $a$ when the agent is in state $s$, so that the agent maximizes its expected *return* starting from the current timestep $T$.

**Return** Return is defined as the *discounted sum of rewards*,

$$R = \sum_{i=T}^{\infty} \gamma^i r_i$$

, where $r_i$ is the reward obtained in timestep $i$ and $\gamma \in ]0,1]$ is a discount factor used to give more weight to the reward of the timesteps closer to $T$, *i.e.* in the near future.

**Value** The value of a state $s$ is the expected value of the return, considering a sequence of actions $\tau$ sampled according to policy $\pi$ and starting in state $s$

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau|s)]$$

**Q-value** The Q-value of a state-action pair $(s, a)$ is the expected value of the return, considering a sequence of actions $\tau$ sampled according to policy $\pi$ and starting in state $s$ and choosing $a$ as first action

$$Q_\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[R(\tau | s, a)]$$

**Advantage** The advantage of a state-action pair $(s, a)$ is the difference between the value and the Q-value, that is, how much better the return of an episode would be, on average, if choosing $a$ as a first action instead of any other action

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

The RL paradigm applies very naturally to intelligent robotics: a robot exploring its environment is an *agent*, and its *actions* depend on its actuators, which allow it to move around in the environment or to modify it. Reinforcement learning, especially when coupled with deep learning and this called *deep reinforcement learning*, has been recently applied to robotics for learning high-level controllers ([70] offers an overview of recent applications of deep reinforcement learning to mobile robotics).

## A.1 Deep Reinforcement Learning

Classical reinforcement learning offers many clever and clean algorithms, which however do not scale to problems with large state spaces and quickly become computationally infeasible, usually because of memory constraints.

Instead of learning exact, closed-form policies that need to keep track of every combination of states an actions, a neural network could be used to approximate the policy, ideally having as input a state representation, and giving as output the best action to undertake, or, more often, by using a neural network to estimate the future return of a certain $(state, action)$ pair. This approximation makes RL algorithms scalable to much bigger state spaces, as demonstrated for the first time with Deep Q-Learning [71], which learns a policy for playing Atari videogames taking as input simply the raw frames from the videogame screen (since the Atari 2600 has 128 colors and a 192x160 resolution, the resulting state space, if enumerated completely, would be impossible to fit in memory, being of size $128^{192 \times 160} \gg 2^{100000}$[1]). The neural network thus allows for meaningful dimensionality reduction, allowing the algorithm to operate efficiently with small though dense (and seldom interpretable) vector representations of the state.

[1]The estimated number of atoms in the observable universe is smaller than this number by several thousands of orders of magnitude. Note that the actual number of different frames observable in an Atari game is much smaller, and the overwhelming majority of the states in this theoretical state space would be made of meaningless random images.

## A.2 Proximal Policy Optimization

Proximal Policy Optimization is a Deep Reinforcement Learning algorithms introduced in [64] that tries to optimize policy learning by ensuring that every policy update (weight update step) does not result in a policy too different from the previous one, to improve robustness to noisy training data (rare training samples may not excessively influence the policy) and sample efficiency ("destructive" updates to the policy are less likely to occur).

PPO is an *on-policy* algorithm (the policy used for exploration of the environment is the one being learned), and an tDefactor-critic algorithm, since it uses two neural networks, one for approximating the policy, and the other to approximate the value of the state and thus computing advantage values.

The aim of PPO is reached by maximizing a surrogate objective function, based on the *clipped* ratio between the probability of the current action-state pair before and after the policy update.

$$\rho(\theta; a, s) = \frac{\pi_\theta(a|s)}{\pi_{\theta old}(a|s)}$$

$$L^{CLIP}(\theta) = \mathbb{E}[\min(\rho(\theta; a, s)A(a, s), \mathrm{CLIP}(\rho(\theta; a, s); \epsilon)A(a, s))]$$

where

$$\mathrm{CLIP}(x; \epsilon) = \begin{cases} 1+\epsilon & \text{if } x > 1+\epsilon \\ 1-\epsilon & \text{if } x < 1-\epsilon \\ x & \text{otherwise} \end{cases}$$

$\epsilon$ is a hyperparameter used to choose the clipping range of the ratio between the old and new likelihoods of action $a$ in state $s$. Thanks to the clipping, if $a$ is given a much better probability in the new policy with respect to the old one, then the objective gradient is null and the weights are not updated. The min operator ensures that the final objective is a pessimistic bound of the unclipped objective. In other words, the objective function is only ever clipped when either the advantage is negative and the current action is considered much worse than in the old policy or the advantage is positive and the current action is considered much better in the new policy than in the old policy.

The actual objective function being minimized also includes an entropy bonus to encourage exploration (the entropy of the policy is included in the computation of the objective function and gives a positive contribution).

PPO can be applied to both continuous and discrete action spaces, and it was empirically shown to have better sample efficiency than other on-policy algorithms and perform well with a smaller amount of hyperparameter tuning.

# Bibliography

[1] N. NILLSON, “Shakey the robot,” *SRI International, Technical Note*, vol. 323, 1984.

[2] R. R. Murphy, *Introduction to AI robotics.* MIT press, 2019.

[3] F. J. Perez-Grau, J. R. Martinez-de Dios, J. L. Paneque, J. J. Acevedo, A. Torres-González, A. Viguria, J. R. Astorga, and A. Ollero, “Introducing autonomous aerial robots in industrial manufacturing,” vol. 60, pp. 312–324.

[4] M. Jacquet and A. Franchi, “Enforcing vision-based localization using perception constrained n-MPC for multi-rotor aerial vehicles,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1818–1824, IEEE.

[5] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *2011 IEEE international conference on robotics and automation*, pp. 3400–3407, IEEE, 2011.

[6] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1653–1660, 2014.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[9] T. L. Munea, Y. Z. Jembre, H. T. Weldegebriel, L. Chen, C. Huang, and C. Yang, “The progress of human pose estimation: A survey and taxonomy of models applied in 2d human pose estimation,” *IEEE Access*, vol. 8, pp. 133330–133348, 2020.

[10] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu, “Rmpe: Regional multi-person pose estimation,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2334–2343, 2017.

[11] U. Iqbal and J. Gall, "Multi-person pose estimation with local joint-to-person associations," in *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part II 14*, pp. 627–642, Springer, 2016.

[12] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. Murphy, "Towards accurate multi-person pose estimation in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4903–4911, 2017.

[13] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," *Advances in neural information processing systems*, vol. 27, 2014.

[14] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1302–1310. ISSN: 1063-6919.

[15] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: Realtime multi-person 2d pose estimation using part affinity fields."

[16] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4724–4732, 2016.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[18] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[19] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*, pp. 213–229, Springer, 2020.

[20] Y. Liu, Y. Zhang, Y. Wang, F. Hou, J. Yuan, J. Tian, Y. Zhang, Z. Shi, J. Fan, and Z. He, "A survey of visual transformers," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[21] Y. Xu, J. Zhang, Q. Zhang, and D. Tao, "Vitpose: Simple vision transformer baselines for human pose estimation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 38571–38584, 2022.

[22] L. Stoffl, M. Vidal, and A. Mathis, “End-to-end trainable multi-instance pose estimation with transformers,” *arXiv preprint arXiv:2103.12115*, 2021.

[23] H. Dai, H. Shi, W. Liu, L. Wang, Y. Liu, and T. Mei, “Fasterpose: A faster simple baseline for human pose estimation,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 18, no. 4, pp. 1–16, 2022.

[24] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, “Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013.

[25] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, “Learning from synthetic humans,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 109–117, 2017.

[26] F. Picetti, S. Deshpande, J. Leban, S. Shahtalebi, J. Patel, P. Jing, C. Wang, C. M. I. au2, C. Sun, C. Laidlaw, J. Warren, K. Huynh, R. Page, J. Hogins, A. Crespi, S. Ganguly, and S. E. Ebadi, “Anthronet: Conditional generation of humans via anthropometrics,” 2023.

[27] Unity Technologies, “Unity Perception package.” `https://github.com/Unity-Technologies/com.unity.perception`, 2020.

[28] A. Qammaz and A. A. Argyros, “Mocapnet: Ensemble of snn encoders for 3d human pose estimation in rgb images.,” in *BMVC*, p. 46, 2019.

[29] W. Shan, Z. Liu, X. Zhang, Z. Wang, K. Han, S. Wang, S. Ma, and W. Gao, “Diffusion-based 3d human pose estimation with multi-hypothesis aggregation,” *arXiv preprint arXiv:2303.11579*, 2023.

[30] D. C. Luvizon, D. Picard, and H. Tabia, “2d/3d pose estimation and action recognition using multitask deep learning,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5137–5146, 2018.

[31] G. E. Hinton, A. Krizhevsky, and S. D. Wang, “Transforming auto-encoders,” in *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*, pp. 44–51, Springer, 2011.

[32] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.,” in *International Conference on Learning Representations*, 2019.

[33] F. D. S. Ribeiro, K. Duarte, M. Everett, G. Leontidis, and M. Shah, “Learning with capsules: A survey,” *arXiv preprint arXiv:2206.02664*, 2022.

[34] G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with em routing,” in *International conference on learning representations*, 2018.

[35] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” *Advances in neural information processing systems*, vol. 30, 2017.

[36] T. Hahn, M. Pyeon, and G. Kim, “Self-routing capsule networks,” *Advances in neural information processing systems*, vol. 32, 2019.

[37] P. Barham and M. Isard, “Machine learning systems are stuck in a rut,” in *Proceedings of the Workshop on Hot Topics in Operating Systems*, pp. 177–183, 2019.

[38] N. Garau and N. Conci, “Capsulepose: A variational capsnet for real-time end-to-end 3d human pose estimation,” *Neurocomputing*, vol. 523, pp. 81–91, 2023.

[39] I. Ramirez, A. Cuesta-Infante, E. Schiavi, and J. J. Pantrigo, “Bayesian capsule networks for 3d human pose estimation from single 2d images,” *Neurocomputing*, vol. 379, pp. 64–73, 2020.

[40] F. D. S. Ribeiro, G. Leontidis, and S. Kollias, “Capsule routing via variational bayes,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3749–3756, 2020.

[41] I. Sárándi, T. Linder, K. O. Arras, and B. Leibe, “Metrabs: metric-scale truncation-robust heatmaps for absolute 3d human pose estimation,” *IEEE Transactions on Biometrics, Behavior, and Identity Science*, vol. 3, no. 1, pp. 16–30, 2020.

[42] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” *arXiv preprint arXiv:1610.02136*, 2016.

[43] N. B. Gundavarapu, D. Srivastava, R. Mitra, A. Sharma, and A. Jain, “Structured aleatoric uncertainty in human pose estimation.,” in *CVPR Workshops*, vol. 2, p. 2, 2019.

[44] J. N. Kundu, S. Seth, P. YM, V. Jampani, A. Chakraborty, and R. V. Babu, “Uncertainty-aware adaptation for self-supervised 3d human pose estimation,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 20448–20459, 2022.

[45] Z. Liu, R. Feng, H. Chen, S. Wu, Y. Gao, Y. Gao, and X. Wang, “Temporal feature alignment and mutual information maximization for video-based human

pose estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11006–11016, 2022.

[46] A. Loquercio, M. Segu, and D. Scaramuzza, "A general framework for uncertainty estimation in deep learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3153–3160, 2020.

[47] J. Zhang, Z. Tu, J. Yang, Y. Chen, and J. Yuan, "Mixste: Seq2seq mixed spatio-temporal encoder for 3d human pose estimation in video," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13232–13242, 2022.

[48] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015.

[49] J. Liu, J. Rojas, Y. Li, Z. Liang, Y. Guan, N. Xi, and H. Zhu, "A graph attention spatio-temporal convolutional network for 3d human pose estimation in video," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3374–3380, IEEE, 2021.

[50] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, S. Sridhar, G. Pons-Moll, and C. Theobalt, "Single-shot multi-person 3d pose estimation from monocular rgb," in *2018 International Conference on 3D Vision (3DV)*, pp. 120–130, IEEE, 2018.

[51] D. Pavllo, C. Feichtenhofer, D. Grangier, and M. Auli, "3d human pose estimation in video with temporal convolutions and semi-supervised training," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7753–7762, 2019.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations (ICLR 2015)*, Computational and Biological Learning Society, 2015.

[53] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[54] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, "Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling," *IEEE Transactions on Image Processing*, vol. 29, pp. 1999–2012, 2019.

[55] F. Bolelli, S. Allegretti, and C. Grana, "Quest for speed: The epic saga of record-breaking on opencv connected components extraction," in *International Conference on Image Analysis and Processing*, pp. 107–118, Springer, 2022.

[56] M. Ryll, G. Muscio, F. Pierri, E. Cataldi, G. Antonelli, F. Caccavale, and A. Franchi, "6d physical interaction with a fully actuated aerial robot," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5190–5195, 2017.

[57] M. Hamandi, F. Usai, Q. Sablé, N. Staub, M. Tognon, and A. Franchi, "Design of multirotor aerial vehicles: A taxonomy based on input allocation," *The International Journal of Robotics Research*, vol. 40, no. 8-9, pp. 1015–1044, 2021.

[58] M. Jacquet and A. Franchi, "Motor and perception constrained NMPC for torque-controlled generic aerial vehicles," vol. 6, no. 2, pp. 518–525. Conference Name: IEEE Robotics and Automation Letters.

[59] A. Klöckner, "Behavior trees for uav mission management," *INFORMATIK 2013: informatik angepasst an Mensch, Organisation und Umwelt*, pp. 57–68, 2013.

[60] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[61] A. Serrano-Munoz, D. Chrysostomou, S. Bøgh, and N. Arana-Arexolaleiba, "skrl: Modular and flexible library for reinforcement learning," *Journal of Machine Learning Research*, vol. 24, no. 254, pp. 1–9, 2023.

[62] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[63] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[64] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[65] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," *The ICLR Blog Track 2023*, 2022.

[66] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

[67] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[68] G. Corsini, M. Jacquet, H. Das, A. Afifi, D. Sidobre, and A. Franchi, “Nonlinear model predictive control for human-robot handover with application to the aerial case,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7597–7604. ISSN: 2153-0866.

[69] D. S. Blough and R. B. Millward, “Learning: operant conditioning and verbal learning,” *Annual review of psychology*, vol. 16, no. 1, pp. 63–94, 1965.

[70] K. Zhu and T. Zhang, “Deep reinforcement learning based mobile robot navigation: A review,” *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[71] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

# List of Figures

1.1 Typical powerline maintenance scenario . . . . . . . . . . . . . . . . . . 2

2.1 Typical output of a 2D pose estimator model. The keypoints given as output are red points in the image, and the linking between them are decided a priori by the skeletal model being considered. Image by [9] Licensed under CC BY 4.0 . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.2 Fictitious representation of a heatmap given as output by a pose estimator. This representation wishes to highlight several characteristics of a heatmap: *non-normalization* (the sum of all values is not 1), *noisy values* (especially far from the peaks, where negative values are common), *multimodality* (it is possible to have several peaks in a heatmap, if the same type of skeletal keypoint has been detected in multiple locations). Multiple detections of the same keypoint may merge togther to create a single peak with wider support. . . . . . . . . . . . . . . . . . . . . . . 8

2.3 The PAF between the shoulder and elbow keypoints is designed to contain vectors that lay on the arm and point from the shoulder to the elbow. Taken with permission from [14] © 2017 IEEE . . . . . . . . . 8

2.4 The single parts of the object are recognized by lower capsules, while the whole object is recognized by the higher capsule. Viewpoint invariance is achieved by recognizing the single parts of the object and abstracting away the different instances of the same object: in fact, the pose matrix of the single parts is absolute, but the voting matrix only includes pose information of the single part relative to the object. Image from [33] . . 12

3.1 The COCO-18 skeletal model used by the pose estimator. The 18 keypoints are: nose, eyes, ears, shoulders, elbows, wrists, hips, knees, feet, neck. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.2 In this frame, it can be noted that the pixels of the occluded joints (right knee, ankle and hip) actually correspond to occluded locations, and their depth measure will therefore be severely skewed. . . . . . . . . . . . . . 21

3.3 Pose estimation output (left) and heatmaps (right). The intensity of the colors in the heatmap representation is proportional to the value in the heatmap, *i.e.* to the unnormalized probability of finding the joint in the given pixel. The heatmaps have higher peaks where the keypoint is detected with higher confidence. . . . . . . . . . . . . . . . . . . . . . 23
3.4 Graph of the function $w_T$ . . . . . . . . . . . . . . . . . . . . . . . . 26
3.5 Diagram representing the upper body kinematic chain, using the joints detected by the pose estimator: wrists, elbows, shoulders and "neck" (actually *manubrium sterni*). . . . . . . . . . . . . . . . . . . . . . . . 27
3.6 Within-range bone lengths are marked with a green tick, while bone lengths outside of the acceptable range are marked with a red X. The combination of detected bone lengths detected above results in the combination of correct/incorrect keypoints reported below. . . . . . . . . . 28
3.7 Camera view for the drone-mounted RealSense. The only occlusion is part of the propeller in the top left corner. . . . . . . . . . . . . . . . . 31
3.8 Histogram of the running time needed for the $QoE$ computation (milliseconds) in 10000 runs. The first three columns account for over 90% of the trials. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

4.1 Examples of the multiview dataset generated with Unity. The background textures are randomized. . . . . . . . . . . . . . . . . . . . . . . . 36
4.2 The profile of $\alpha$ for different values of $\beta$, assuming a maximum episode length of 100 steps. On the x axis: the step number; on the y axis: the value of $\alpha$. Note how the parameter choice gives more or less influence to the "distance from target" reward component. . . . . . . . . . . . . 39
4.3 High level schematic of the three processes involved in the reinforcement learning pipeline. The arrow direction indicates the order of execution of the processes, not the direction of communication„ which is bidirectional between the RL agent and each of the other two modules. . . . . . . 40
4.4 Examples of frames from the Blender 3D simulation wiht synthetic humans 41
4.5 Discretization of the Blender 3D environment: in this case the x coordinate is increasing. Note how the camera is always facing the human model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
4.6 In a 19x19 grid, the drone (letter D) appears in a random position for each episode and can move to the 8 adjacent cells at each step (the ones with a yellow circle). The human (letter H) instead is always in a fixed position in the center of the grid. . . . . . . . . . . . . . . . . . . . . 43

4.7 Schematic representation of the basic Network architecture used for learning an exploration policy. The LSTM internal structure is not explicitly drawn, as it is simply the standard implementation of a 2-layer LSTM. The hidden state vector **h** is meant to show the recurrent computation that allows to keep track of a sequence of inputs. . . . . . 46