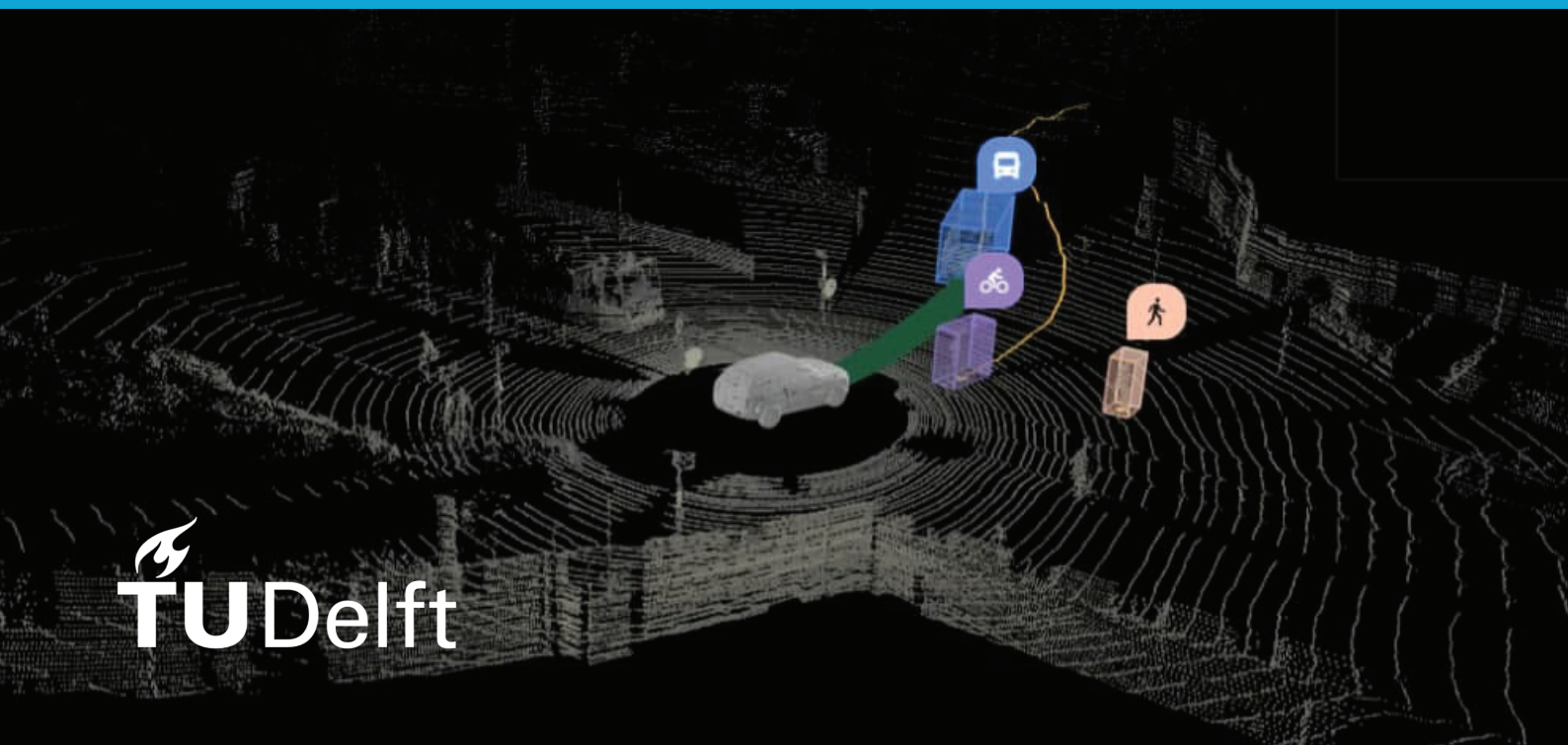
MSc thesis

# Exploring Computer Vision for Human Awareness in Human-Cobot Interactions

Elvis Borges

2021



TU Delft

# EXPLORING COMPUTER VISION FOR HUMAN AWARENESS IN HUMAN-COBOT INTERACTIONS

A thesis submitted to the Delft University of Technology
for the degree of

Master of Science in Software and Networking, Embedded Systems

by

Elvis Borges

Sep 2021

Elvis Borges: *Exploring Computer Vision for Human Awareness in Human-Cobot Interactions* (2021)
This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit
http://creativecommons.org/licenses/by/4.0/.

The work in this thesis was made in the:

Smart Advanced Manufacturing Lab
Department of Sustainable Design Engineering
Faculty of Industrial Design Engineering
Delft University of Technology

Supervisors: Dr. Doris Aschenbrenner
Dr. Arjan Genderen

# ABSTRACT

Designing the future of work is a goal that is currently being tackled by both industry and academia. In a scenario where we have Human-Robot hybrid teams, it will be necessary for both human and robots to be aware of each other in their respective tasks. This thesis explores how specific robots (cobots - collaborative robots) can use Computer Vision-based methods for awareness of a human's presence, while using this data to influence the cobot's own behaviour. After explorations of the Computer Vision field in regards to this problem, and selected approaches in the domain of Human Awareness for Human-Robot Interaction in literature, this thesis presents a proof of concept software/hardware system that, given all the apriori findings, implements simple demos that demonstrate both the feasibility and the suitability of a pure vision-based system for the problem of human awareness. It then analyses the strengths and weaknesses of the proposed system solution, ending with a road-map of recommendations for a following production prototype.

# ACKNOWLEDGEMENTS

This thesis is a distillation of the past months of my life, but also, in a way, to all of the moments that have lead me here. This section is dedicated to all the people who gave me advice, interesting leads, gave me feedback, kept me sane, offered their friendship, love and support as I undertook this current adventure (and all of my previous ones). I am forever thankful.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# List of Algorithms

# ACRONYMS

# 1 INTRODUCTION

Since the introduction of robots in the workforce, there has been research focused on how to develop human-robot interactions within this context (Sheridan [2016]). In current times, the recent advance of computer vision techniques has lead to the increasing ability of "smart" vision-based Human-Robot Interaction (HRI) computational systems, and the possibility of integrating these systems in the workforce productively. Within this new research direction, this thesis explores exactly how these techniques can be deployed from literature, for both HRI and Computer Vision (CV), their implementability, and proposes a demo system, to simulate such interactions, while testing and prototyping it holistically, and its components.

## 1.1 RESEARCH PROJECT

This thesis project will be centered on a Collaborative Robot (Cobot) (this will be refined further in Section 1.4), doing a set of preset action/tasks, influenced by vision feedback of a human "operator" (thereby using a computational analog to the way humans perceive the environment and use that visual date to adapt their behaviour).

The proposed "human-robot hybrid" system used in the demos that tries to validate this will have the setup described in Figure 1.1 (this setup will be explored in depth in Chapter 4).

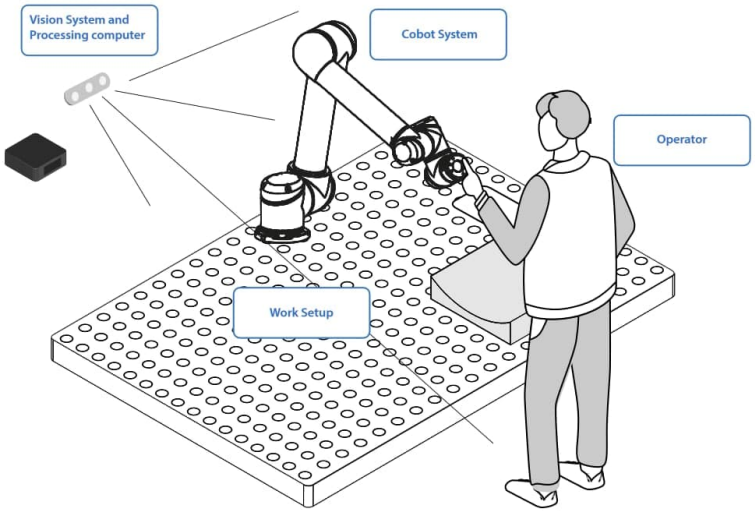



**Figure 1.1:** Schematic Hybrid System Setup with Human, Cobot, Camera and Compute in work environment

There will an exploratory study of the fields of CV and HRI, followed by the development of an intelligent system prototype to detect the presence of an human operator, to influence task behaviour. Initially the goal is to send a safety signal (such as a stop or slow-down signal). If more time is available, more extensive interactions will be explored. The initial sensing will be centered on raw vision

using data through depth enriched video feeds, from a RGB-d depth camera, in our case an Intel realsense D431i (detailed in full in Chapter 4).

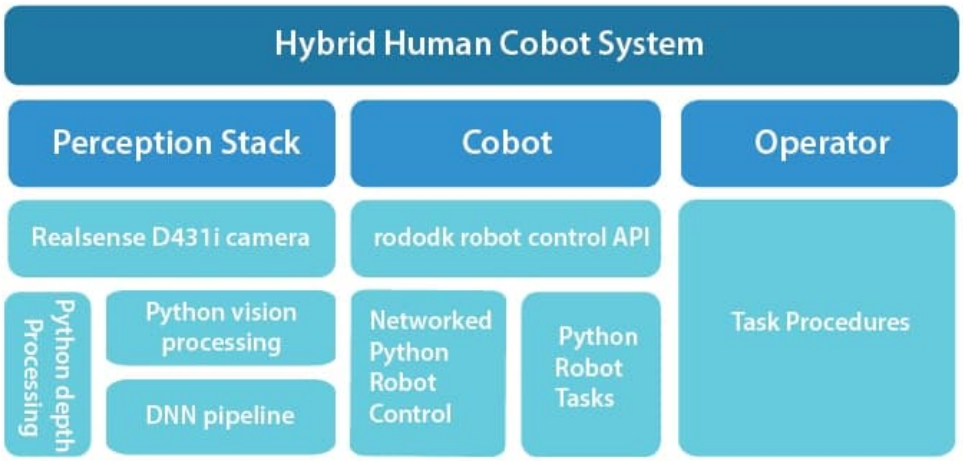



**Figure 1.2:** Final System Tree Design for cobot setup

We can see from the final system tree overview analysis (Figure 1.2) that the main areas of focus for the prototype are Perception and Sensitivity. This diagram is presented here to give a brief overview on the intended design, with the approach that lead to this point being explored further in Chapter 4.
Perception will be used for detecting human intent: 3D camera/pose/motion tracking (which could be used for intention analysis and predictive responses from the robot (Colley et al. [2020])), and from the sensitivity perspective, establishing feedback between robot and human (specifically robot ← human (vision, sensors etc)). A more detailed analysis of both the overview and system tree will be done in the respective approach sections of the following 3 chapters.

This prototype, along with the demos devised to test it, pertain to the main devised research question (1.4), that will be refined by the end of this chapter (in Section 1.3 and Section 1.4), this thesis focuses on using computer vision based models for triggering the Cobot reactions. The motivations for this choice, and further development of this topic will be reviewed in the next two chapters.

## 1.2 CONTEXT

After the automation revolution in the manufacturing world in the 1970s, the manufacturing industry is experiencing a new technological wave - known as Industry 4.0 (Ludwig et al. [2018]).

Industry 4.0 roughly encompasses all the recent digital technological advances, that of smart data pipelines, IoT, and the increasing use of smart Automation and AI techniques into production and manufacturing systems.

Simultaneously, societal changes have led to an aging population (and workforce) in Developed Economies, and an increasing trend in automation by substituting human labour, initially in mechanical work, but increasingly for creative endeavors (BCG Group [2015]).

With these new paradigms, the future of work is being redefined, and questions such as what jobs will disappear, how will jobs profiles evolve, and others must come to the fore (Figure 1.3).

From the manufacturers' standpoint, the adoption of these new technologies is paramount for growth and productivity, but there are concerns in the literature on the automation approach first introduced in the 1970s, which tended towards the

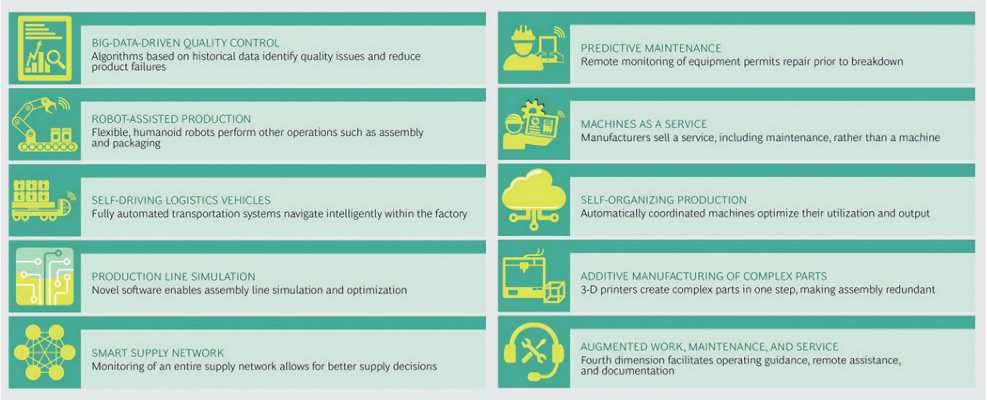



**Figure 1.3:** Digital Influence in Workforce (BCG Group [2015])

complete replacement of human labour from assembly and production (BCG Group [2015]).

Meanwhile, robot assisted production is predicted to become the largest component on how an industrial worker will do their job, creating new job families while making others obsolete.

This presents a few possible scenarios for the future of human work and the extent of robots and humans in labour (Ittermann and Niehaus [2018]):

- "Robots take over" - a complete substitution of human labour by automated solutions
- The "Operator 4.0", where the integration and focus of the tech-augmented human worker (Romero et al. [2016]), ends with Human-centered organization of factories and work
- "Winners and Losers" - a polarized scenario where "middle layer" sections of work, such as administrative tasks, etc are automated, and both simple work and highly qualified would remain to humans
- "No boundaries" - complete flexibility and modularity, leading to decentralization of the classical patterns of work, that of the fixed hierarchical organization and management structures, becoming time and location independent

Given these options, the role of research for the future of work relies on developing towards the optimistic "Operator 4.0" scenario.

This "stronger human factor" research leads us to "human-automation symbiosis work systems", that leverage both intelligent machines as well as human intelligence.

Most literature in this topic concerns new technologies in industry 4.0 (BCG Group [2015]; Inagaki et al. [2003]; Dalenogare et al. [2018]). There are also studies focused specifically on the human factor (Gorecky et al. [2014]), but hybrid human-robot systems are not very prevalent yet.

A first premise of this research leads us to the pretext that, **as more hybrid human-robot systems come into the work space, HRI (Human Robot Interaction) and Co-Production are increasingly relevant for developing the future of work.**

## 1.3 BROADER PROBLEM

Summarizing the previous sections, the manufacturing industry is at a crossroads. The envisioned "Industry 4.0" paradigm will depend on humans working alongside intelligent machines and robots in factories, and explores the introduction of

new technologies like Cobots (collaborative robots), Augmented and Virtual Reality, but also Artificial Intelligence into the production floor (BCG Group [2015], Ludwig et al. [2018]), advocating for a stronger *human factor* within the "Industry 4.0" manufacturing environment.

As such, new interactions, methods and tools are needed in order to be able to design the "future of work" for production workers, one that will involve further human-intelligent system integration, in which technology helps to enhance the physical, sensing and cognitive capabilities of the worker; and the collaboration levels that this can enable, specifically for designing and analyzing systems for the Human-Robot Interactions needed for production systems and their related tasks (Figure 1.4). Currently, most literature concerns the new technologies used in industry 4.0 (Wang et al. [2019a], Wilhelm et al. [2016])

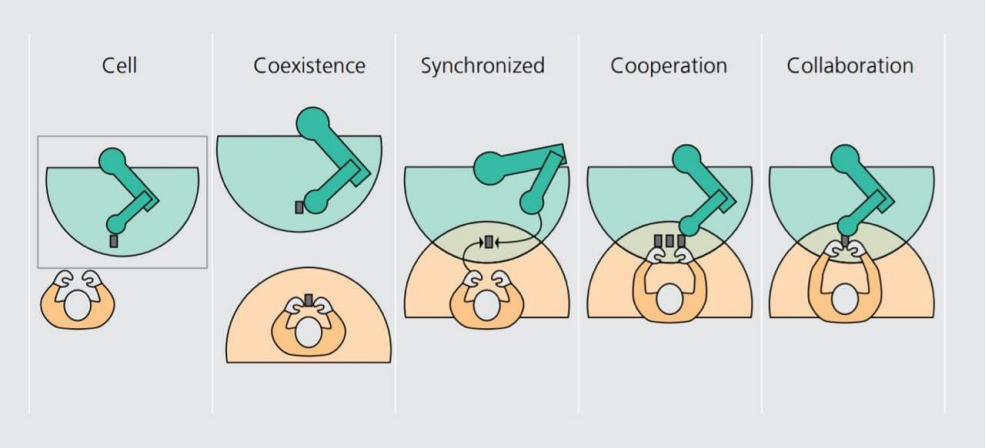



**Figure 1.4:** Proposed cooperation levels in Cobot-Human tasks (Bauer et al. [2016])

One of these research directions is the effective combination of an autonomous robot and the human worker, which together form what we interpret as a "human-hybrid system". This can be be split into problems such as task allocation, in which, for example, the "Sharing and trading of control" approach considers different levels of automation and determines the appropriate level of the production environment depending on the task at hand between the robot and the human user (Sheridan [2016] and Inagaki et al. [2003]).

These themes also relate to the dutch AI research agenda (van den Bosch et al. [2019]), where the aim is for humans and AI to cooperate, leveraging complementary strengths, and where each agent focuses on the tasks they are optimally suited for. This agenda encompasses a few major research question fields - How do we create an AI system, such that:

RQ-1. Humans and Robots interact and understand each other's behavior in context?

RQ-2. The human trusts the autonomous system?

RQ-3. Enables task sharing between hybrid human-robot teams?

For this collaboration and coordination with humans, the AI system needs to observe and understand, to some degree, human behaviour (and also explain itself so that humans understand it).

For example, as a system, how can it capture and reason about the human state, such as interaction behaviour? How to design systems such that these hybrid teams behave as desired, such that the operator can "trust" the system, engage it safely and vice versa? There is a growing sentiment in literature that hybrid human-and-AI teams can develop solutions that neither single human, nor AI-only teams can achieve. This research direction leads to how this future of hybrid AI-human work can be developed.

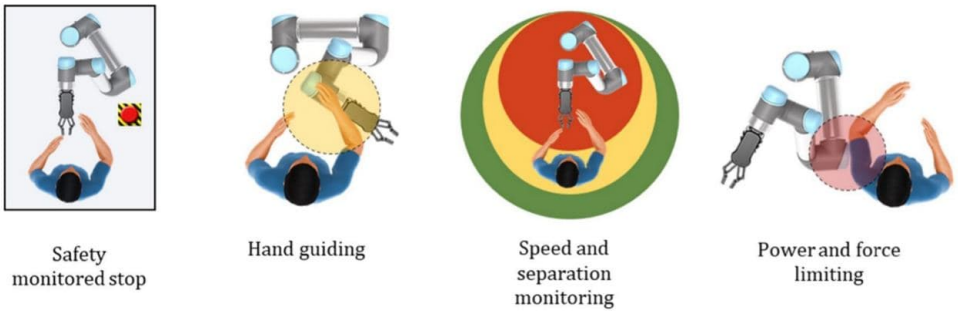



**Figure 1.5:** Different Safety Aspects involved in Cobot workflows according to ISO15066, (Malik and Bilberg [2019b])

## 1.4 SCOPE AND CONTRIBUTIONS

From the initial breadth of the previous research questions, we must come to the key areas of focus for this thesis research.

This thesis will focus on the robot perspective in HRI. More precisely, it will focus on all the CV based techniques that can make a Human-Cobot system feasible. In this context, a Cobot is a class of robots geared around direct human-robot interaction within a shared space, or where humans and robots are in close proximity (Colgate et al. [1996]). Cobot applications contrast with traditional industrial robot applications in which robots are isolated from human contact (IFR [2018]).

From this frame, the initial research question 1 (1.3), will be refined into:

RRQ-1. How can we create a Vision based system such that a Cobot (collaborative robot) can react to a human within its environment?

This will be this thesis' key research focus. A literature study and a state of the art of deep Neural network based vision techniques geared towards human awareness will be presented, as well as a brief field history and fundamental background needed for the further analyses. The main research question (Section 1.4) will also be the key goal of the thesis prototype, to test the efficacy of a vision based system for human perception and reaction from a Cobot.

Research questions 2 and 3 (1.3), pertaining more to the human-robot trust dynamics will only be touched upon in regards to the corresponding literature, presented mainly in the first 4 sections of Chapter 3, together with HRI's state of the art and background, finally motivating why the chosen vision-led approach is valid.

Furthermore, an implementation design is presented (available through the **4TU repositories**), for a system that uses pure vision for developing early stage human-awareness. Both the design process, as well as the implementation and results are presented, together with qualitative and quantitative benchmarks.

## 1.5 CONCRETE PROBLEM

By reframing the broader overview, there are two main sides to answering these research questions. The *human perspective*, in which the operator needs to be aware of and interact productively with the robot; and *the robot perspective*, where it needs to be aware of the human, and be able to engage intelligently to both the operator and the production context (Figure 1.4). There are different trends in the market on the level of cooperation (Figure 1.6), and it can be seen that these hybrid-scenarios are still in their infancy.

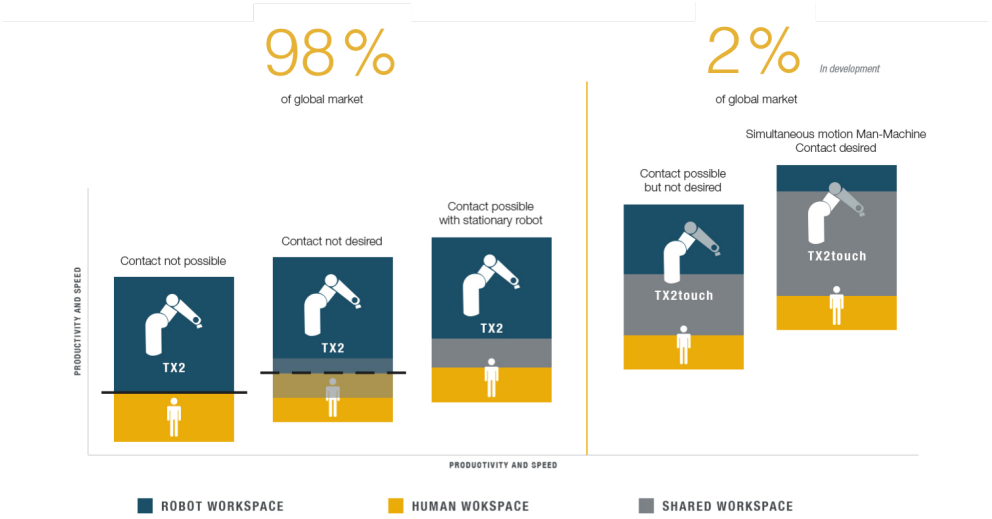



**Figure 1.6:** Conceptual framework for "stages" in HRC, and industrial robotics market penetration as of 2016 (Stäubli [2016])

From the main research question (Section 1.4 ), we will explore recent advances in the field of computer vision (Hafiz and Bhat [2020] and Sultana et al. [2020]), more specifically advances in the use of deep learning for Computer Vision tasks, that have allowed more advanced tracking algorithms to be deployed in production systems, from detecting the location/gait of the human, to recent advances in inducing intent/trust from such vision feeds, and approaches leading to helping the awareness of the human by the Cobot system ( Colley et al. [2021], Avelino et al. [2021], Widder et al. [2021] and Esterwood et al. [2021]).

The initial awareness goal is centered on safety (Figure 1.5), having the Cobot receiving signals pertaining to the awareness of the human, and it influencing a real task ( Safety stopping/slowdown, etc).

Further on, more complex interactions will be explored both through literature review (Chapter 3), as well as in the prototype development process (Chapter 4).

## 1.6 THESIS OVERVIEW

Beyond this chapter, Chapter 1, which presented the goals, broader problem context and motivations for the intended research question and project, as well as the contributions herein, a brief overview of the full thesis can be given. After this introduction, Chapter 2 introduces CV, the sub-domains that are most relevant to this research, background into previous solutions and their weaknesses, leading towards the Deep Learning (DL) based solutions implemented herein, why they work, the state of the art techniques and datasets, as well as how they can be evaluated. Following this, Chapter 3 introduces the domain of HRI, provides a literature background into general approaches to modelling human focused interaction. It then provides a literature review of methods for HRI, both vision and non-vision based, ending with a discussion on the approach/simplifications used for the research project. Given this base foundation, Chapter 4 gives the general outline of the designed system, further delving into each system component's details in the later sections, while briefly underlying the process taken, ending in the final implemented system. Then, Chapter 5 presents both qualitative and quantitative results for the implemented system, as well as a critical analysis of this data. Finally, Chapter 6 explores the main weaknesses of the system, its suitability given the final research question, and avenues for further work.

# 2 VISION; BACKGROUND,RELATED WORK AND STATE OF THE ART

Vision is arguably the most important component of the designed system. In the following sections, an in-depth review of the suitability of this field for the problems stated in Chapter 1 will be presented, together with all the necessary domain knowledge and literature research for enabling the system.

## 2.1 WHY VISION?

In Chapter 1, one of the main research questions introduced was the **problem of operator awareness in a robotic system**. There are different modalities that could achieve this, from sensor networks and IoT, to sound processing or through computer vision, among others.

This thesis focuses especially on vision, because CV is a rich data source for intent tracking (Colley et al. [2021]), but also due to promising recent advances in the field of Machine Learning (ML)-based algorithms which have lead to increased solveability of CV problems, both in cost and performance, of a large set of related sub-problems deemed previously unpractical, specifically in the field of Human Pose Estimation (HPE) (Zheng et al. [2021]).

Modelling human feedback for robotic design (Noceti et al. [2020]) has been a goal of robotics design from the beginning of the field, and there is a significant section of researchers dealing with the problem through inferring human awareness through computer vision, with this approach having had promising results in recent years (Zheng et al. [2021]).

While this approach still presents significant challenges, such as robustness, insufficient training data, depth ambiguities, this thesis **will justify why this approach should be considered and can be implemented in such a cobot system.**

From the previous points, a postulate is devised: for our problem and main research question (Section 1.4), **a vision led approach can build an effective system for inferring operator intent, and react accordingly during a task**.

### 2.1.1 Vision vs Other Sensing Modalities

One of the main questions one may ask is **"is vision alone sufficient for building a reactive system to humans"**, and **"what other methods of perception are used in the field, and how does pure vision compare to this?"**. Pure vision's main disadvantage is the inherent difficulty in inferring a 3D scene from purely a 2D input (Kopf et al. [2021]; Casser et al. [2019]).

There are also other approaches for human awareness that are not vision-based. Some of these are explored in Chapter 3. An example of other sensing modalities for this problem would be for example LIDAR - which constructs accurate high definition 3D maps of the environment. However these sensors are far more expensive and the solutions derived from these are therefore less scalable. Again the main challenge with pure vision is getting well structured 3D information and this is something LIDAR excels at.

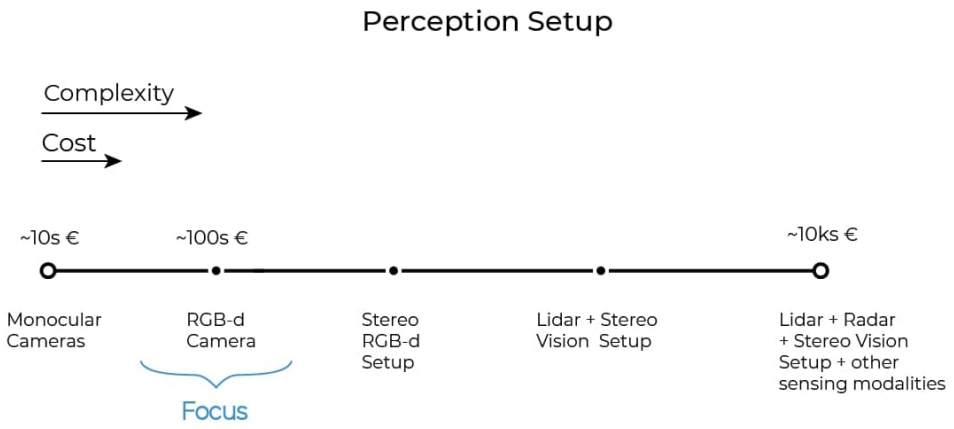



**Figure 2.1:** Perception Setup Approaches

One can see a range of vision setup solutions, both in terms of complexity and cost (Figure 2.1). This will be further developed in Section 4.4, but the initial premise is that pure lower dimensional vision feeds (with added vision-based depth estimation) are **sufficient for 3D human estimation**. We can begin by defining the fields within CV that will be explored.

## 2.2 BACKGROUND

### 2.2.1 Problem Domain

There are a few sub-fields within Computer Vision which are specially relevant for the task of human awareness and our research question (1.4). First, we would need to track the presence of the human throughout the frame. There are several vision sub-fields relevant to this, with different complexities in the "tracking".

**Object Detection** (Figure 2.3a) focuses on providing both the "class" of objects (this includes, but is not limited to humans) within an image and their location through bounding boxes (the part of the image the object is located in) or centroids/features (tracking specific landmarks of an object) (Figure 2.2).


**(a)** Box: center $(b_x, b_y)$ , height $b_h$ and width $b_w$ **(b)** Reference points $(l_{1x}, l_{1y}), \ldots, (l_{nx}, l_{ny})$

**Figure 2.2:** Bounding box and feature representations (Amidi and Amidi [2020])

**Semantic Segmentation** predicts labels for each image pixel so that each one is labelled to a "class", depending on the object or region enclosing the given pixel. Thus these algorithms will split a given image into "semantic sections" (Figure 2.3b).

Building up on these previous sub-fields, **Instance Segmentation** is an enhancement which attempts to provide unique labels for separate instances of objects belonging to the same object class in the given image (i.e. person 1, person 2, etc). **Panoptic Segmentation** mixes the previous two sub-fields together (finding a solutions to both object detection and semantic segmentation while doing so, i.e. a semantically segmented image with individual segments for different instances of the same object class) (Hafiz and Bhat [2020]). Another enhancement, **Part-Based Segmentation** decomposes each segmented object into its respective sub-components i.e., a person into head, torso, etc.

**(a)** Object Detection **(b)** Segmentation

**Figure 2.3:** Relevant Sub-fields of Computer Vision (CV)

Closing in our specific problem, **Pose Estimation** (Figure 2.4) blends all these fundamental building blocks, to predict and track the location of a person or object through the localization of **main keypoints** in either images or videos. It may also be used for determining the position and orientation of a camera relative to a given person or object (Gong et al. [2016]). These keypoints depend on the target object/person. For humans, they usually represent major joints like the knee.

**Figure 2.4:** 2D Human Pose Estimation

Pose estimation is mainly divided into two major domains - *2D pose estimation* (keypoint location given relative to the 2D frame), and *3D pose estimation* (provides not only 2D localization information, but also an additional dimension of depth),

this allows for actual spatial positioning of a depicted person or object, end to end. Objects are mostly rigid. Predicting the position of these objects is known as rigid pose estimation. By focusing these on Humans, we come to HPE. Humans belong to the category of objects that are flexible i.e. when the body "deforms", the keypoints will be in different positions relative to before. As an addendum, there is a distinction between detecting one (single pose estimation) or multiple objects (multi pose estimation). Additionally, we may also encounter **Head Pose Estimation** where facial landmarks are used to obtain the 3D orientation of a human head with respect to the camera. Next, a brief field overview and history will be given.

### 2.2.2 Field Overview and Advent of Deep Learning

Initially, most techniques involved in these problems were based on **feature** representation and detection in images (where a feature is an identifying component of an object i.e. corners or edges).

These techniques would focus on detecting and tracking these features in given frames, using local descriptors (descriptions of the visual features that identify key characteristics of a feature - shape, color, etc).

Figure 2.5: SIFT example

From these, the most relevant approaches would start from either Scale Invariant Feature Transform (SIFT) (Lowe [1999], Figure 2.5) or Histograms of Oriented Gradients (HOG) (Dalal and Triggs [2005], Figure 2.6), which detect and track features through image processing based algorithms. These methods try to make the methods invariant to scene/capture configurations, so that detection is independent from the context.

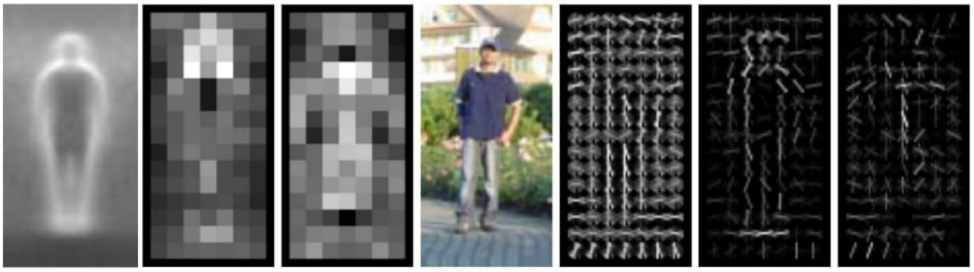


Figure 2.6: Stages in Histograms of Oriented Gradients (HOG) based Human Detection

From these descriptors, an higher level semantic representation can be assembled, by constructing spatial arrangements that allow the parameterization of angles and joint positions as vectors, using techniques such as Bag of Visual Words (BOV) (Sivic and Zisserman [2003], Figure 2.7), in which a text retrieval inspired approach (similar to search query optimization in search engines) is used, where documents are parsed into distinguishing words, which are then used to represent that particular document by a vector with components ordered by the frequency of occurrence of these words and their "contribution importance" to the query.

A particular text query is then retrieved by computing its vector of word frequencies and returning the documents with the closest (measured by angles) vectors. BOV matches pre-computed descriptors (using vector quantization of descriptor vectors as a visual analogy of a word instead), to an object detection description, through inverted file systems (with a structure similar to the indexing process of chapters in an ideal book index page, where essentially every object is connected to a Bag-of-Words description of that same object), with document rankings, or using the Fisher Kernel (FK) (Perronnin et al. [2010]) - an extension to BOV going beyond using just

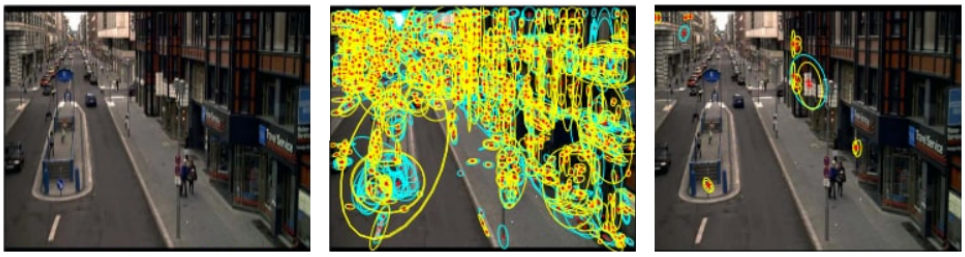
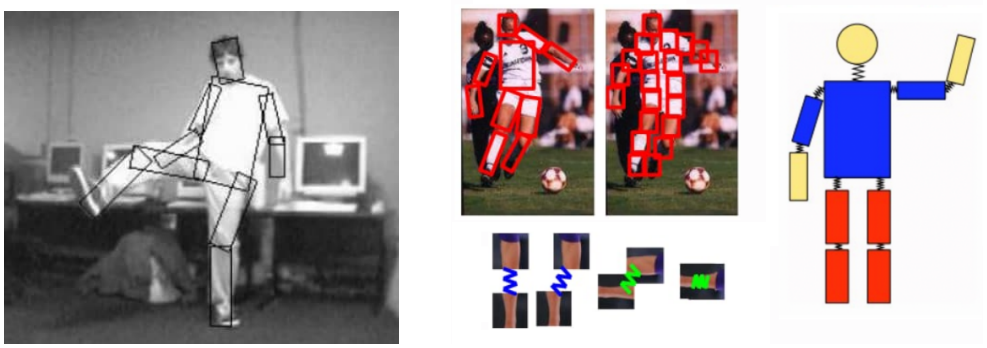


**Figure 2.7:** Stages in Bag of Visual Words (BOV) for Sign Detection

basic count statistics for the image descriptor quantization.

Other, more spatial approaches were also used, in which an object was represented by a collection of "parts" arranged in a deformable configuration e.g. *Pictorial Structures* (Felzenszwalb and Huttenlocher [2005], Figure 2.8a ) and deformable part models (Yang and Ramanan [2013], Figure 2.8b).

The main problems with these approaches was the need for specific, extensive tuning from domain experts to get useful results, and bad generalization performance. A more in-depth review of these classic methods can be found in Gong et al. [2016].



**(a)** Pictorial Structures **(b)** Deformable Part Models

**Figure 2.8:** Classical Model Abstractions for Human Pose Estimation (HPE)

In 2012, advances in GPU performance and cost, together with advances in machine learning and Neural Networks (NNs), brought the introduction of deep learning to computer vision, starting with *AlexNet* (Krizhevsky et al. [2012]).

These methods removed the need for domain experts to break down visual recognition into local descriptors, and allowed for more *end-to-end* solutions, on which large NNs architectures, tuned to work well with visual inputs, were trained on large labelled datasets (in the previous case, *ImageNet*).

These methods (mainly Convolutional Neural Networks (CNNs), and more lately transformers) not only simplified the algorithms in use, but they also provided far more flexibility of techniques, higher performance, and more conventional pathways for performance improvement (Zheng et al. [2021]). Moreover, further developments have led to increasingly large parts of these structures being left to optimization, providing a promising pipeline for future improvements - due to the inherent differentiability of ML models, something which will be explored further in the next section.

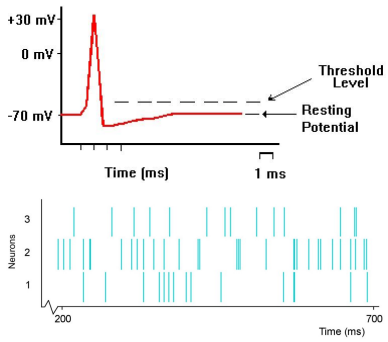
### 2.2.3 Neural Networks

As mentioned previously, the normal approach of decomposition (breaking problems down into sub-problems, creating algorithms for those and then stacking these together to solve the general ones) to tackle a challenge is not always the best approach. This classical methodology has proven rather ineffective in some domains that are of great current interest, such as problems pertaining to recognition (machine translation, natural language processing & speech recognition,etc.) and in our particular case - Computer Vision (CV).

This is where Machine Learning (ML) comes in. It is a different approach to problem solving, where the focus is instead placed on **data and optimization**. Within the field of ML, we have the concept of **Supervised Learning** (Russell and Norvig [2009]), the task of learning a function that maps an input to an output based on input-output examples. This is the core principle of the paradigm that allows Neural Networks (NNs) to solve problems.

NNs are a class of bio-mimetic algorithms that try to tackle this task. The "vanilla" Artifial NNs (a.k.a. *multilayer perceptrons*), first introduced by McCulloch and Pitts [1943], are based on mathematical models of the behaviour of neurons in the brain. They can be seen as an adaptation of the spiking neuron model in biological neuron networks (Gerstner and Kistler [2002]), where certain neurons spiking trigger other "related" neurons to also fire, as seen in Figure 2.9b.

**(a)** Motor and Parietal Cortex, by Greg Dunn




**(b)** Membrane spike potential, train

**Figure 2.9:** The brain and its modelled behaviour

As seen in Figure 2.10, these models consist of a network of interconnected "neurons", which can be seen as entities/functions which provide a number - called its activation - (between 0 and 1), given inputs.

A layer is a set of neurons in the NNs hierarchical structure (the number of neurons in a layer is termed width). Layers besides the Input-Output (IO) layers (at both ends of the network) are called the hidden layers (the number of these layers in a network is termed depth). The exact meaning of these neuron activations, both on the IO layers and the hidden ones, depends on the network architecture, design and application. Generally, the activations of one layer will determine the activations of the next layer.

The information processing mechanism through which the network encodes information is encoded in how the activations of one layer lead to the activations of the "next" layer.

This encoding of information happens through the weights associated with the connections leading to a particular neuron. These weights are numerical values associated with the "connections" between a neuron and the neurons of the "previous" layer. A neuron's activation value is the scaled weighted sum of all of the connection weights associated with that neuron and the activation of the neurons from the previous layer on the other end of these connections. Since we want the

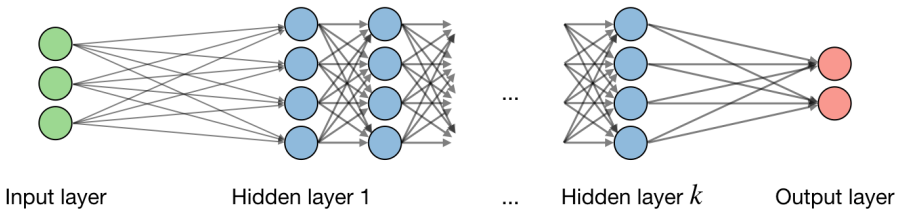



**Figure 2.10:** Classical Feed Forward Neural Network (FFNN) structure (Amidi and Amidi [2020])

activations in the range of (0-1), we also need activation functions, that will map the weighted sum result into our specified range (the activation functions are doing the scaling). Another parameter we might tweak is how "sensitive" a neuron's activation is to its weighted sum value, so we also have a bias, which is subtracted to the weighted sum before the result is scaled by the activation function.

By summarizing all of this information, a "forward pass" can be encoded as a simple succession of *multiply and add* matrix operations, to which we apply the non linear activation function at each step, from the input to the output layer. By denoting $i$ the $i^{th}$ layer of the network and $j$ the $j^{th}$ hidden neuron of the layer, we have:

$$z_j^{[i]} = \sigma(w_j^{[i]^T} x + b_j^{[i]}) \tag{2.1}$$

Where $\sigma$ denotes the activation function, **w** denotes the weight matrix, that stores all the weights in the network, **b** denoting the bias term, **z** denoting the succeeding layer activations and **x** the preceding one.

These neurons, and their interconnections provide the parameters that can be optimized to train a network, i.e. figuring out the valid values for the weights and biases such that the input-output mapping to input queries provides a solution to our problem. As such, the whole network models a function mapping its inputs into particular outputs.

Intuitively, the layered structure of the network would replicate the "breaking down" process of solving a problem, in which the earlier layers would, after training, encode information allowing for, taking the example of image recognition, detection of the initial features (edges and features), and the latter layers, encoding the detection of more complex structures and objects based on these primitives. A "forward pass" would progressively extract higher-level features (Figure 2.11) from the raw input until the final output result, the "inference".

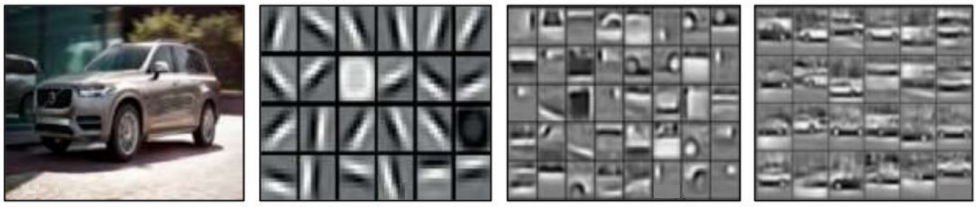


**Figure 2.11:** Abstraction levels in layers, from input images to high level features (Sze [2020])

These **activation functions** can be any desired function, and the performance of the network will depend on their choice. A core principle to note is that for any IO mapping there is an optimal set of activation functions such that the neural network contains the lowest amount of adaptable parameters to obtain a particular cost function value (a metric which can then be used, together with a learning method, to optimize the function).

Table 2.1 summarizes some often used Activation functions.

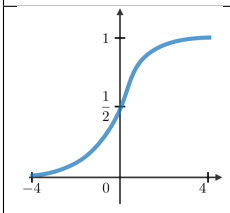
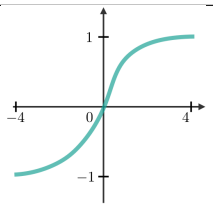
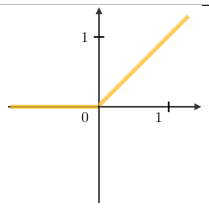
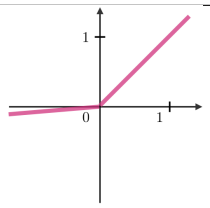
| Sigmoid | Tanh | ReLU | Leaky ReLU |
|---|---|---|---|
| $g(z) = \frac{1}{1+e^{-z}}$ | $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ | $g(z) = \max(0, z)$ | $g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$ |
|  |  |  |  |

Table 2.1: Different Activation functions (Amidi and Amidi [2020])

### *Why do NNs work?*

One might wonder, **why do NNs themselves work?** The mathematical proof underlying them is given by the **universal approximation theorem**, which asserts that, given appropriate weights, NNs can represent a wide variety of functions of interest (Csáji [2001]). Stated formally (for the single layer Feed Forward ("vanilla") artificial NNs) by Cybenko [1989], we have:

**Theorem 1.** *A feedforward neural net with at least one hidden layer with sigmoidal activation functions can approximate any continuous nonlinear function* $\mathbb{R}^p \to \mathbb{R}^n$ *arbitrarily well on a compact set, provided that sufficient number of hidden neurons are available.*

This principle can be generalized to arbitrary network architectures, depths, and widths (Zhou [2020]).

### *Learning*

Previously, the concept of **Cost Function** (aka loss function) was introduced. This is how the network can "learn". When a new network is started, its parameters, the weights and biases will start with random values. By taking the IO sample pairs from our dataset, we can feed the input part to the network, and compute the cost function from the difference between the computed output - the inference - and the expected output for that input. By running this on the entire dataset, measures such as the average cost of the network can be computed, which will provide a measure of how "bad" the network will be in the given task. A commonly used loss function is the "cross-correlation" entropy variety (Equation 2.2) (with $\mathbf{z}$ denoting the activation output and $\mathbf{y}$ the expected output):

$$L(z, y) = -[y \log(z) + (1 - y) \log(1 - z)] \tag{2.2}$$

The training process involves all the techniques and algorithms that find the minima of this cost function. The overall concept is, given that the NNs approximate a function, we try to find this cost function's minima by iteratively taking "steps" in the "direction" that decreases this loss function value. This "direction" is encoded mathematically by the negative gradient ($\nabla$) of our cost function, which is a vector that will "point", for any point of our function towards the set of parameters that will decrease its value. For a generic function $f$ the gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is defined at the point $p = (x_1, \ldots, x_n)$ in $n$ -dimensional space as the vector exemplified to the left of the following equations, with its application to the generic loss function (in relation to the network weights) on the right (Equation 2.3):

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix} \qquad \frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w} \tag{2.3}$$

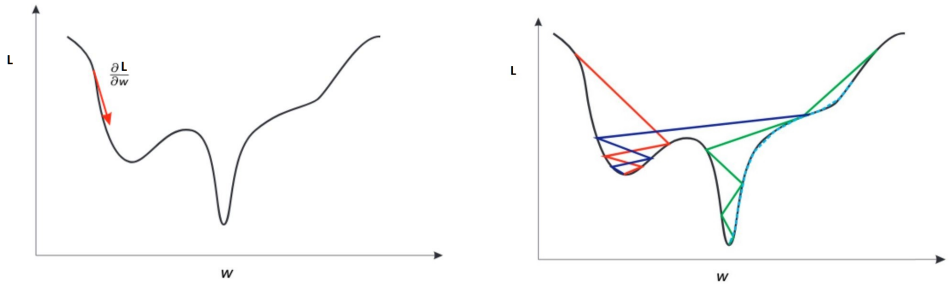
The algorithm which will compute this gradient (with the help of the chain rule of derivation) efficiently is called **backpropagation** (Rumelhart et al. [1986], Algorithm 2.1).

**Algorithm 2.1:** BACKPROPAGATION $(w, \alpha)$

**Input:** $w$, $\alpha$, training data
**Output:** $w$: the new network weights

1 **while** $L(z,y)$ *error has not stabilized* **do**
2 Take batch of training data;
3 Compute **forward propagation** to obtain corresponding **loss**: $L(z,y)$;
4 **Backpropagate Loss** to get **gradients with respect to the weights**: $\frac{\partial L(z,y)}{\partial w}$;
5 **Update network weights** by taking **step in negative gradient**: $w \longleftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}$;

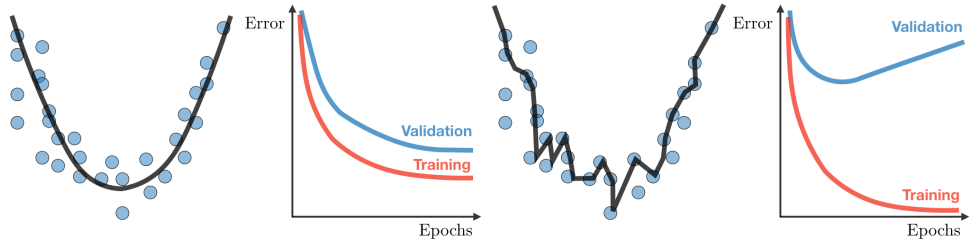
The learning process in NNs consists then on the **minimization of our cost function** (Figure 2.12a). For this, it is important that the cost function is "smooth", so that it is possible to find our local minima by our iterative step (the step size is denoted by $\alpha$ - Figure 2.12b) process in the direction of the negative gradient. This is known as **gradient descent** - a way to converge to the local minima of a cost function (this is a first order method, but there are others that can be used).




**(a)** Idealized movement in Loss function values **(b)** Training results with different step sizes

**Figure 2.12:** Feed Forward Neural Network (FFNN) Training Process

This learning process is not trivial. NNs have generalization ability due to the proposal that by reducing our cost function error, the NNs will encode an approximation of the manifold of the input dataset. Getting a good approximation of this data space is not easy (Figure 2.13) as, first the training dataset has to represent a **statistically significant representative sample of the input data set**, and the training process must not lead to **overfitting**, where the manifold approximation is too focused on the training dataset versus being able to handle more general inputs. Network structure, activation functions, step size, and weight initialization will also influence training results.




**(a)** OK manifold approx. **(b)** Ideal loss metrics **(c)** Overfitted approx. **(d)** Overf. loss metrics

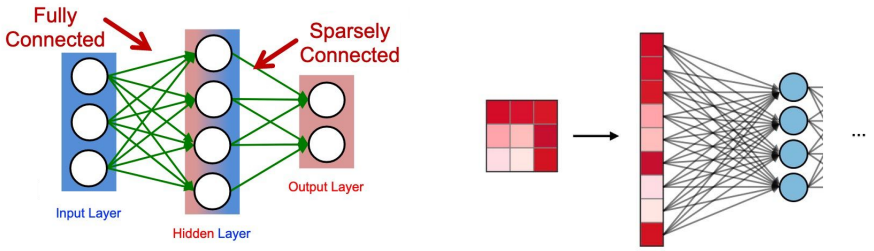
**Figure 2.13:** Overview of Learning Scenarios (Amidi and Amidi [2020])

### 2.2.4 Deep Learning and Modern Vision-Focused Network Structures

While NNs allow for a different paradigm in problem solving, and push efforts from the previous approach of breaking down a task and building up a solution through conventional algorithms (whether in a bottom-up or top-down manner), this type of Bio-inspired design brings an entirely different set of considerations. What type of neural network architecture is the most optimal (how many neurons are needed, and how are they linked between layers)? What activation function must one use? What should be the IO structure? What training algorithm should be used? And how to test performance and manage the training set? Most of these questions will not be directly explored in this thesis, except for small briefings of particular chosen vision model architectures.

For the purposes of this exploration, it would be useful to clarify a few other terms in the field. DL is a modern specialization of ML, focused on NNs with an unbounded number of hidden layers of bounded size (width), Deep Neural Networks (DNNs), with the goal of optimizing practical creation, network optimization, and inference (the process of computing an output for NNs) while still retaining the theoretical underpinning stated in the **Universal Approximation theorem** for the "vanilla" ANN. These layers can also be heterogeneous, and can deviate from the biological models, for the sake of the previously stated goals.

Within Vision, the most relevant deep neural network architectures are what we call CNNs and, in more recent research - Transformers.




**(a)** FC vs SC layers **(b)** Flattened image FC layer

**Figure 2.14:** Neural Networks (NNs) structures (Sze et al. [2017], Amidi and Amidi [2020])

This is mainly due to having to deal with high dimensional input data (images/videos). In a simple Feed Forward Neural Network (FFNN) the layers are termed **fully-connected** (Figure 2.14a). This means that each neuron in a certain layer has a connection (with its corresponding weight, etc) to **every neuron of the previous layer**. This leads to problems when dealing with more advanced vision task inputs, due to poor scalability related to the increasing number of parameters that need to be estimated by the network (due to the input image having to be "flattened" into input pixels which are assigned to a neuron, Figure 2.14b). There

are other issues aswell, for instance how a FFNN architecture does not take into account the spatial manifold structure of the image inputs, i.e., it treats input pixels which are far apart and close together on exactly the same footing. Spacial structure must then instead be inferred through training. Given this, other (deep) network architectures have prevailed in vision.

### *Convolutional Neural Networks (CNNs)*

CNNs were developed (Lecun et al. [1998]) to take advantage of the spatial structure inherent in image data, are faster to train and are particularly well-adapted for image recognition tasks. CNNs have 3 major concepts underlying their design: **local receptive fields**, **shared weights**, and **pooling** (Figure 2.15, Nielsen [2015]).

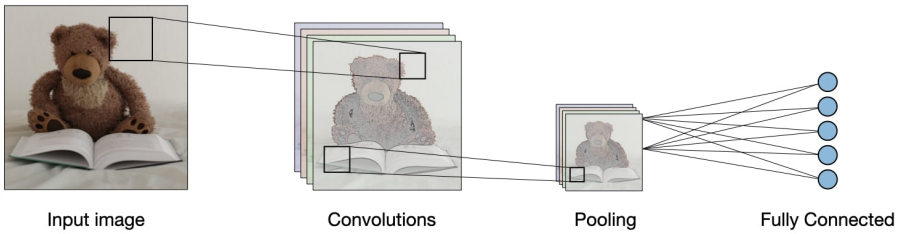
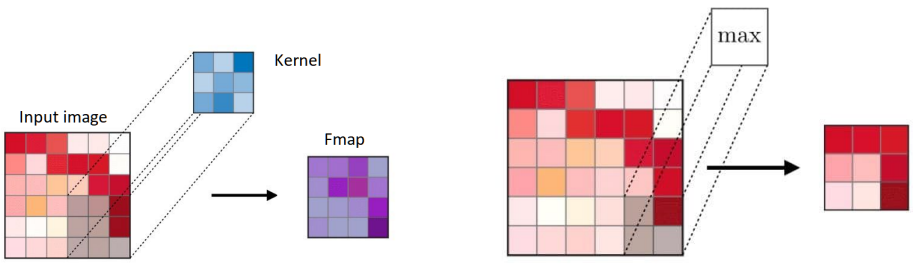



**Figure 2.15:** Summary of Convolutional Neural Networks (CNNs) structure (Amidi and Amidi [2020])

Instead of connecting every pixel from the input layer to the first hidden layer, only small, localized regions of the input image are connected. More precisely, each neuron in the first hidden layer will be connected to a small region of the input neurons, called the Local Receptive Field (LRF) for the hidden neuron. This can be thought as a sliding window on the image. So, each hidden neuron "learns to analyze" its particular LRF. Each hidden neuron will also apply a convolution (analogous to a weighted moving sum) operation on its inputs. Then this LRF can be slid across the entire input image. For each LRF, there is a different hidden neuron in the hidden layer.

Another important part is the **shared weights and biases**. All the hidden neurons in that layer will have the same weights and biases. i.e., all the neurons detect the same "feature" just at a different location of the input image, making it well adapted to translation invariance of features in images. The "map" from the input layer to the hidden layer is termed a feature map, with the shared weights and bias of a feature map defining a **"kernel" of "filter"**. A complete convolutional layer consists of several different feature maps. This kind of "sliding window" compressing structure with shared information is what reduces the parameter size (Figure 2.16a).




**(a)** Convolutional layer **(b)** Max pooling layer

**Figure 2.16:** Convolutional and Max pooling steps (Amidi and Amidi [2020])

Finally, we have **Pooling** layers. Usually used immediately after convolutional layers, they simplify the information from the convolutional layer, resulting in **condensed feature maps** - e.g. max-pooling where, for each feature map, it outputs a smaller one, using the maximum activation in a specific sub-region of the input feature map (Figure 2.16b).

### *Transformers*

More recently, due to the successes of the **transformer** architecture in the Natural Language Processing (NLP) perception domain (Brown et al. [2020]), some research (Ranftl et al. [2021]) has started into adapting these architectures for the backbone of dense prediction vision tasks (pixel level labelling, e.g. semantic and instance segmentation), instead of CNNs.

A brief description of a proposed Vision Transformer (ViT) ((Figure 2.17), Dosovitskiy et al. [2020]) will follow. Being one of the termed "Self-attention-based" architectures, transformers are usually "pre-trained" on a large core dataset and then fine-tuned on a smaller task-specific one.

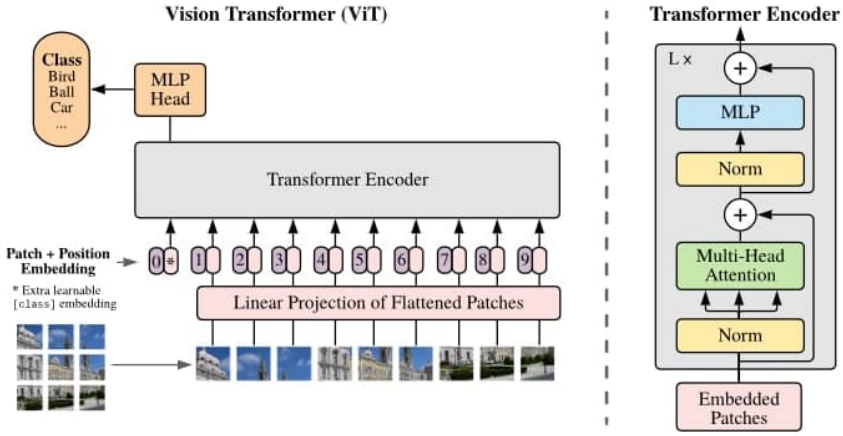



**Figure 2.17:** Overview of Vision Transformer (ViT) Architecture (Dosovitskiy et al. [2020])

Transformers, in general, have two main components: An **encoder**, and a **decoder** block. Inside these blocks we can find an (or a series of) **attention** and FFNN layers. The self-attention layer helps the block look at other relevant "words" in the input as it encodes a specific word. In essence, in the **encoder**, the attention-mechanism will, for every input "word" take into account several other inputs at the same time while deciding which ones are important by attributing different weights to those inputs. The **decoder** will then take as input the encoded sentence and the weights provided by the attention-mechanism (Luong et al. [2015]). In ViT ((Figure 2.17), Dosovitskiy et al. [2020]), the "head" FFNN (aka Multi-Layer Perceptron (MLP)) converts the encoded sequence into a sequence of class detections and no decoder block is used.

Loosely related to the operation mode of the previously mentioned (BOV) Sivic and Zisserman [2003] method, a visual transformer network operates on a bag-of-words representation of the image, with Image "sections" taking the role of "words", being embedded individually into a feature space, i.e. , these "words" being deep feature descriptors extracted directly from the image. These embedded "words" can be also termed as **tokens**. Transformers transform the set of obtained tokens using sequential blocks of Multi-Headed Self-Attention (MHSA), which relate tokens with each other in a manner analogous to the one previously mentioned for detecting

patterns and relationships between features.

In the case of ViT, the image is split into fixed-size patches, which are then linearly embedded together with position embeddings (thus encoding the visual structure of the image in the network), with the resulting sequence of vectors then being fed to a standard Transformer encoder. In order to perform classification, an extra learnable "classification token" is added to the sequence.

At this point, there is enough information to analyse the structure/architecture of the models that will be used, and general detail as to how they work. However, to evaluate the performance of these models, and how they compare between themselves, a review of commonly used datasets and performance metrics for the tasks mentioned in Section 2.2.1 is needed.

### 2.2.5 Performance Metrics, Indicators and Datasets

#### *Datasets*

To compare different networks on a particular domain, they need have equal starting points. This is achieved through standardized datasets upon which all networks can benchmark against. These datasets are curated for specific problems.

For general image tasks, we have **ImageNet**, a large-scale hierarchical annotated image database for a lot of object detection and image classification tasks. In **object detection** we also have **PASCAL VOC**.

**Figure 2.18:** ImageNet Dataset

For 2D HPE, there are two major datasets currently in use: **COCO** (Lin et al. [2014]), for large-scale object detection, segmentation, key-point detection, and captioning; and **MPII** (Andriluka et al. [2014]), for HPE. Different datasets have slightly different keypoint indexes/output formats. These vision tasks have brought the need for new metrics for evaluating performance. A summary of the most relevant for the project are as follows:

#### *Metrics for Segmentation*

There are two main performance domains for analyzing image segmentation problems, segmentation accuracy and segmentation efficiency. **Segmentation accuracy** encompasses accurate localization and recognition of objects in images/frames, i.e. large variety of detectable objects, and the ability for the algorithm to accurately and consistently detect the same type of object, even given natural variations (recognizing humans of all kinds of traits). **Segmentation efficiency** refers to computational intensity of the segmentation algorithm.

Now, we will go over several common metrics, such that most of the literature present for these vision tasks can be evaluated properly.

For **image segmentation**, there are the following performance metrics (Guerrero [2015]):

**Per-pixel classification**:

$$\text{performance} = \frac{correct\_classified\_pixels}{total\_number\_of\_pixels} \tag{2.4}$$

**Per-class classification**:

$$\text{performance} = \frac{1}{C}\sum_{i=1}^{c}\frac{number\_of\_correct\_classified\_pixels\_class\_i}{total\_number\_of\_pixels\_class\_i} \tag{2.5}$$

**Overlapping (Jaccard index)**:

$$\text{performance (Intersection over the union)} = \frac{1}{C}\sum_{i=1}^{c}\frac{TP}{TP+FP+FN} \tag{2.6}$$

TP -True Positive, FP - False Positive, FN - False Negative

### *Metrics for Object Detection*

For object detection, most metrics are related to mean Average Precision (mAP), which focuses on measuring the accuracy of an object detector. Two terms are also important for this, **precision** measures the percentage of correct predictions. **Recall** measures how well all the positives are found. These metrics are the used to calculate Average Precision (AP) for the detection classes.

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{Recall} = \frac{TP}{TP+FN} \tag{2.7}$$

Intersection over Union (IoU) measures the overlap between 2 boundaries. That is used to measure how much the predicted object boundary overlaps with the ground truth boundary. Some datasets predefine an IoU threshold which classifies whether the prediction is a true or a false positive.

AP computes the average precision value for recall values between 0 and 1 . In literature, the following terms may also be found: **APS** which is AP of small objects, **APM** (AP of medium objects), and **APL** (AP of large objects).

### *Metrics for HPE*

Percentage of Correct Parts (PCP) **[PCP@0.5]** measures rate of limb detection. A limb is "detected" if the distance between the two predicted joint keypoint locations and the true limb joint locations is $\leq 0.5 \times (limb\_length)$. There is also the **PCPm** variation, which uses the mean ground-truth segment length over the entire dataset instead.

Its main problem is that shorter limbs will be evaluated more harshly than longer limbs.

Percentage of Correct Keypoints (PCK) **[PCKh@0.5, PCK@0.2]** measures the correctness of keypoint detection. A joint keypoint is considered correct if the distance between the predicted and the true joint is within a certain threshold T ( PCKh@0.5, T = 50% $head\_bone\_link$, PCK@0.2, T = $\leq 0.2 \times torso\_diameter$).

Percentage of Detected Joints (PDJ) **PDJ@0.2**, measures the correctness of joint detection. A detected joint is considered correct if the distance between the predicted and the true joint is within a certain fraction of the torso diameter (0.2 for PDJ@0.2).

For Object Keypoint Similarity (OKS) (metric used in the COCO dataset keypoints challenge), its formula is as follows:

$$\frac{\sum_i \exp\left(-d_i^2/2s^2k_i^2\right)\delta\left(v_i > 0\right)}{\sum_i \delta\left(v_i > 0\right)} \tag{2.8}$$

Where $d_i$ is the Euclidean distance between the detected keypoint and the ground truth, $v_i$ is the visibility flag of the ground truth, with $\delta$ (impulse function) being the function that outputs 1, given a detection flag $v_i$ if it is a labelled prediction (v=0: not labelled, v=1: labelled but not visible, and v=2: labelled and visible). $s$ is the object scale, and $k_i$ is a keypoint constant (which controls falloff).

OKS is similar to IoU from object detection. Its calculated from the distance between predicted and ground truth keypoints normalized for the person's scale.

## 2.3 STATE OF THE ART VISION MODELS

### 2.3.1 Object Detection

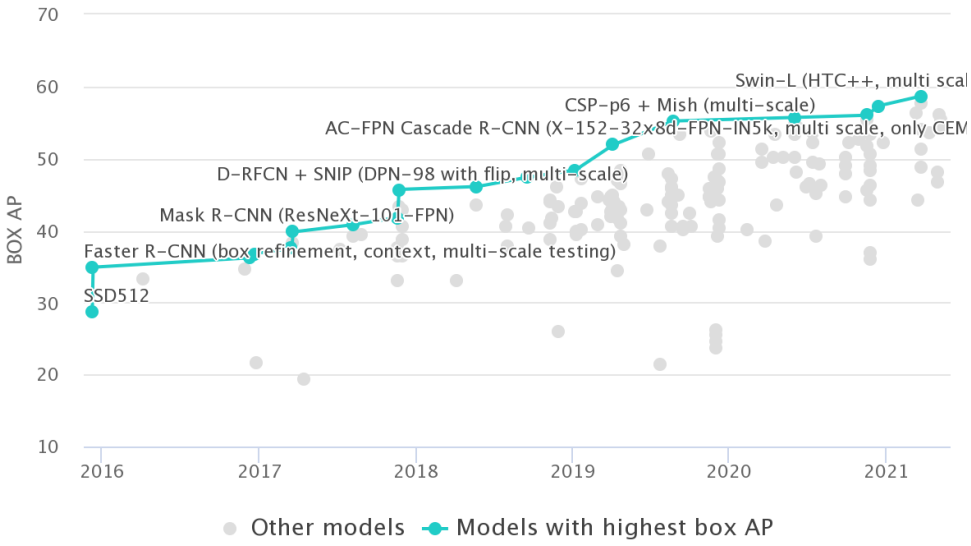



**Figure 2.19:** Object Detection performance over COCO dataset (Average Precision (AP)) (Facebook-AI-Research [2021])

Object detectors (Figure 2.19) can be divided into two main categories (Jiao et al. [2019]), **one and two-stage detectors** (Figure 2.20). A prominent example of a two-stage detector is Faster Region Based Convolutional Neural Network (R-CNN) (Ren et al. [2017]). For one-stage detectors, architectures such as YOLO (Redmon et al. [2016]) and SSD (Liu et al. [2016]) are common.

In Figure 2.20, yellow cubes denote series of convolutional layers (block), while blue cubes represent series of convolutional layers ( $>$ 1)(thick cubes), or Region of Interest (RoI) pooling layers which generate feature maps for objects of the same size (flattened cubes). In a two-stage detector, a region proposal network feeds region proposals to a classifier and a regressor, whilst a one-stage detector predicts bounding boxes from input images directly.

The main difference between these approaches is that one-stage detectors are optimized for high inference speeds (making them more suited for real-time tasks, ergo our research case), while two stage detectors have the goal of high localization and object recognition accuracy.

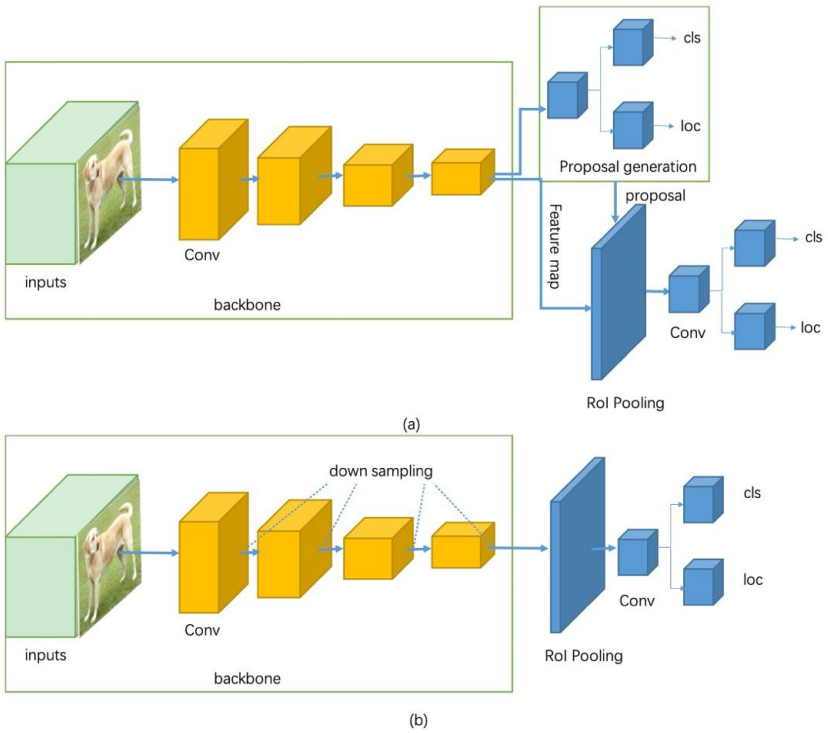



**Figure 2.20:** One (a) vs two-stage (b) detectors (Jiao et al. [2019])

Usually the two-stage detector's stages are connected through a RoI pooling layer. As can be seen for the architecture model over the years, quite a few RCNN based architectures have been presented over the years. Further detail will be given on the particular models in use by the system in Section 4.4.

### 2.3.2 Segmentation

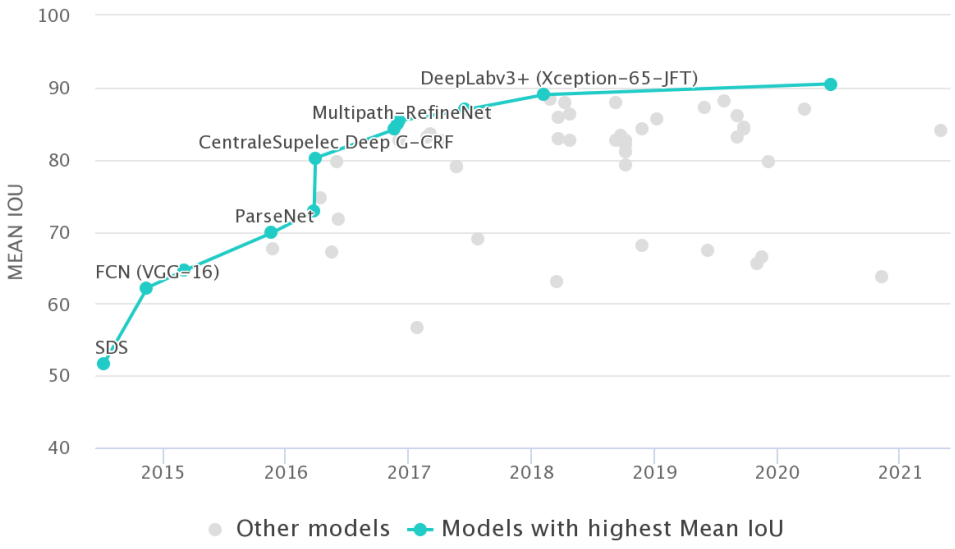



**Figure 2.21:** Segmentation Solutions performance (Intersection over Union (IoU)) over PASCAL VOQ (Facebook-AI-Research [2021])

There is a significant overlap between the network structures involved in segmentation (Figure 2.21), image classification and object detection. Notable Convolutional Neural Network (CNN) based Segmentation models, use DNNs detector architectures and then use features from the topmost CNN layer for object representation (Hafiz and Bhat [2020]). These techniques generally have the problem of detecting and segmenting objects at different scales (this is usually circumvented by applying the models to a "pyramid of images", each at a different scale (He et al. [2015]) ). Other problems that are common to these models include occlusion handling and image degradation, which is why most of the used datasets only include high quality images. One **focal evaluation** will be **how well these models handle real world data**.

In general, the underlying trend has been the increasing depth of these networks. Common CNN based segmentation techniques (Figure 2.22) include Mask R-CNN (He et al. [2020]) (based on R-CNN object detector, where it extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition). Inspired by the fast/faster R-CNN methods, the Fully Convolutional Network (FCN) predicts segmentation masks next to box regression and object classification.

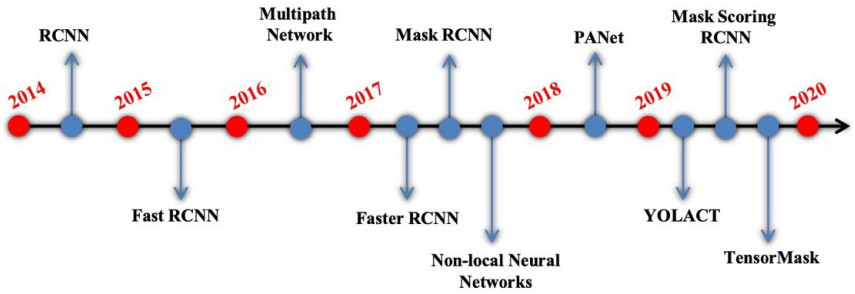



Figure 2.22: Timeline of Main Segmentation approaches in literature (Hafiz and Bhat [2020]

MultiPath Networks (Zagoruyko et al. [2016]), build on the standard Fast R-CNN with 3 improvements - "skip connections" incorporation which allows the object detector to access features of different network layers (thus creating a multipath information flow across the network); added elements that exploit context of objects at different resolutions, and the use of a loss function of integral nature. All of these lead to improved localization of the instance segmentation masks (Hafiz and Bhat [2020]).

There is also Deeplab (Chen et al. [2016]), which tackles semantic image segmentation using three main techniques - Atrous convolution (convolution with upsampled filters) to incorporate larger context without increasing the number of parameters or the amount of computation, Atrous Spatial Pyramid Pooling (ASSP) to robustly segment objects at multiple scales, and using a final fully connected CNN layer which improves localization performance. More specific detail on the chosen networks for the system will follow in Section 4.4.

### 2.3.3 Human Pose Estimation

Most explored solutions (Figure 2.23, & Figure 2.24) can be subdivided into two architectural approaches - bottom-up and top-down (Munea et al. [2020]).

With a bottom-up approach, the model detects every instance of a particular keypoint (e.g. all left hands) in a given image and then attempts to assemble groups of keypoints into skeletons for distinct objects (first localizing semantic entities, and then grouping them into person instances). A top-down approach is the inverse

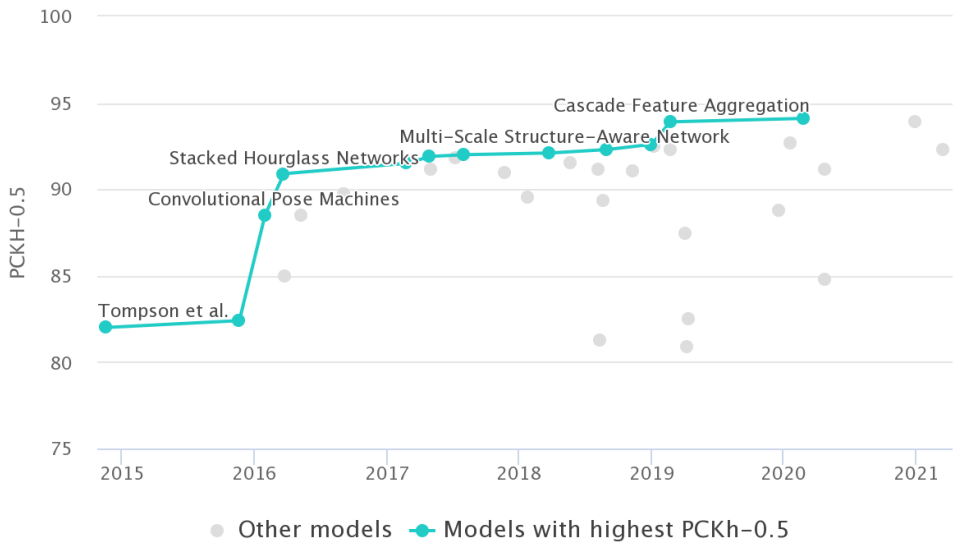



**Figure 2.23:** HPE performance over MPII dataset (Percentage of Correct Keypoints (PCK) (Facebook-AI-Research [2021])

– the network first uses an object detector to draw a bounding box around each instance of an object, and then estimates the keypoints within each cropped region (each individual person instance is localized using a bounding box, followed by the pose estimation the pose of each instance).

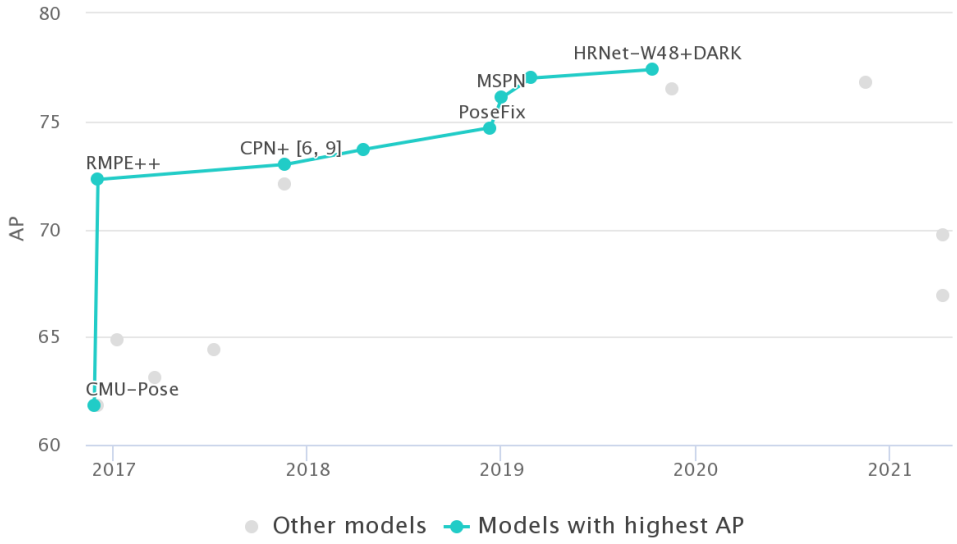



**Figure 2.24:** HPE performance over COCO dataset (Average Precision (AP)) (Facebook-AI-Research [2021])

DL architectures suitable for HPE are based on variations of CNNs. As alluded to previously, methods based on DL are able to learn powerful feature representations with various abstraction levels from images, transferring the problem of feature representation to the development of more performant network architectures and more optimized training procedures.

DL models for pose estimation come in a few varieties related to the top-down and bottom-up approaches discussed above. Most begin with an block that accepts an image as input and extracts features using a series of narrowing convolution blocks. What comes after this block depends on the method of pose estimation. The most

conceptually simple method uses a regressor (variable in a regression model used to predict a response variable) to output final predictions of each keypoint location. The resulting model accepts an image as input and outputs (X, Y, and Z, in case of 3D HPE, coordinates for each prediction keypoint). This normally requires more improvements for usable results.

More recent approaches use **Tranformer** based models, an Encoder-Decoder (EC) architecture. Instead of estimating keypoint coordinates directly, the encoder is fed into a decoder, which creates heatmaps representing the likelihood that a keypoint is found in a given region of an image.

During post-processing, the exact coordinates of a keypoint are found by selecting heatmap locations with the highest keypoint likelihood. In the case of multi-pose estimation, a heatmap may contain multiple areas of high keypoint likelihood (e.g.multiple right hands in an image). In these cases, additional post-processing is required to assign each area to a specific object instance.

Top-down approaches also use EC architectures to output heatmaps, but they contain an additional step, in which an object detection module is placed between the encoder and decoder and is used to retrieve regions of an image likely to contain an object. Keypoint heatmaps are then predicted individually for each box. Rather than having a single heatmap containing the likely location of all left hands in an image, a series of bounding boxes are retrieved that should each contain only a single keypoint of each type. This approach makes it easy to assign keypoints to specific instances without a lot of post-processing.

Commonly used architectures for HPE include Stacked-Hourglass networks (CNN based), Mask-RCNNs, and EC networks (PersonLab / PoseNet and OpenPose).

Pure EC networks take an image as IO heatmaps for each keypoint. Mask-RCNN predicts bounding boxes for objects in an image and then predicts poses within the regions of the image enclosed in the bounding boxes.

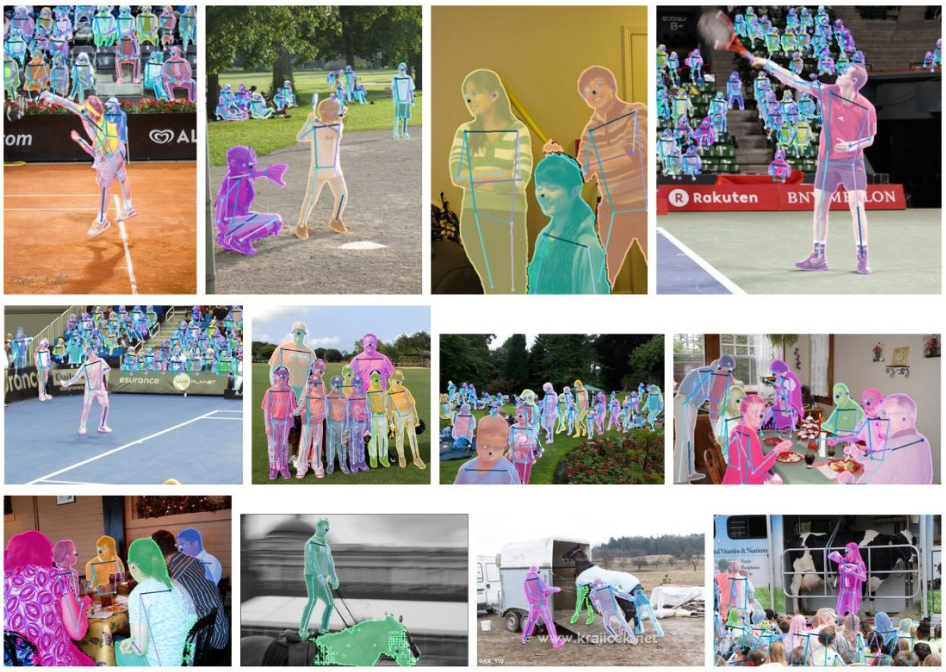



**Figure 2.25:** Personlab instance segmentation and pose estimation examples (Papandreou et al. [2018])

OpenPose and PersonLab (aka PoseNet) are variants of an EC architecture. In addition to outputting heatmaps, the model also provides heatmap refinements in the form of short, mid, and long-range offsets. Heatmaps identify regions of an

image in which a keypoint is likely to be found, while offsets guide towards to a more accurate final prediction within a region.

Convolutional pose machines build on the EC architectures by iteratively refining heatmap predictions using additional network and feature extraction layers. The final output is a single set of heatmaps, and post-processing involves identifying the pixels whose heatmap probability is the highest for each keypoint.

A broad overview of 2D HPE methods over the years can be seen in Figure 2.24, with a more in depth exploration of the used models in the Section 4.4 section.

## 2.4 VISION OVERVIEW

There are a few conclusions we must take from this review:

- Classical CV approaches pre-DL **were not robust enough** for applications in real life scenarios without extensive "tweak engineering";
- DL **allows** for the development of models that **achieve these tasks, with higher performance metrics**, given that the training process is exhaustive enough;
- There are several models for CV tasks, however they **tend to share common architectures and components**;
- These models have different focuses, whether more towards inference speed, or towards precision. While both are relevant, inference speed is more relevant for this project;
- New Transformer based models have **better precision performance, but require higher amounts of compute, and are generally only available for lower level stages of computer vision tasks**, which makes them, for now, not a focus for further development;

Given that **Transformers** are still in early stages of development, require more resources to train and use and are still at early stages of validation, and classical vision techniques under-perform compared to DL techniques, this project will use mostly CNNs, due to their midpoint accuracy performance, and faster inference speeds. This will be discussed further in Section 4.4 .

# 3 HUMAN ROBOT INTERACTION; BACKGROUND, AND RELATED WORK

Having explored the perception side of our problem, we need to develop the framework within which this information can be used, both to **extract useful information** for driving our interactions, as well as **how these can be driven** such that the system can work optimally. This is within the domain of Human-Robot Interaction.

## 3.1 BACKGROUND

Coming back to Chapter 1, one of the main research questions introduced was the problem of operator awareness in a robotic system (Section 1.4). The second component of this research point is how can a robotic system react productively to a human's actions? Such a system needs to model human feedback for robotic design (Noceti et al. [2020]). We will specifically focus on inferring human awareness through computer vision, a promising topic of HRI and CV research in recent years (Zheng et al. [2021]).

Revisiting the postulate 1.4 - that a vision-led approach can build an effective system for inferring operator intent, and react accordingly during a task; and referring back to the problem context and setup (Figure 1.1), we will start with theoretical underpinnings for HRI.

## 3.2 MODELLING HUMAN MOTION

In a scenario where a robot is deeply embedded in activities together with humans, the robot will require perceptual and reactive skills, for a seamless interaction with people. For understanding human partners (and be understood by them (Sandini and Sciutti [2018])) the **ability to detect, represent and recognize human dynamics to generate appropriate responses** is essential - the **task of understanding human motion**.

In humans, action/perception is deeply related to the parieto-premotor brain network, for both action execution and during the perception of those actions being performed by other agents. Activations in this neuron system have been associated with action anticipation and comprehension (Rizzolatti and Sinigaglia [2010]).

Studies by Johansson [1973a]; Attention et al. [2008] (Figure 3.1) explore this system of perception and the different levels at which an action can be represented, while introducing an hierarchical model of action representation (Figure 3.3), where actions can be described:

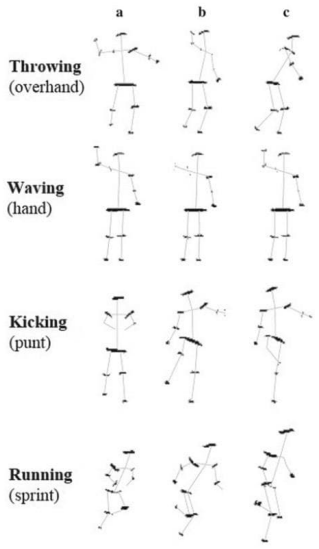



Figure 3.1: Human motion Awareness at beginning(a), middle (b) and end (c) stages of motion (Johansson [1973a])

- at the muscular level—encoding (pattern of muscular activity);
- at the kinematic level—encoding (spatio-temporal properties of the motion of the effector;
- at the level of goal—encoding (short-term aim of the action);
- at the level of intention—encoding (long-term purpose of the action)

However actions are sometimes ambiguous. Further studies (Amoruso et al. [2016]; Cretu et al. [2019]) explore contextual effect on action comprehension and helping with perceptual ambiguity and signaling which intentions are more likely to drive upcoming actions (Figure 3.2).

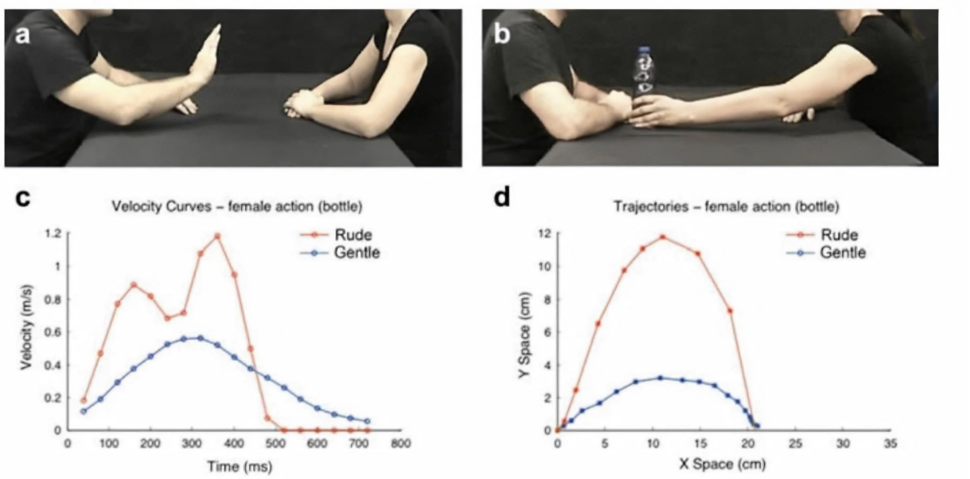



**Figure 3.2:** Bottle Passing with different kinematic profiles given differing "rudeness" (Di Cesare et al. [2014])

Human actions are multifaceted, containing both internal contexts (i.e. personality traits or experience), and external contexts (i.e. environmental constraints, etc) (Figure 3.3). Studying the way the Human Visual System (HVS) understands human motion (Johansson [1973b]; Hemeren and Thill [2011]) can help with designing vision systems to also do this task. From this research, two categories of critical visual features for biological motion perception can be distinguished: the global features (action concepts and categories), and local features (pertaining to specific movements).

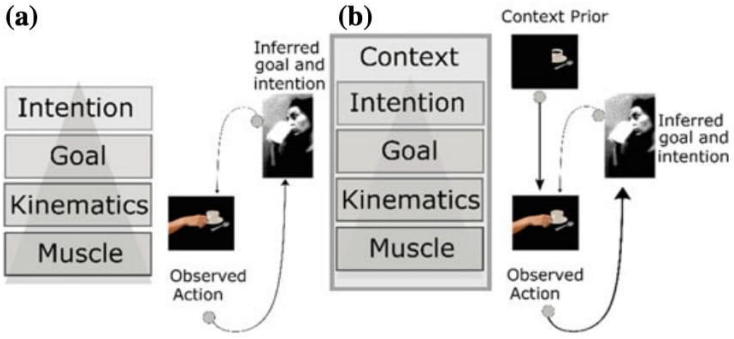



**Figure 3.3:** Hierarchical model of action representation (a). Extended model taking into account Contextual Priors (b) (Noceti et al. [2020])

### 3.2.1 Computational Motion Comprehension

Modeling and understanding human motion from visual data remains challenging, due to the variable dynamic information present in a visual scene, though DL has provided significant improvements to previous approaches (Asadi-Aghbolaghi et al. [2017]). Bio-mimetic approaches are a natural solution for devising computational perception models, referencing and being guided by neuroscience literature and mechanisms underlying human motion perception, some of which were briefly introduced in Section 3.2.

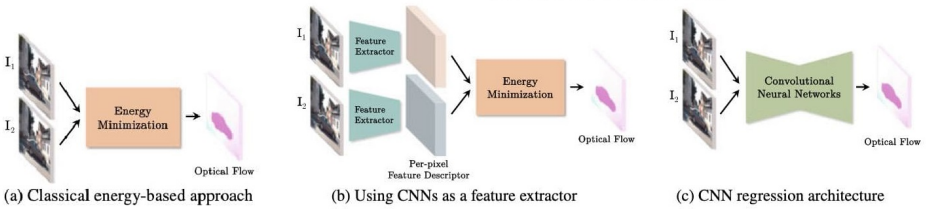



**Figure 3.4:** Three main Optical Flow Estimation Approaches in CV literature (Noceti et al. [2020])

Analysing the first levels of **motion analysis** can be reframed into a detection problem, with the aim to identify spatio-temporal image regions where the motion/object of interest is occurring. At its most abstract, such techniques include optical flow (Fortun et al. [2015], Figure 3.4), a method to obtain salient low-level motion information - an estimation of motion vectors in the image plane. These fields show strong connections with the behavior of human brain areas involved in the perception of motion.

Building on this, the ability to precisely identify when and where the motion is occurring "motor planning", requires detecting and segmenting actions (Lea et al. [2017]; Peng and Schmid [2016], Figure 3.5). These tasks are needed for automatic motion recognition systems.(Peng and Schmid [2016]; Gkioxari and Malik [2015]).

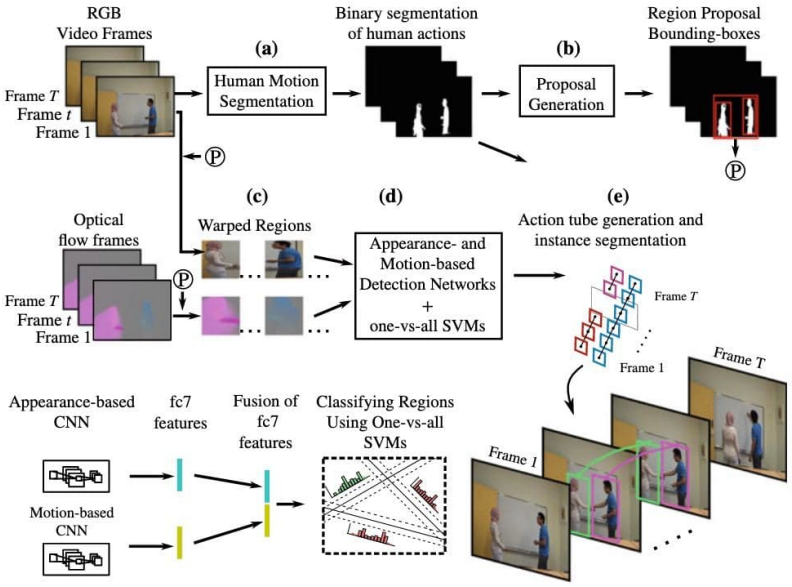



**Figure 3.5:** Spatio-Temporal action instance segmentation and detection pipeline. From video frames to human motion segmentation, ending in region proposal bounding boxes where both RGB and optical flow frames serve as inputs to respective appearance and motion-based detection networks (Noceti et al. [2020])

Focusing further on object manipulation actions, contact and motion become essential for encoding information about the actions themselves (Weinland et al. [2011]). Contact encodes where the object is touched or grasped, and the interaction duration. Motion conveys information on what part of the environment is involved in the interaction and how it moves (Aksoy et al. [2011]). Technique frameworks approaching this include imitation learning (Hussein et al. [2017]), in which robots acting in less controlled work-spaces require effective mechanisms for learning how to perform manipulation tasks (action observation and planning) using an attention mechanism for detecting moving objects and then understanding their motion from **RGB-d** sequences, by detecting actor-environment contact locations and time intervals, and segmenting the objects' motion while estimating their pose. Figure 3.6 demonstrates a possible system for action understanding from lower-level detection primitives encoded in the Knowledge Technology (KT) dataset.

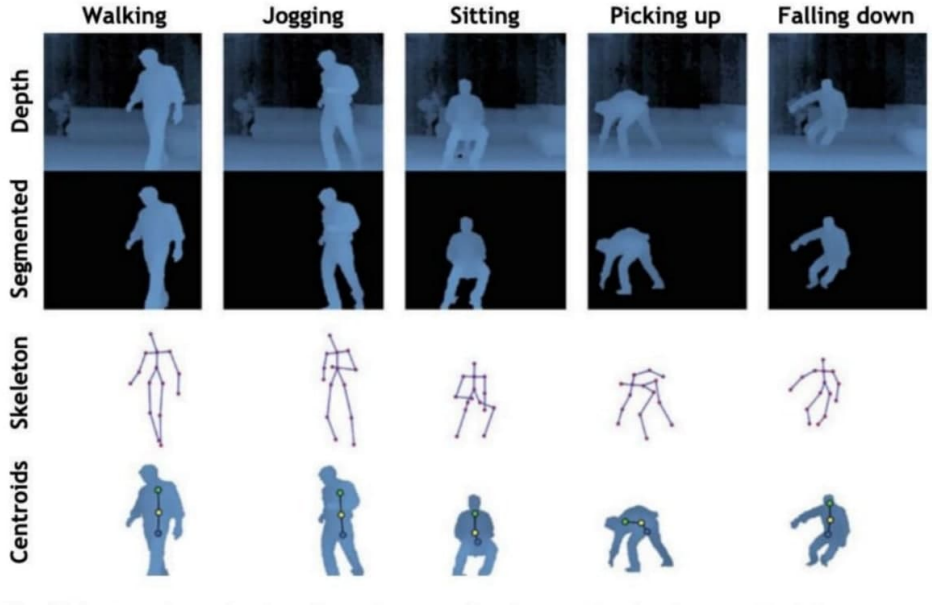



**Figure 3.6:** Actions from KT (Knowledge Technology) action Dataset, with depth buffers, segmentation, skeleton and centroid primitive stages (Bartneck et al. [2009])

Moreover, such systems must continuously acquire and fine-tune knowledge over time, what is termed as **Continual learning**, while not retrograding previous abilities (Chen and Liu [2016]). Again, bio-mimetic approaches come into play here (Nelson [2000]; Willshaw and Von Der Malsburg [1976]), e.g. hierarchical arrangements of variants of growing self-organizing networks (Marsland et al. [2002]). There are two classes of note in growing networks: the Grow-When-Required (GWR) model ( (i.e., neurons are grown and removed) updated in response to a time-varying input distribution), and **Gamma-GWR**, which extends the previous class with temporal context for efficient learning of visual representations (as motion is a temporally correlated input). The dynamic events considered can scope both short-term behaviors (such as daily life activities) and longer-term activities. However the former is more relevant for our task of Human Robot Co-Production (HRC).

One last point of study in HRI literature is the analysis of movement expressivity (whole-body motor component of emotion expression) and of emotional intelligence (Breazeal [2003]; Giraud et al. [2016]; Sherer [1984]). Also referred to as the dynamic movement component in affect perception, in contrast to the static form component (Kleinsmith and Bianchi-Berthouze [2013]). The main computational goal of such studies (Piana et al. [2016]; Varni et al. [2010]) is to devise computational solutions to replicate, on a machine, a humans' capacity for emotional aware-

ness and response, while integrating that information in communication/interaction. Movement expressivity analysis can span "sensing" human motion/emotion (Elfaramawy et al. [2017]; Lambert et al. [2019], Figure 3.7), to datasets of human expressive movements, or how such models can be employed in robotics scenarios.

Given all these priors, the correct development and deployment of models for how robots can use their movements to interact with their human partners encompasses the last goal of HRI. An interaction is composed of two aspects: movement as a form of communication (widely studied in the field of animation, Balit et al. [2018]; Bartneck et al. [2009]; Gray et al. [2010]), and movement as form of synchronization (to establish effective and intentional coordination (Lorenz et al. [2011]).

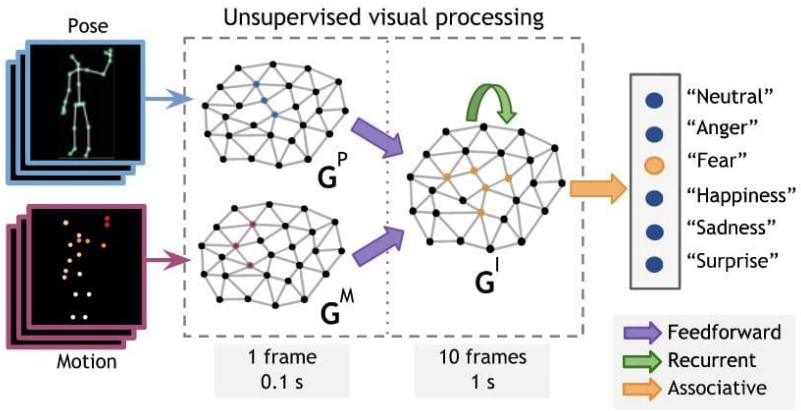



**Figure 3.7:** Learning Network Architecture for emotional insight given Pose and Motion data (Elfaramawy et al. [2017])

Communication is the "director" in the action–reaction paradigm. Strategies can depend on action and reaction expressed in the form of movements (gestures, navigation) or behaviors (conversation and dialog). This can involve strategies that measure the cognitive load on the partner, for opportune rhythm planning (Althaus et al. [2004]; Ciolek and Kendon [1980]). The robot can also use the same perception for its own skills, such as robot learning of manipulation tasks (Pairet et al. [2019]) and object grasp affordances (Ardon et al. [2019]).

Another important concept in robot motion is the **enactivism** concept (Varela et al. [1991]), the assumption that cognition arises through a dynamic interaction between an agent and the environment, so that changes in the dynamics of the environment generate perturbations in the dynamics of the robotic agent. As a dynamical system, self-regulative behaviors of the agent are necessary to aim to compensate environment perturbations and generate actions that react to these changes in order to maintain balance, so such robot systems can be accepted into mixed human–robot environments, as is our goal. Applications of this methodology include enactive robot assisted didactics (Mubin et al. [2013]; Tanaka et al. [2015]), where there is close interaction between the teacher and the student, with the robot assuming the role of tutor that mediates in the enactive didactics process between the student and the teacher, through nonverbal communication competencies based on movement. Finally, timing is an important field of research in motion understanding. Movement is characterized by several different parameters, in which timing is a key aspect that has to be encoded in robot behavior. The timing of the robot movement has a wide impact on the users' satisfaction (Darling [2017]; Kidd and Breazeal [2005]). Robotic movements can elicit a "priming" effect (impact that an entity's movement has on how the observer moves) (Kashi and Levy-Tzedek [2018]; Vannucci et al. [2019]).

All the different aspects explored so far are important for designing the future of human interaction, with the common goal of understanding humans, to build machines that are proficient at interacting with us (Sciutti et al. [2018]).

## 3.3 HUMAN ROBOT CO-PRODUCTION

Previously mentioned in Chapter 1, HRC is a sub-field of HRI, specifically focused on designing cooperative work scenarios between robots and humans. This is a multifaceted issue, but there are some core aspects that are explored in literature.

One of the first problems is in how to allocate and divide tasks between humans and robots (Figure 3.8). As such, the methods to solve **task allocation** (all tasks assigned to an entity, enforcing all order and precedence requirements, as well as workload balance), and **function allocation** (assigning a particular task/function to an entity) need to be explored (Krüger et al. [2009]; Inagaki [2001]).

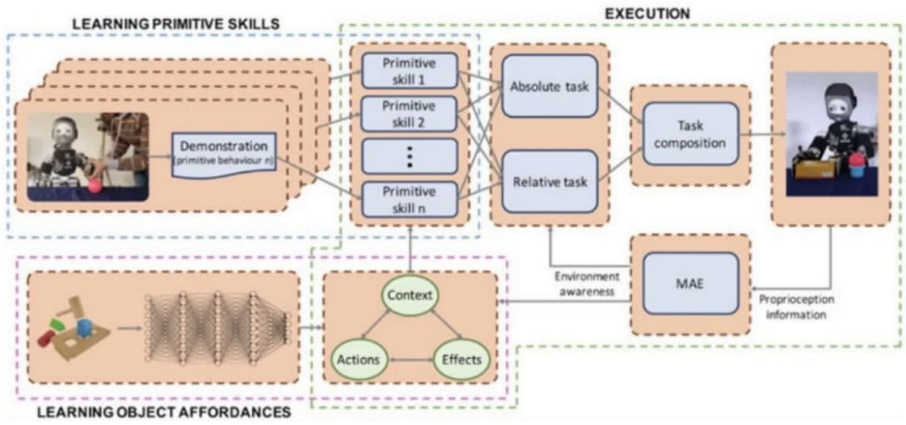
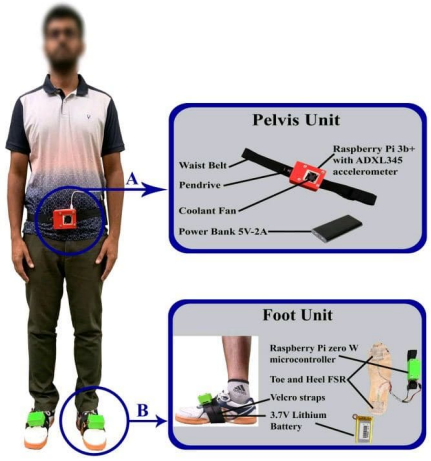



**Figure 3.8:** Overview of an Hierarchical Machine Learning Based Framework Structure for HRC (Human Robot Collaboration) (Noceti et al. [2020])

Such methods include **comparison allocation** - Humans are Better at Machines are Better at (HABA-MABA) approach, which assumes both humans and robots have certain domains in which proficiency is natural, and allocation should be guided by such a framework. We also have **leftover allocation**, which automates everything that is possible, allocating the rest to humans.Other approaches focus more on the cost benefit analysis between the task division. Finally, **Sharing and trading of control** approach considers different possible levels of automation and, for a given task, determines the appropriate level of automation. The literature also evaluates these methods, and how they are deployed (OLDER et al. [1997]).

HRI literature, also explores the classification (Malik and Bilberg [2019a]; Wang et al. [2019b]) of the amount of interaction between humans and robots, or how attention can be used as a model for intention in HRC tasks (Eldardeer et al. [2020]). This research project will not focus on task deployment/allocation, scheduling nor on higher-level system task design, but rather on using vision for implementing and controlling "proto" interactions, with the objective of instituting a lower-level software safety layer, demonstrating ways that CV can be used to enhance the in-built safety features in Cobots, allowing for more flexible and adaptable software systems that can track the human, which allows the design of interactions such that more trust is placed on the automated components by the human operators (Widder et al. [2021]; Colley et al. [2021]; Hancock [2011]). First we will review approaches in literature for building HRI/HRC systems.

## 3.4 ALTERNATIVE APPROACHES TO VISION IN HRI




**(a)** IDGR Pelvis (Calibration and Compute unit) + Foot Unit (Extra Sensing)

**(b)** HAVEP Smart Workwear, to keep track of Human Posture

**Figure 3.9:** Sensor Suit System Examples (HAVEP Workwear [2020]; Singh et al. [2019b])

Human Awareness has not been exclusively the domain of vision in literature, there are, of course, alternative approaches. One of the most common ones is the use of wearables/sensor suits that keep track of the Human joint states, and communicate through Internet of Things (IoT) platforms to the decision maker. An example of such a system is Intention Detection and Gait Recognition (IDGR) (Singh et al. [2019b], Figure 3.9a), which is a wearable system that mixes micro-controllers and sensors for gait correction and gesture detection, in real time, with a focus on keeping track of the user's gait, and providing real time suggested adjustments. Other related systems include Afzal [2015]; Redd and Bamberg [2012]. More commercial products are also available in industry, such as **HAVEP** (HAVEP Workwear [2020], Figure 3.9b)).

There are also non-vision systems designed specifically for cooperative tasks. These tend to focus on tracking the arms and hands, rather than general posture or intent. For example, research done by Musić et al. [2019] with the intent of allowing tele-operation of multi-robot systems with free-hand motions by the operator through the use of wearable haptic devices (Figure 3.10) - devices that are installed on the fingertips and provide haptic feedback on them. Such a system was then demonstrated on pick-and-place robot manipulation task.

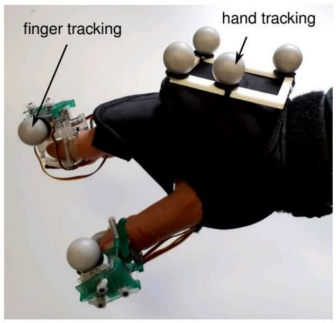



**Figure 3.10:** Example of wearable fingertip haptic device (Musić et al. [2019])

The common thread in all these approaches is the predominant hardware system design issue. All these systems tend to require the development/integration of several sensors and actuators, as well as the platform that integrates them. This overhead introduces other problems besides hardware complexity, as the increased number of components not only introduces error vectors, but also requires the need for the solutions to fit different body types, while having a calibration setup such that the designed systems still function within the performance requirements.

## 3.5 VISION-BASED APPROACHES

Vision however, remains mostly a software problem. Though there is some variance in the types of vision-based perception setups (Figure 2.1), the main problem set is in how to tackle all the previously mentioned Human Awareness tasks using solely pixels/voxels.

Starting on the lower levels of the perception stack, we begin by noting research into human tracking through Simultaneous Localization and Mapping (SLAM). SLAM is an ubiquitous technique in robotics, used generally in mapping tasks, but it can of course be focused in humans, and in the case of (Chau et al. [2019]), humans in indoor environments. This is achieved through human pose coupled with acoustic speech information, such that a microphone and camera equipped robot can track, map, and interact with humans. In this case the sound components estimate the Direction of Arrival (DoA) of active sound sources, which, together with human pose, provide enough relative human positions. Both the human and the objects that the human used during collaborative tasks are important for hybrid systems, so real-time **gazed object detection** is also an important topic. Yuguchi et al. [2019] (Figure 3.11) uses **RGB-d** inputs for this task.

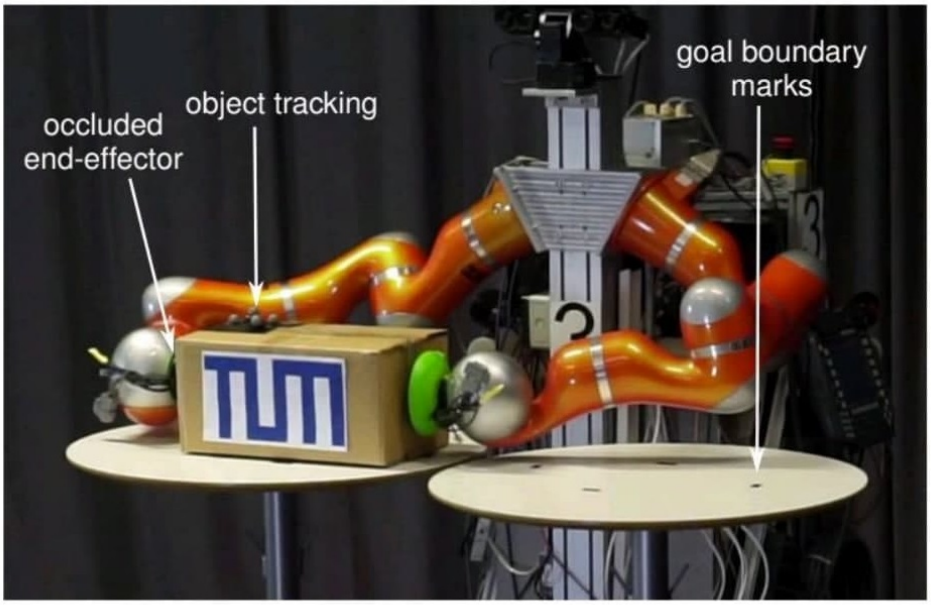
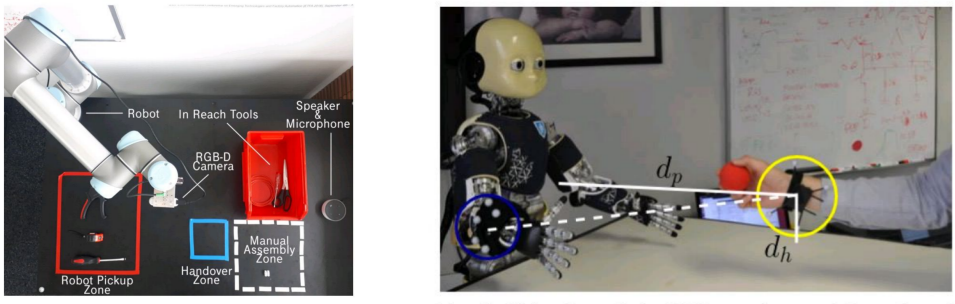



**Figure 3.11:** Experimental Setup for Grasping and Transportation of Object to Goals denoted with "Boundary Marks", with two robot Manipulators (Yuguchi et al. [2019])

In DL focused pipelines, research has also been developed on using this low-level information for motion prediction, e.g. **the MoGaze dataset** (Kratzer et al. [2021]), which is focused on full-body human manipulation tasks.
In a higher-level perception stage, accurate tracking is not sufficient, there is a need for deriving higher level abstractions, such as **intention and gesture prediction**, especially for cooperation tasks. There are several approaches to these, such as **BIL-SCNN** (Zhang et al. [2019]), a CNN based network that outputs intention from input -action visual sequences. There are also intention detectors based on the human gaze, who use an "Attention" concept (Fuchs and Belardinelli [2021]). More on the control aspect, **Gestures** are commonly used as inputs for robot systems. Kwan et al. [2020] explores such systems for handling Human-to-Robot Handovers.




**(a)** Collaborative Workplace Setup for handovers (Lambrecht and Nimpsch [2019])

**(b)** Experiment Focused around wrists and relative localization parameters extracted from the system (Duarte et al. [2019])

**Figure 3.12:** "Higher-Level" Collaborational HRI Interaction Research Examples

Integrating these sub-layers into specific common Human/robot workflows is, of course the end-goal of all this research. Such research applications are explored in, for example, by Lambrecht and Nimpsch [2019] (Figure 3.12a), where **RGB-d** inputs are used for flexible integration of Robotic Handover assistance in tasks directed by a human, or in Duarte et al. [2019] (Figure 3.12b), where human movements are studied for learning correct approaches to handle handover operations from the robot's perspective. Another example is the work from Raina et al. [2019], where vision is used as a control framework for a robot, through a path-planning and execution framework. This can also be explored in an completely integrated ML approach, as in Singh et al. [2019a], where Visual Servoing (VS) - guiding robot motion solely using visual feedback - is achieved with a Deep Reinforcement Learning (DRL) architecture.

## 3.6 DISCUSSION AND DIRECTION

In this chapter, the theoretical underpinnings of HRI systems were introduced, together with common approaches in literature, as well as specific uses of vision for enabling Hybrid Human/Robot systems. As seen over this chapter, there are different levels of "Awareness" that can be achieved - from tracking to high-level predictions. This thesis will focus on the early stages of awareness - detection and pose tracking, and building a system that demonstrates that even the lower-level perception stages are enough to drive **proto-interactions**, such as the safety cases mentioned in Chapter 1. These interactions are fundamental-level type of interactions in the sense that more complex processes can be built upon these simpler ones. This lower-level approach comes from two main reasons, first, it allows more focus on the perception problems - Tracking and Mapping visual feeds into a 3D reference frame, but also due to the fact that higher level systems are more abstract, and bring extra ambiguity, scheduling and task abstraction modelling problems. This will be undercut, with the demo tasks being **hard-coded**, using vision inputs and decision parameters from the perception nodes to parameterize their behaviour.

The first step for motion recognition (and later on human interaction) is human awareness, and tracking of a human's position over the frames (from which eventually motion techniques will be implemented, such as optical flow), together with gesture and intent models being deployed on top of that, which together with perhaps attention and emotional estimation components, will eventually feed into a Task system scheduler and allocator to deploy an interactive system for human robot cooperation.

Again, this thesis project will focus on the initial stages of such an HRI pipeline, utilizing the latest vision techniques for the initial stage of motion recognition, tracking people, objects and areas of interest across frames, and using these already to show how basic "proto-interactions" can be already devised from this early data, specifically with enhancing the embedded Cobot safety triggers through software based higher-level interactions, e.g. a predictive stop even before last moment "collision" trigger embedded in the cobot's hardware. The higher levels of motion recognition, such as intent, attention, etc, are left out of scope.

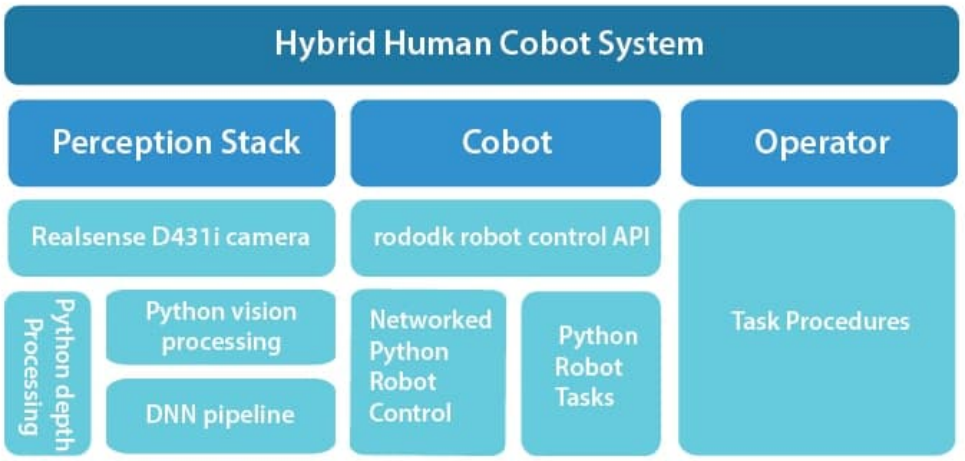
From the literature analysis, "proto-interactions" already contain key elements needed for the development of such HRC systems, and constitute a solid baseline for further work, implementing higher-level perception, cognition and response systems.

These demos will progressively use more complicated models, with safety stop being the most simplistic one, using a 3D mapped human location to stop a robot task, progressing into a more nuanced, accurate and developed human tracking model, being used to influence the Cobot's operation through the estimated human position, ending with Constant 3D tracking, where the human pose is tracked in real time, together with a mapping of this pose to the 3D environment.

# 4 SYSTEM; APPROACH

Having reviewed and discussed the fundamental building blocks needed for the system introduced in Chapter 1, this chapter will now delve further in the system design, chosen approach and implementation given this review.

## 4.1 FUNDAMENTAL PRECEPTS




As mentioned in Chapter 1 and Chapter 3, the system will be mainly focused on **vision** and triggering robot behaviour in a given task. As such, task modelling/scheduling and deployment will not be a focus of the project. These tasks will be implemented as hard-coded demos, instead of a fully agnostic system with a state manager/scheduler controlling task fulfillment.

Referring back to the initially proposed system (Figure 1.2) in Chapter 1, the software architecture of such a system must integrate these different components into a human reactive demo and establish a asynchronous networking interface, as certain sub-components must run simultaneously and, with the entire system being (soft) real-time, i.e. events should not deadlock the system, and its components should be free to run asynchronously where needed.

In addition, this system takes a modular software architecture approach. There are a few reasons for this:

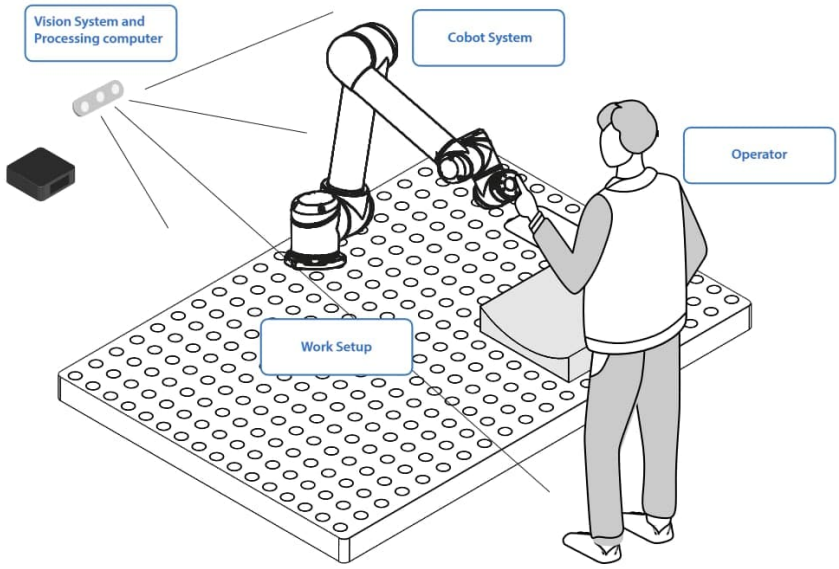
- Separating the code into modules complicates communication overhead, but also isolates the development of different modules, allowing for a more staged, iterative process in the development of the individual modules
- By starting development at the communication interfaces, and locking this at the beginning, the modules can be iterated while the general behavior of the system is maintained. (a later stage improvement to a module will not affect the whole system, as long as the communication interface's behaviour remains)

- The separation of the code in software modules allows for the replacement/adaptability of these components further on, preventing the system being locked to particular hardware.
- Under a fixed but modular structure, further pipeline improvements as well as the addition of new nodes are more streamlined, especially if the communication structure remains fixed.
- If the communication interface is abstracted to the Operating System (OS) level, the modules do not need to be developed in the same tooling/language/environment, as long as the communication is structured agnostically.

This main weakness in this approach is in the added communication complexity, and problem handling when the system is running asynchronously. How the implemented interface deals with these problems will be discussed further on in this chapter.

## 4.2 MAIN SYSTEM COMPONENTS

Introducing the demo setup again (idealized in Figure 1.1), there are a few main hardware components:




- Intel Realsense D431i Depth Camera - the main perception input, provides both RGB and depth frames, this particular camera choice will be explored in Section 4.4.3;
- Universal Robots (UR5 cobot) - collaborative robot which will be controlled through ethernet and the **robodk** python api;
- Intel NUC Computer - main computing device, ubuntu based system interfacing all the modules during the demos.
- Razorcore External GPU enclosure with Nvidia Geforce GTX 1070, the secondary computing device, offloads NNs inference.

## 4.3 SOFTWARE ARCHITECTURE

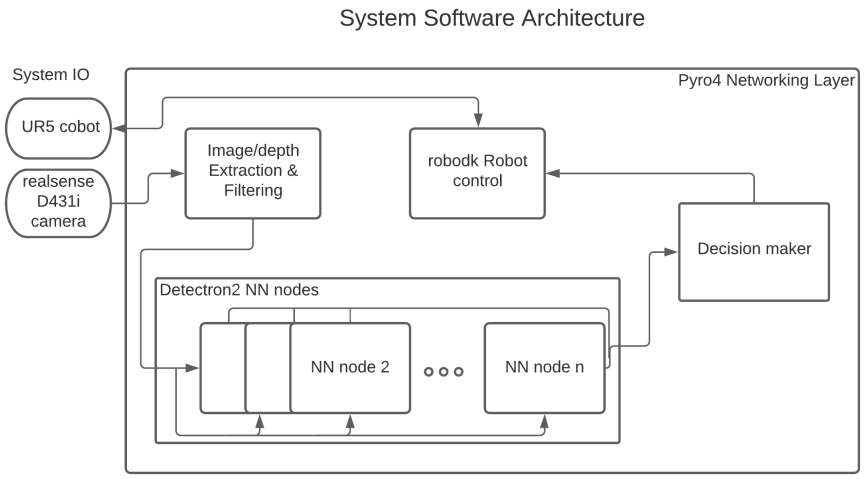



**Figure 4.1:** First Software Architecture Overview

Figure 4.1 presents an overview of the implemented software architecture. In Section 4.4, these components will be explored individually, in greater detail, both in their current version, as well as the previous iterations leading to it.

## 4.4 APPROACH

**Figure 4.2:** Actual Workspace used in the demos

The actual robot workspace can be seen in Figure 4.2, while Figure 4.1 introduces the software components. We start by describing the modular structure as well as the environment it runs in.

### 4.4.1 Modular System Architecture

The software system is composed of a series of python scripts (with small C++ components, mainly for realsense camera dependencies). All the python modules run inside a **virtual environment**, powered by **pipenv**. This is done so that the software modules run in an **isolated container from the Operating System (OS)**

**(ubuntu 20.0 LTS)**, which will help keeping the dependencies on track and stopping system updates from interfering with the modules' functionality.

### *Communication Interface Development*

A common tool for developing systems interfacing with hardware and robots is Robot Operating System (ROS) (Figure 4.3 , Stanford Artificial Intelligence Laboratory et al. [2018]). It assumes a modular node structure and embeds both extensive messaging functionality and other packages, from interfacing with certain hardware, to higher-level software Application Programming Interfaces (APIs). However this is not appropriate for our system, mainly due to the fact that the ROS system interferes with virtual environments, and effectively locks all python nodes to a single version (this is not useful for our system, as we might want the python module controlling the robot to be a different version than the one handling the NNs, for compability reasons. Additionally, the ROS robot control packages (Kavraki et al [2021], Coleman et al. [2014]) have compability issues related with the UR5 Cobots firmware drivers, which makes ROS less suitable for our use case.

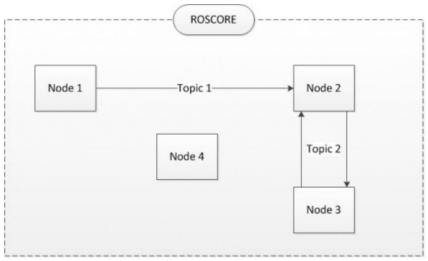



**Figure 4.3:** ROS Modular Structure and Communication

The initial approach for connecting the software modules was to use the UNIX networking pipeline, directly from python, with **pipes** and **sockets**. This was implemented in the first demo prototype, and was functional, however it brought some issues:

- using the core network features meant that the networking component would have to be added to every module, explicitly, as well as passing information between modules would always require custom built data serialization;
- pipe interfaces are not by default asynchronous, and handling exceptions or errors would also be necessary;
- while sockets solve the asynchronicity problem somewhat, the development overhead was still substantial, as well as problems involving data-passing or serialization remained (e.g.image buffers, tensors, etc)

Given these initial experiments, a different, higher level approach was taken, though one that would still be language/platform agnostic, uses the networking stack, while providing a more convenient programming interface.

As such, the main communication system will instead be based on python, more specifically through the Python Remote Objects (Pyro) library, which enables building applications in which objects can talk to each other over the network interface. There are several reasons for focusing on a modular architecture. First, this structure provides extra flexibility of deployment (different cameras, cobots, etc). It also allows the independent development of each software module so that working on a particular aspect of the system has a minimal impact on the rest. Finally there are clear advantages for future development, as detached modules are more easily upgraded/swapped-out without requiring refactoring work. All of these advantages come with the added asynchronicity, environment setup, and communication overhead costs in regards to a centralized system, but with the help of the following library, the trade-off makes sense in this particular case.

### *Pyro4*

Pyro builds on system Interprocess Communication (IPC) infrastructure, allowing python application objects' to communicate with each other/provide external access to their methods over the networking interface. Pyro takes care of locating the right object on the right computer to execute these methods. The project uses **Pyro4** to allow compability between all versions of python, allowing for modules to use both versions *2.x* and *3.x*.

Pyro presents some key advantages:

- It can use the in-built python data structures and data serializers (serpent, json, pickle, etc), discarding the need to create custom streaming/processing code for IPC;
- It is inter-operable between heterogeneous system architectures and OSes, relying only on the common networking interfaces;
- It is built on Agnostic Networking infrastructure - IPv4, IPv6 and Unix domain socket;
- It is designed to handle asynchronicity with added time-outs and other functionality for handling errors/corner-cases and setting up the right sequence of corrective actions.

However for correct functionality, an environment needs to be setup to properly enable the Pyro back-end to work as a **communication glue library to easily integrate various pars of a heterogeneous system**. This enables a "module" to call methods on remote objects (even in remote machines), illustrated in Figure 4.4.

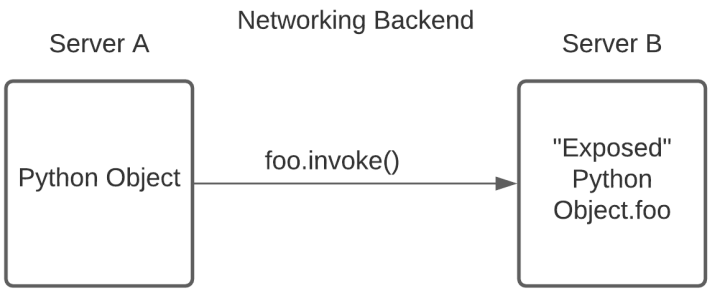



**Figure 4.4:** Accessing Remote Objects' Methods Through Pyro

This remote access is enabled through system networking (socket based) interfaces. Upon starting a project, a Pyro **daemon** is initialized, which will keep track of exposed function calls, intersect them in each module, and convey the requests through the networking interface to the other modules, and then the response to the correct caller. This emulates the Remote Procedure Calls paradigm in python. (similarly to java's Remote Procedure Invocation (RMI) APIs). For this to be achieved, every Pyro object has a uniquely identifying Uniform Resource Identifier (URI), which is registered by the daemon on a separate utility, that remains active throughout the system, the **Pyro name server**. This utility keeps a registry accessible to all Pyro daemons of all active (and accessible) registered objects.

### *Pyro4 project configuration*

For the demo project, this nameserver is one of the main factors in the boot sequence (Figure 4.5). First, the python environment is launched, creating a python

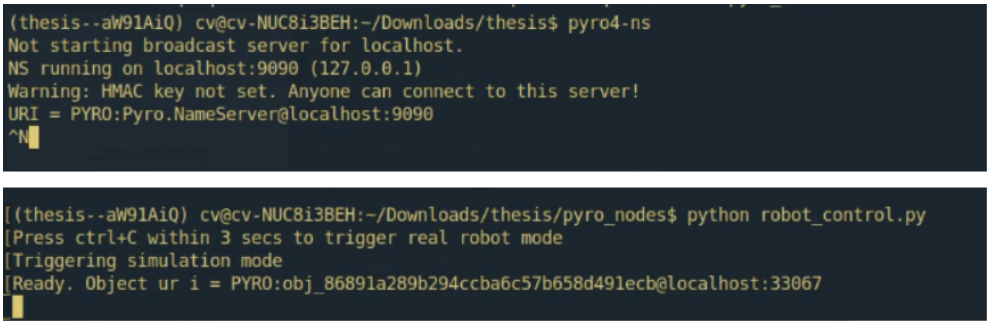



**Figure 4.5:** Example of **Pyro** system setup, where, inside the pipenv "thesis–aW91AiQ" environment, a local system pyro nameserver is created (top image), and the robot control script that, acessing that nameserver, "exposes" the unique RobotControl object to other pyro scripts.

container in the system, isolating possible dependencies that would interfere with the system. Then a Pyro nameserver is created, allowing for any script running in this environment to search for, identify and interact with active Pyro objects. For the project, one of these scripts, **RobotControl** creates a class that is responsible for interacting with the robot (of which only a single instance must exist at any given time), and registers a Proxy of this object in the nameserver. After this, any proxy enabled script can try to locate this object, if given the right proxy details. This can also occur in other devices, through networking, but for now the system is implemented on a single computing device (while still using the networking interfaces). In the demo's case, an example of such a system are the perception nodes, that need to alter the robot's behaviour given the vision data.

Currently the project runs with a single "nameserver" module, running continuously, and each functional model, (e.g.the robot control script) will "expose" certain methods to any running python program, that "hooks" to this module using the nameserver.

### 4.4.2 Robot Control and Proto-Interactions

To react to the perception input, we must control the Cobot in real-time. A Cobot is a robot that is designed to work around humans. As such, it has embedded triggers to stop in case it detects a collision (ostensibly with a human worker), as well as several other inbuilt primitive features for HRC. They come in varying configurations, from the more humanoid **Baxter series**, to the Cobot arms, such as the **UR5**, **UR10** (Figure 4.6), or the **Doosan M01013**.

The demo project will use the **UR5** CB2 Cobot arm. For this configuration, there are a few options that were explored for robot control, with the final chosen option being **robodk**. As alluded to previously, ROS (Figure 4.3 , Stanford Artificial Intelligence Laboratory et al. [2018]) has embedded robot control packages (Kavraki et al [2021], Coleman et al. [2014]). However these libraries have compability issues related with the UR5 firmware driver used in the experimental demo setup, more specifically, ROS enforces the use of certain ubuntu versions and python installations for the moveit module to be compatible with the **UR5** CB2 firmware driver (**Polyscope** version 3.5). This ROS driver (ur modern driver for ROS) would bring extra complications to the compute configuration of the system, especially given the additions in Section 4.4.3. As such this approach was sidelined.

Another control option was **PythonURX**, a python library designed to control the **UR5** Cobot using the networking stack through ethernet (as all the other interfacing libraries do). While decent for robot control, **PythonURX** does not have extensive simulation/visualization tools, which make system testing and monitoring more complicated.

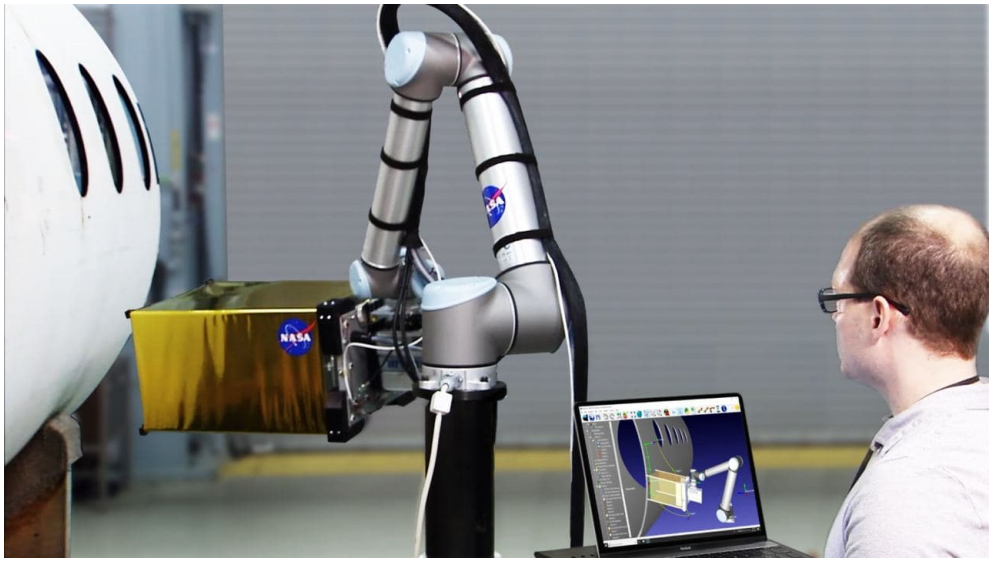


**Figure 4.6:** Deployed RoboDK Setup on UR10 Cobot

**robodk** (CoRo Laboratory [2015], Figure 4.6) provides several advantages in regards to the other options. First of all, it provides a powerful control interface, the option of setting up virtual robot configurations for several robots with minimum driver configuration, as well as several code APIs for real-time control. It also allows for setting other environment variables, such as the coordinate frame (the frame of reference that is used by the robot to interact with the world) and well as others (such as switching the type of movement and its attributes in real time). All this while providing a real-time visualization tool.

For the demo, a **robodk** "workstation" was created, replicating the **UR5** location in the workbench, with the camera position and robot base as reference frames (the full reasons for this will be further explained in Section 4.4.3).

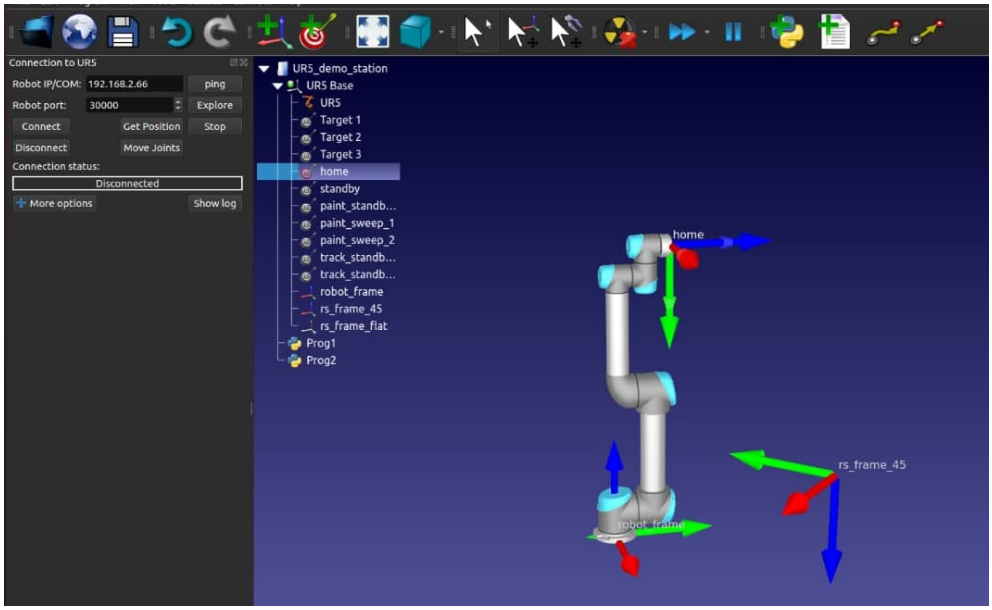



**Figure 4.7:** UR5 robodk "Station", with Pre-Configured Coordinate Frames, for the Base, Tool End-point and Camera Frames

The main python module interacting with the **robodk** python **API** is the **RobotControl** object. It is responsible for activating the robot (whether in simulation

mode or real life mode), and providing triggers for the "simulation tasks". These tasks are pre-implemented, but require an external module to activate, or interrupt them. **RobotControl** then behaves as a high-level controller that handles all the direct interfacing tasks, but leaves the high-level decisions on how to start/stop a certain pre-built task to an external python script.

**RobotControl** runs in two implemented main modes : **Simulation** and **Real-Life**. Simulation mode interacts with the **robodk** APIs and controls only the virtual robot setup (shown in the robodk GUI (Figure 4.7)), while **Real-Life** mode does not only that, but also translates the high level python code into Transmission Control Protocol (TCP) commands that the UR5 firmware can understand, controlling the robot through the networking interface, with Internet Protocol (IP) through an ethernet connection.

Given a fully setup workspace mimicking the workbench containing the real robot, the following sections describe the implemented demos (mentioned previously in Chapter 3), together with the demo code structure.

### *Task 1: Safety Stop*

Safety Stop is the simplest, most "coarse-grained" interaction that can be implemented. In the algorithm's flow, for every received frame, for the selected human bounding box, a "mask" is applied to the depth buffer, giving a mean distance to the detection. This value is then converted to the distance from the perspective of the robot's frame. If this value is less than 1.5 meters, the **pyro robot proxy** is invoked, triggering a safety shutdown of the cobot.

**Algorithm 4.1:** SAFETY STOP (*Robot_IP*, *RS_Pipeline*)

```
1  RobotControl node
2    Input: stop_safety_signal, Robot_IP
3    Output: safety_signal_remote_callback
4     Connect to Robot through IP;
5     Expose Robot Control through Pyro Proxy;
6       while NOT received_safety_signal do
7           Trigger Painting Sequence;
8       Engage Robot Brakes;
9  Perception Node
10   Input: RGBd_frame, robot_control_proxy
11   Output: safety_stop_signal
12    Connect to RS_Pipeline;
13      while RGBd_frame is available do
14          Preprocess RGB frame;
15          Run model inference;
16          if person_detected then
17              retrieve depth values at tensor predictions;
18              use bounding mask to get average distance to camera;
19              project from camera_ref_frame to robot_base_frame;
20              if person_distance < 1.5m then
21                  trigger safety_stop_signal in robot_control_proxy;
```

### *Task 2: Safety Slowdown*

Safety Slowdown builds on the previous demo by implementing a so called "animation curve" (mentioned in Section 3.2.1), implementing a more "reactive" proto-interaction. In the algorithm's flow, for every received frame, for the selected human bounding box, a "mask" is applied to the depth buffer, giving a mean distance to the detection. This value is then converted to the distance from the perspective of the robot's frame. However, this value is not used for a thresholding operation, it is instead mapped into a variable which is then used as a live-parameter to update the state of **RobotControl**'s movements. The mapping curve choice as well as its efficacy will be explored further in the results chapter (Section 5.5).

**Algorithm 4.2:** SAFETY SLOWDOWN ($w$, $\alpha$)

```
1  RobotControl node
2    Input: joint_velocity, Robot_IP
3    Output: ∅

4     Connect to Robot through IP;
5     Expose Robot Control through Pyro Proxy;
6       while TRUE do
7           Trigger Painting Sequence(joint_velocity);

8  Perception Node
9    Input: RGBd_frame, robot_control_proxy
10   Output: joint_velocity

11    Connect to RS_Pipeline;
12      while RGBd_frame is available do
13          Preprocess RGB frame;
14          Run model inference;
15          if person_detected then
16              retrieve depth values at tensor predictions;
17              use bounding mask to get average distance to camera;
18              project from camera_ref_frame to robot_base_frame;
19              Map computed distance to velocity;
20              trigger update joint_velocity in robot proxy;
```

### *Task 3: Safety Track*

Safety Track (Algorithm 4.3) is a slight variation on both Algorithm 4.1, and Algorithm 4.2. Instead of using just the bounding box and segmentation masks, safety track uses a keypoint's tensor prediction allowing for these proto-interaction to react to a specific 3D location of a sub-part of the human, e.g. the head or an hand. Given a keypoint tensor 2D location, a small kernel around this location is used to retrieve a depth value for this keypoint, at which point it can be converted to a 3D point from the camera's perspective. This reference frame is then converted into the robot's reference frame, so that there is a 3D mapping of a particular keypoint. As such the only (major) change is in the inference stage:

**Algorithm 4.3:** SAFETY TRACK ($w$, $\alpha$)

```
...
for instance_detected (person) do
    select operator in focus(1st by default);
    retrieve depth values per tensor prediction keypoint;
    deproject (tensor, depth) into 3D points;
    select used keypoint;
    project from camera_ref_frame to robot_base_frame;
    Map computed distance to velocity;
    trigger update joint_velocity in robot proxy;
...
```

Given these demo algorithm specifications, the next section approaches the selected perception approach, to be able to realize the proposed demos.

### 4.4.3 Perception

Perception is the driver of the "proto-reactions" introduced in Section 4.4.2. In Chapter 3 the concept of "awareness" was refocused on modelling human motion, through initially tracking the human in the robot environment. There are two main challenges in this task:

- Detecting a person from a camera feed
- Tracking that detection in the robot's 3D environment

The perception approach will tackle these two challenges separately, and then merge the results into a coherent "map" of the environment and operator, to drive the tasks that were embedded in **RobotControl**. This task has similarities with *Structure from motion* (Ullman [1979]) from classic Computer Vision (CV), a technique which estimates three-dimensional structures from two-dimensional image sequences.

#### *Camera Setup and RGB-d frame Acquisition*

As mentioned on Section 4.2, there is a single visual input for the perception stack - an Intel Realsense D431i camera. This camera is what is termed a **RGB-d** sensor. This is because it provides not only a RGB colour image frame, but also a "depth" frame, thus being a richer source of data than the classic monocular RBG camera. It achieves this by integrating several sensors in one package, as seen in Figure 4.8. This depth frame is obtained through the use of **stereo techniques** (mixing views from two cameras in fixed positions to extract 3D information from 2D sources, much like the HVS).

In this assembly (Figure 4.8), the Infra-Red (IR) projector is added to help handling depth estimation when the realsense encounters low texture scenes, (large white objects, etc), by covering such scenes with pattern of dots of IR light (this technique is termed *active stereoscopic depth sensing*.

The reason this setup was chosen was influenced by two motives:

- Current State of the Art 3D-HPE models do not have matching accuracy to 2D models, and require far more compute;
- Nevertheless, the depth frame given by these 3D models is only a relative z-buffer (depth), so it also requires external calibration for location estimation.


**Figure 4.8:** Realsense D431i camera assembly. Left to Right in image: R IR sensor, IR projector, L IR sensor, RGB sensor

Given this, merging a purely 2D pose together with a depth frame is a compelling compromise. The realsense camera not only provides a cost efficient (Figure 2.1) setup, but also APIs for the depth frame and comes pre-calibrated, simplifying the process of relating the relative depth buffer into actual distance estimates. It possesses a wide angle of view, as well as being optimized for properly capturing dynamic scenes and having very configurable capture parameters.

To retrieve the depth buffer from the camera, Stereoscopic Vision uses two view ports (Figure 4.9) of the same scene, and calculates depth by tracking **the same keypoints across both left and right images** and estimating the disparities (difference between the coordinates in both images concerning the same keypoint in the world) between them (Longuet-Higgins [1981]). [1]. An example of an approach for this approach is the naive Sum of Squared Differences (SSD) block-matching algorithm (Li et al. [2009]), which uses a statistical method to calculate a mean disparity value around a certain area around each "pixel". The default realsense factory calibration is used. Any camera has two parameters **extrinsics** and **intrinsics**. The **intrinsinc** parameters relate to to fact that any real camera is not a perfect model of the ideal "pinhole" camera. The imperfections of the construction of a particular device are encoded in these. The **extrinsic** parameters relate to the embedding of the vision system in the 3D environment. Any estimates are given from the camera reference frame, **extrinsic** parameters allow the encoding of the data to another reference frame, for example the world reference frame in question.

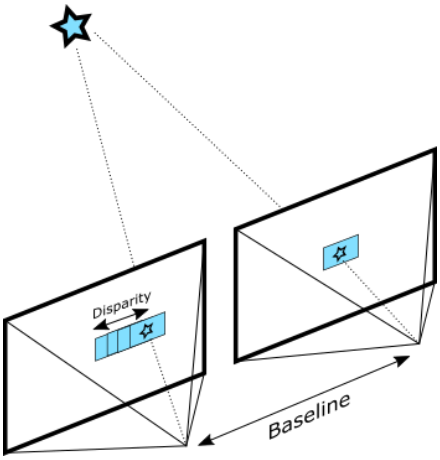



**Figure 4.9:** Depth Disparity Calculation Schematic

1 This projection transform of a 3D point on two image planes from the views and all the parameters of the stereo imaging system, is termed *epipolar geometry* (Xu and Zhang [2013]) in CV literature

Given the fixed position of the realsense given the robot frame, and the depth information from the camera perspective, the demos can merge this information to influence the "proto-Interaction tasks".

### *Realsense module*

Given the previous information, the realsense module interacts with the realsense camera through the **pyrealsense** python API. This module is encapsulated in an object that opens the depth and RGB streams, aligns these into a single RGB-d image, and stores these framesets. It also encapsulates the access to the camera ex/intrinsics and provides a method that estimates the depth in meters from these (this depth has a stated Z error of 2%, according to realsense (Intel Corporation [2021]).

The depth frame does not need to be left unaltered. Due to its method's inherent noisiness, there are post-processing filters available that make a more tractable depth buffer (Grunnet-Jepsen and Tong [2020], Figure 4.10). This will, of course come with extra processing tradeoffs both in performance and resulting artifacts.

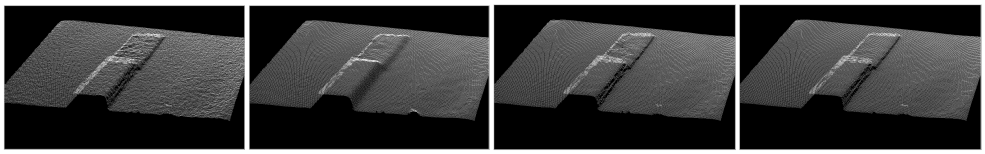


**Figure 4.10:** Depth filters, Left to Right: Raw depth buffer, spatial smoothing (hole-filling) filters with different smoothing constraints, and both temporal and spatial filters (Grunnet-Jepsen and Tong [2020])

Beyond the depth buffer, **lib_realsense** also provides the **deproject** functionality, for converting a 2D pixel location and a depth value $(u, v, d)$ into a 3D point$(x, y, z)$. This is achieved through the camera intrinsic (pre-calibrated) and preset parameters (s = depth scale, $f_{x,y}$ for horizontal and vertical focal lengths, and $c_{x,y}$ for the center of projection).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{d}{s} \begin{bmatrix} \frac{(u-c_x)}{f_x} \\ \frac{(v-c_y)}{f_y} \\ 1 \end{bmatrix} \tag{4.1}$$

This **pointcloud conversion** accuracy is harder to evaluate practically, but in a literature benchmark, given a specific 3D arrangement of items (Pratusevich et al. [2019]), these pointclouds presented a Root Mean Square Error (RMSE) of 3 to 15% per meter of distance of the camera to the object, with the best results at around 60cm. While this is not necessarily generalizeable, it provides both a sense of inaccuracy, and also a method for constructing a validation setup for a particular application scenario.

This module also (for debugging) handles pre-visualization of framesets through **OpenCV** (Bradski [2000]), and implements "masks" to retrieve parts of interest in the images.

### *DNN Modules*

The final main components of Figure 4.1 are the modules that implement the CV models. These models were implemented in python, through a computer vision framework, **Detectron2** (Girshick et al. [2018]). There were a few other frameworks that were considered, such as **Alphapose** (HPE focused python CV library), but **Detectron** was chosen, due to it supporting a greater variety of state-of-the-art models (through **torchvision**), but also providing a broader range of subfields of CV. **Detectron2** is a high-performance codebase for object detection research, with in-built tools for rapid implementation and evaluation of novel research models. **Detectron**

is implemented on top of **pytorch** (Paszke et al. [2019]), a library that implements core DL and ML features and primitives in python. It allows for evaluation, implementation and training of neural networks, as well as inference through **CUDA**, which we make use of (explored in Chapter 5).

The DNN modules are encapsulated by an object, accessible through the **pyro** interface, that takes the RGB frame and runs inferences on pre-trained models chosen from torchvision, storing their tensor predictions, and using them to trigger reactions in the **RobotControl** object.

### *Pre-trained Models*

#### MobileNet SSD

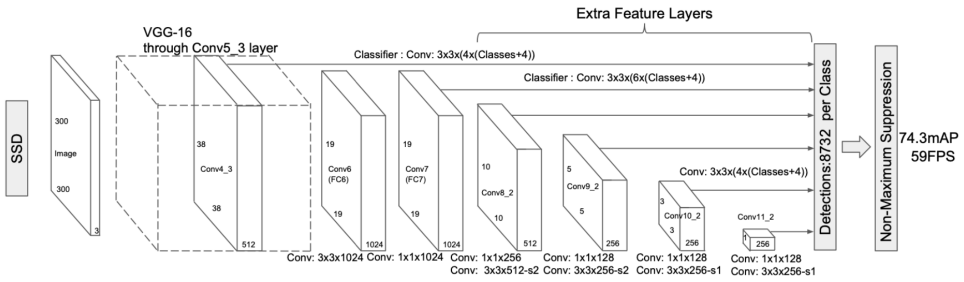



**Figure 4.11:** SSD MobileNet Architecture (Liu et al. [2015])

This model is the oldest one, and is a pure **Object Detection** model. **SSD** comes from "single shot detector" (a one stage detector) (Figure 4.11) underlying the approach that was applied to the underlying CNN-based architecture, in this case **MobileNet** (Howard et al. [2017]). These architectures (based on the underlying Visual Geometry Group (VGG)'s classical CNNs) are streamlined for low compute (e.g.mobile/embedded) applications. As such they have smaller networks ($\sim 2-10M$ parameters), lower complexity primitives and, in general lower precision, though making up for these with higher throughputs and inference speeds.

Going back to evaluating these models (Chapter 2), **SSD** has the lowest accuracy of the state of the art models (Figure 2.19). On average MobileNets achieve a theoretical mAP of 19.3 (AP at IoU=0.50:0.05:0.95)) on the **COCO** dataset (Howard et al. [2017]). This particular model was implemented from the **caffe framework's** pre-trained weights, together with **OpenCV**'s (Bradski [2000]) **DNN** module. Its implementation also required a downsampling of the image into (300,300) resolution frames. Using a **Nvidia Titan X** it had 74.5% mAP (for people) on **VOC2007** at 59 Frames per Second (FPS), and 23.2 mAP on the **COCO dataset** (AP w. IoU=.5:.05:.95)) (Liu et al. [2015]).

#### **FPN** (Lin et al. [2017])

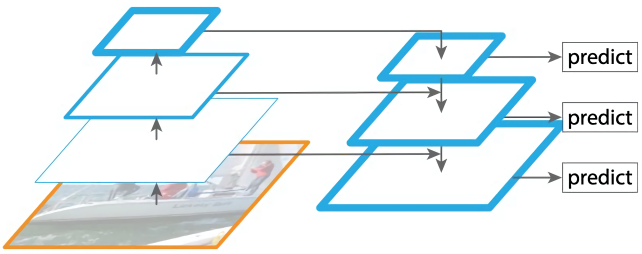



**Figure 4.12:** FPN (Feature Pyramid Network) extractor (Lin et al. [2017])

For the most intricate segmentations, the previously mentioned **Panoptic Segmentation** technique is commonly used. This is the heaviest network implemented in the project, but it ended up not being used, due to its slow performance (Chapter 5). It is based on a **ResNet** (He et al. [2016], which learns through residual functions with reference to a layer's inputs, instead of learning un-referenced functions, which has been shown to be empirically easier to optimize and more precise, due to the extreme depth) backbone architecture, upon which a Feature Pyramic Network (FPN) (Figure 4.12) is attached. An FPN is a feature extractor that, given an arbitrarily sized image as input, creates proportionally sized feature maps at multiple "scale" levels, for segmentation or Object detection tasks, resembling a "Pyramid" dependency structure. The model used in particular, **ResNet FPN** 101 had a mAP of 42.4 and around 5-10 FPS (NVIDIA V100 Graphics Processing Unit (GPU)) (Girshick et al. [2018]).

**Mask R-CNN** (He et al. [2020])

**MobileNet** had two main weaknesses: lower precision, and a rather rough "awareness", the only output is a bounding box.A new architecture is needed to be able to include **Instance Segmentation**, so we have a better "border/-mask" around the people, and also be able to detect multiple people. Moreover we need models that give better detection performance. The CNNs that were deployed given these requirements are **R-CNN** based ones. These provide a middle term between performance and precision, amongst the state of the art (Figure 2.19). These networks (two stage detectors, where the first module is a deep fully convolutional network that proposes regions (FPN), and the second module is the Faster R-CNN detector that uses the proposed regions) can also be used for keypoint detection, so finer grained detection can be applied to the people. The specific architecture (**Mask R-CNN**) extends the previous **Faster R-CNN**. It adds a "branch" for predicting an object mask in parallel with the main "branch" for bounding box recognition (He et al. [2020]). Faster R-CNN (Ren et al. [2015]) also improves on previous R-CNN (Regions with CNN Features) architectures, which use big CNNs with "bottom-up" "region proposals" to localize and segment objects, using selective search to identify a number of bounding-box object region candidates ("regions of interest"), and then extracting features from each region independently for classification/higher-level tasks. The specific model used, gave a tested mAP of 39.8 for Mask R-CNN, nearly double the precision of MobileNet. However there is a harsh rise in inference time, as the networks bottleneck at around 5-10 FPS, depending on whether they use keypoint detection, or just instance segmentation, respectively (Jiao et al. [2019]; Girshick et al. [2018]).

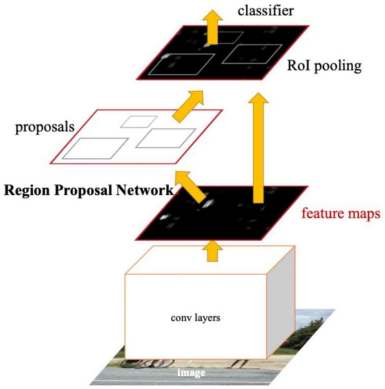



**Figure 4.13:** Faster R-CNN Architecture (Ren et al. [2015])

Now that the top-down system approach has been given, and the sub-components are introduced, the next chapter will go over the qualitative and quantitative results, along with some more implementation related details, followed by the demos and the conclusions.

# 5 RESULTS

This chapter will start with a component-by-component review, followed by a general system overview.

The **pyro** layer was benchmarked on a series of different metrics. A **pyro** server can connect to 2384 proxies per second, with an average remote method call taking 0.101 milliseconds (msec) (9944 calls/sec throughput). By using batched calls (a smarter subsystem) this can be improved to 0.0215 msec (46500 calls/sec), a speedup of around 6.7x, with further optimizations leading even into (211300 calls/sec). For transferring more complex data (byte arrays for images, etc), the system achieved transfer speeds from 46.7 mb/sec up to 85.0 mb/sec, allowing a theoretical (148.57 to 272.38 FPS for constant (naive) frame streaming). There are no discernible bottlenecks in the networking layer in regards to the system functionality.

## 5.1 ROBODK CONTROL

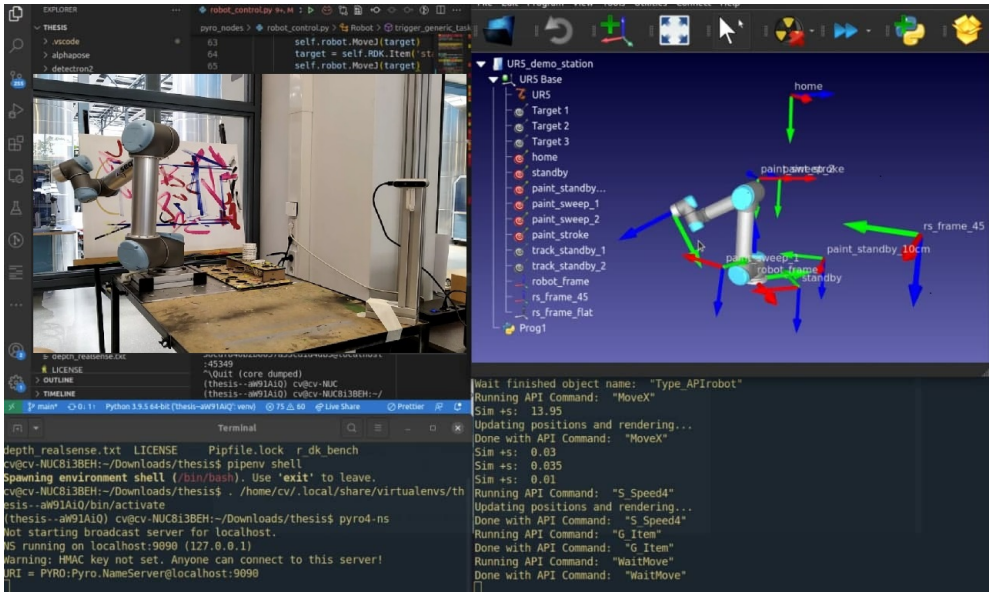



**Figure 5.1:** Example of Joint Movement and Visualization through python **roboDK** API

The **RoboDK** API allows real time visualization and control of the robot. Though the Safety stop signal is immediate, updating a movements parameters, or defining new movements is bottlenecked by the event dispatcher of the UR5 robot controller, as the API merely packages the code instructions as TCP instructions for the robot controller, such as circular, joint or linear robot movements, as well as joint velocity/acceleration parameters. Given this "conversion" process, movements cannot be altered "mid-process". As such, full real-time control is limited to the **individual inter-motion level**, the only available intra-movement control is for complete movement shutdown (thus all non-safety stop actions must wait until the next "movement" order is given to the robot). The interface allows for accurate real-time

visualization of both the reference frames, workstation targets, and even tracked objects.

## 5.2 REALSENSE EXTRACTION AND LOCALISATION

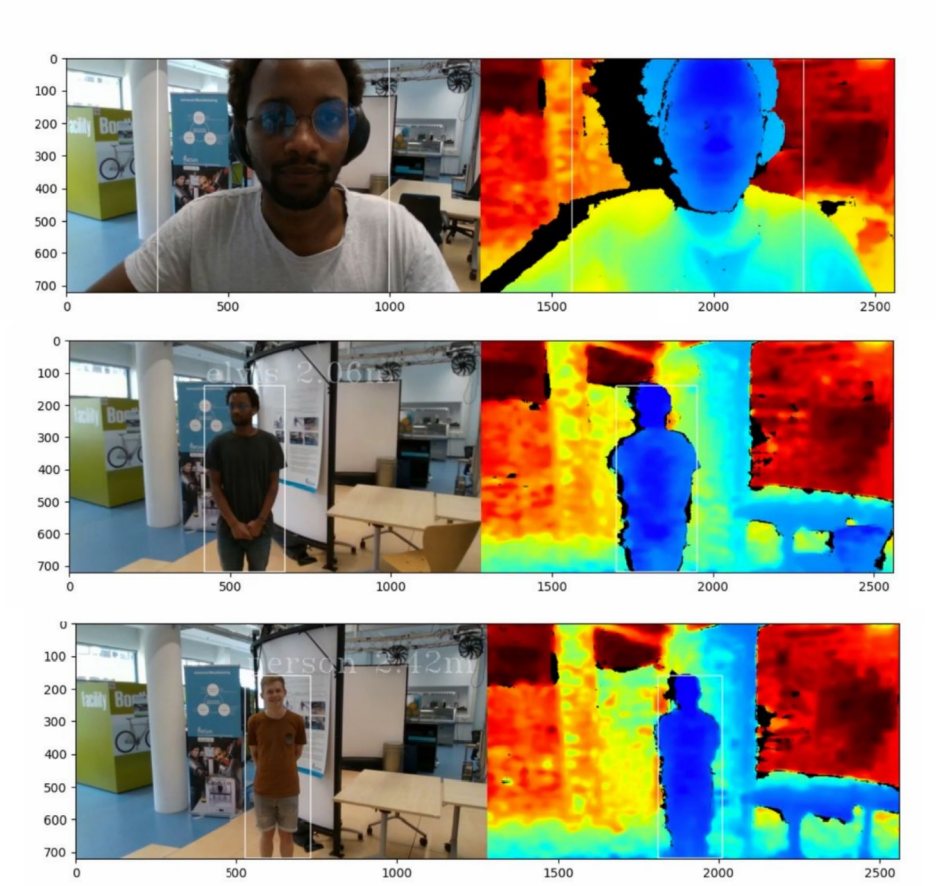
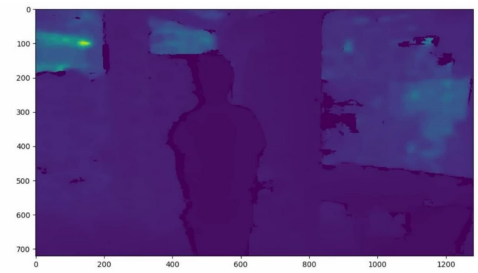



**Figure 5.2:** Examples of **RGB-d** captures, overlayed with MobileNet object detector. "Gap" depth pixels are black.

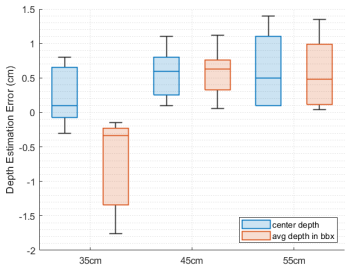
In the first layer of the perception stack, the Realsense image + depth frame is acquired. The realsense API does not provide inbuilt GPU acceleration for the image pipeline/alignment algorithm, so it is a Central Processing Unit (CPU) constrained workflow. The initial acquisition was implemented in a python node running in a Macbook Air with a Quadcore 1.1 Gigahertz (GHz) Intel i5. Acquisition (with alignment was limited to 0.5 FPS. Once the system was ported to the final configuration (Intel NUC with a quadcore Intel Core i3-8109U CPU 3 GHz), the framerate capture improved to 55.3 FPS (with depth buffer, and naive alignment). While this is not the highest framerate that could be achieved with the realsense system (up to 300 FPS), 60 FPS for the innate image+depth buffer allows a compromise between frame resolution, Field of View (FOV) of the capture and the acquisition framerate. This configuration allows for a 848x480 pixel RGB-d frame.

One important point to note is that, for some of these RGB-d examples (Figure 5.2), the position of the camera relative to the main targets will influence the "dead area" around the depth bugger reconstruction. As the depth buffer is reconstructed given the inter-image sequence disparities, an extreme angle to the object of interest will lead to larger disparity gapes, and thus a larger "empty" depth

buffer zone. This becomes a problem especially at extremely close positions of the objects to the center of a camera. Realsense provides hole filling / temporal/spatial hole filters to improve these raw depth buffers (Grunnet-Jepsen and Tong [2020]). These filters were applied in the codebase, however this post-processing can lead to artifacts, and initial experimentation did not lead to conclusive results whether using them (specifically the hole-filling filter) was unequivocally better versus ignoring "gap" data for depth calculations. Also, both the depth estimation error and the "dead zones" increase as objects get too close to the camera aperture, or remain at extreme angles in relation to the line of sight ray coming out from the center of the camera.[1]



**(a)** Aligned Depth frame, prior to conversion to absolute metrics




**(b)** Depth Estimation error in cm, for object center and Bounding box

**Figure 5.3:** Depth Estimation Results from Realsense camera tests

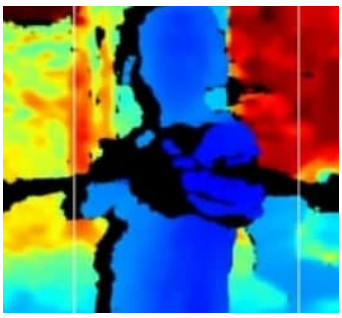


**Figure 5.4:** Aberration example, stick "splits" in close camera range

Figure 5.3 contains a typical example of an obtained depth frame (Figure 5.3a). One must note that even for surfaces in the same depth plane, there are inconsistencies as this method is not completely immune to camera/lighting aberrations. There are certain improvements that could be approached to the depth frame itself (the efficacy of which will be discussed in Chapter 6), but this will be by default the one used for the demos. A simple benchmark for testing the accuracy of depth estimation for flat objects was also used, leading to the results in Figure 5.3b. In , the Z-depth distance error remains in the cm range. The accuracy is optimal ($0.5 \pm 1.5$ cm) within $0.5 - 1m$ of the camera. Closer objects will present distortion effects (Figure 5.4), and for distances further than 3m the disparities will become small enough that accuracy worsens (increasing by $\sim$1 order of magnitude). Retrieving a depth frame is not enough however, as this frame needs to be projected into the same perspective as the **RGB** frame. This is an operation provided by the **rs_library**, but it takes an average of 0.0237 seconds. This means that the aligned frame pipeline is bottlenecked at $\sim$ 7.2 FPS (using a naive "always on" alignment).

## 5.3 OBJECT/PERSON DETECTION

We come now to the core perception nodes, the NNs. As mentioned in Section 2.2.2, 3 CNN based networks were chosen. The simplest (and fastest one) **MobileNet** is based on a **caffemodel** (NN format used for cross-platform implementation), and

1 The **rs** camera possesses a rather narrow FOV - 69° × 42° (Hor, Vert), which is also affected by the resolution parameters, etc (Intel [2020]). This "parameter" exploration will be approached again in Chapter 6.

is implemented with the help of **openCV**'s **DNN module**. This networks requires a standard input, so the input RGB image is pre-processed (cropping, recalculating aspect ratio and a slight down-scaling of the resolution). While faster in inference (achieving an average 25.4 FPS), it is a rather rough object estimator, (no semantic segmentation is provided) and its results are limited - as discussed in section 4.4.3 - leading to lower percentage of pixels inside a **BBx** belonging to a person if all their limbs are outstretched, or moving quickly, for example). This can be seen qualitatively in Figure 5.5.

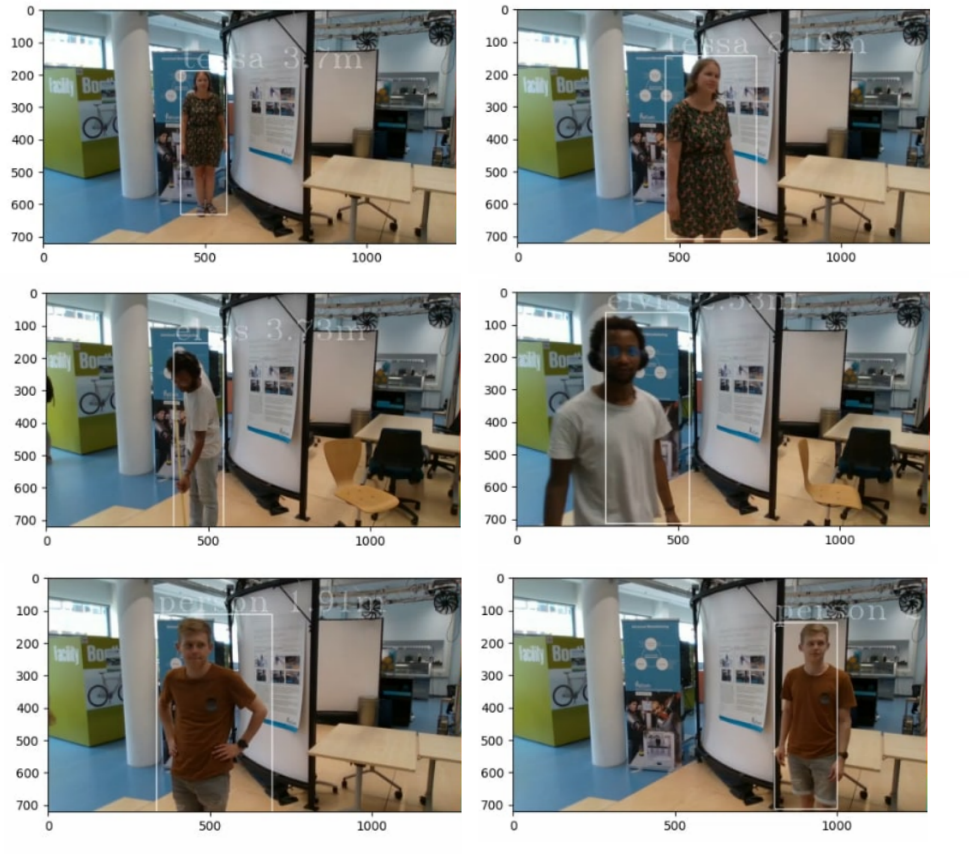


**Figure 5.5:** Examples of **MobileNet** detections, with corner cases on the right (either incomplete or non-tight bounding boxes)

To achieve higher accuracy, and retrieve more rich output tensors, **RCNN** based networks were then implemented, through the use of the **detectron** framework. This had a noticeable performance drop running on the CPU only system (avg 0.5 FPS). So at this point a **razercore** external GPU enclosure was added to the NUC system, together with a **Nvidia Geforce GTX 1070** GPU, allowing for offloading the inference of these networks to it, where they run at 3∼7 FPS. The reason the system is benchmarked in terms of FPS, instead of other metrics, is that it provides a time agnostic way of measuring performance, which can be related to timing by extrinsic factors (the camera frame pipeline can be used as a sort of system clock), i.e. given a constant **RGB-d** buffer sequence with a determined **FPS** rate, all other events of the system can be timed to this "clock". It also makes sense given that the frame sequence is the main "driver" of the functionality of the overall system, as it triggers the reactions from the robot and the perception stack nodes. These **RCNN** networks (Figure 5.6), now running at 6.21 FPS allow for instance detection of multiple persons, with an actual segmentation, rather than just a bounding box. Its main disadvantages relate to its performance and added tensor complexity.

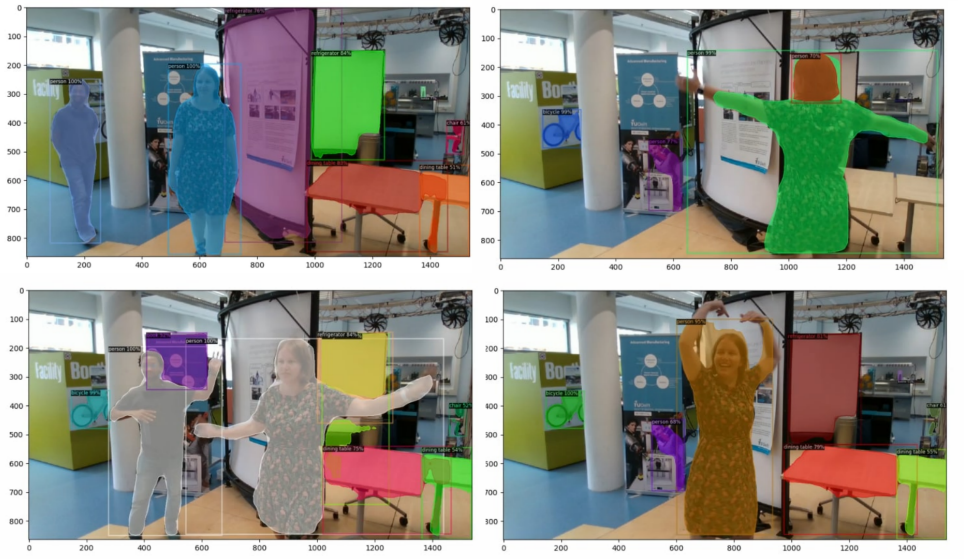


**Figure 5.6:** Examples of **R-CNN** detections, with corner cases on the right (either incomplete segmentations or low confidence results)

A **Panoptic Segmentation** network (Figure 5.7) was also implemented from the **detectron** framework, however this did not end up in the final demos, as its performance was not enough for the implemented demos (roughly 3.36 FPS for inference). However, it showed promising results in terms of accuracy, and possible methods to increase this performance are explored in Section 6.3. It is important to note that these are benchmarks without using the debug visualizers. In this case, the visualizer actually takes up most of the processing, requiring an average of 0.53272 seconds. This limited the whole **NN** pipeline to only ∼1.1 FPS! The average impact on the overall system is exploring the later section of this chapter.

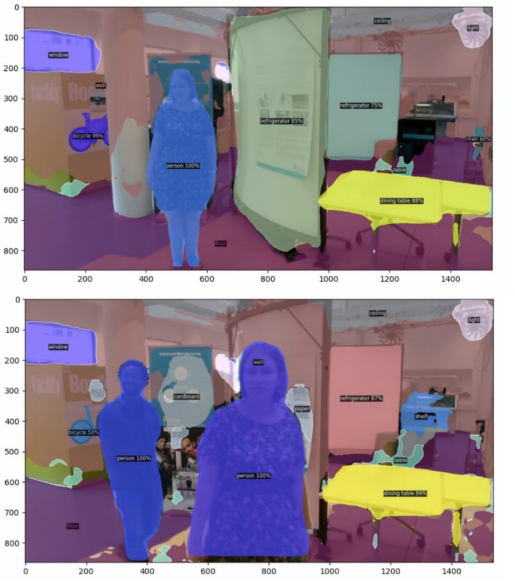


**Figure 5.7:** Examples of **R-CNN** panoptic segmentations, note the main errors being boundary occlusions/fuzzyness

Finally, for even more fine-grained person detection, a 2D HPE **RCNN** was implemented (Figure 5.8), based on another pre-trained detectron module. This, enhanced with the depth localization, provides a multi-person skeleton detection in the 3D scene. This network performed well (avg 6.51 FPS), however it is more prone to boundary case errors, principally in the cases where people are close to one another, causing "merges" in the detected skeletons, or when people are occluded by the FOV of the realsense camera, leading to certain joints' locations being invalid or, in the worse case, being distorted. These results where minimized as the main joints of focus are the head and arms, and most of these examples occur for the legs and knees. The system also suffers when dealing with very high speed movements, as it deals with each frame individually and does not use information from the sequence to bias consecutive predictions.

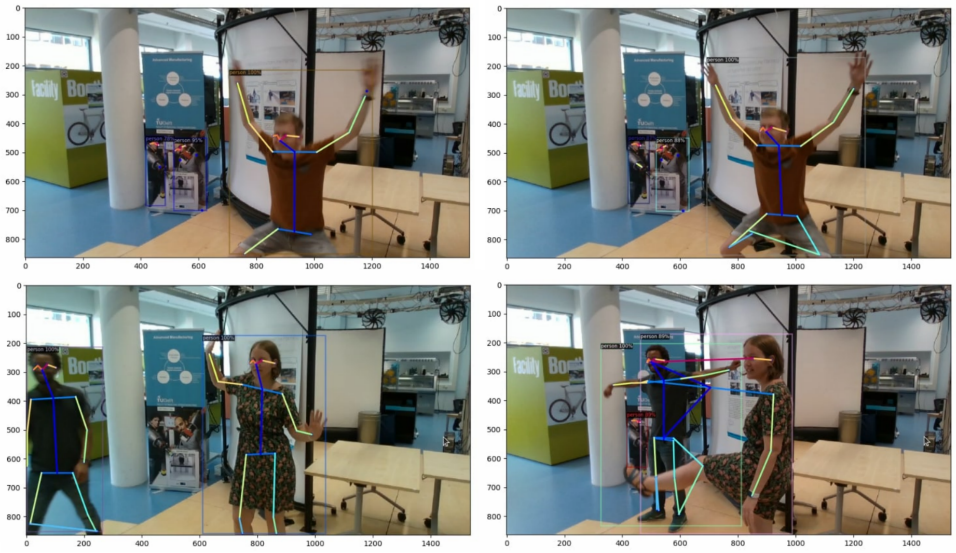


**Figure 5.8:** Examples of **R-CNN** keypoint detections, with corner cases on the right (either incomplete skeletons, joint misplacement near frame borders or incorrect merging of keypoints between people in the same frame

## 5.4 SAFETY STOP

This "proto-interaction demo focuses on safety, as mentioned in Chapter 1. In a scenario where the cobot is doing a preset task (painting simulation where it continuously draws a line in a canvas), the presence of a person within the cobot's vicinity should stop the motion. Through the vision node, if a person's bounding box is detected, the person's average distance to the camera is calculated, (and consequently the distance to the robot through trigonometry (as the camera's reference frame is static relative to the robot's base frame)).

Once this distance crosses a predetermined threshold distance (1.5m), the vision node (Figure 5.9) triggers a stop command through the **pyro** interface ( triggering the safety stop exposed function from **RobotControl**) to the currently running task on the **RobotControl** module. This feedback is immediate, but it was found that the person had little sense of the action, as the only feedback trigger were the cobot's brakes.

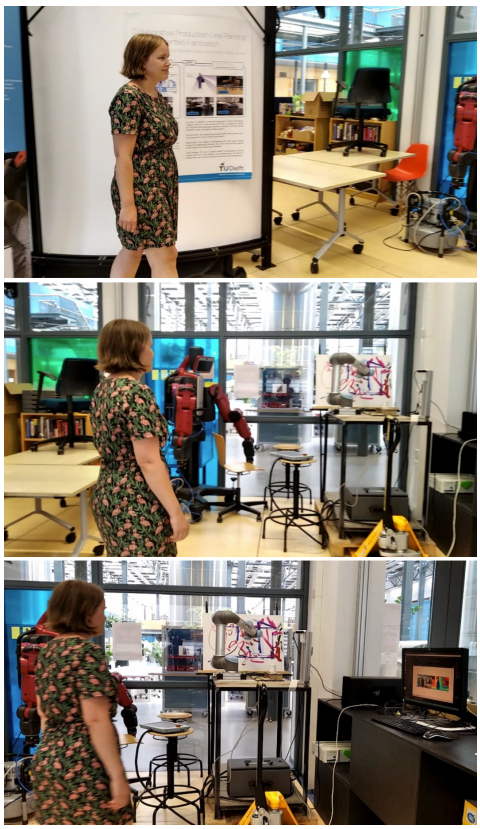


**Figure 5.9:** Safety Stop demo, as person approaches robot at work

## 5.5 SAFETY SLOWDOWN

Safety Slowdown is the second demo (Figure 5.11), and it tries to achieve a more reactive and natural reaction from the robot. Instead of using the raw distance threshold, The camera still captures the operator's active distance, but instead maps this distance into a velocity range through an S-Curve (Figure 5.10), tuned for a range of velocities such that the task at hand slows down to a crawl as the person remains near the robot area, and speeds up again when the person retreats.

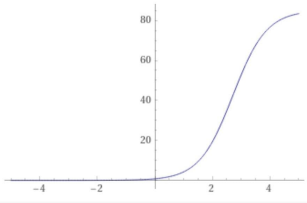



**Figure 5.10:** Designed S-curve

**Figure 5.11:** Safety Slowdown demo, as person approaches robot at work, its task velocity will be dynamically linked to operator's position relative to robot

This S-curve will then map between a person's distance in meters to the maximum movement velocity of the robot's joints, in $mm/s$, This is applying the general principles used in the animation and interaction approaches mentioned in Section 3.2.1. The curve is designed so that within 1.5 meters of a person, the robot's movement velocity is practically zero, and that it reaches its maximum at around 4m. This was obtained by solving a constrained logistic function ($f(x) = \frac{1}{1+e^{-k(x-x_0)}}$) defined in such a way so that constants $a$ and $b$ are found such that an stretched logistic function passes between two points, $(1,5)$ (m, mm/s) and $(3,60)$, leading to the following analytic solution:

$$\begin{bmatrix} a+b = \log_{10}\left(\frac{0.05}{(1-0.05)}\right) \\ a+b\times 3 = \log_{10}\left(\frac{0.6}{(1-0.6)}\right) \end{bmatrix} \tag{5.1}$$

Which leads to the following curve parameters $a = -4.66208$, $b = 1.71764$. The S-Curve's final formulation is as follows:

$$f(x) = \frac{85}{1+105.856e^{-1.71764x}} \tag{5.2}$$

This allows for a "smooth" interplay between the distance to the operator and the way the robot moves. Of course this curve can be adjusted, and other parameters besides just velocity approached.

## 5.6 SAFETY TRACK

The last demo attempts to improve on that, although useful, an average distance to a person is not information that is fine-grained enough for more complex interactions. As such, a 2D skeleton allows for an estimation in the 3D environment of every joint of a particular person, which then allows for things such as the application of the

previous two tests to just the person's hands for example, or give priority to certain body parts rather than the person as a whole.

By retrieving the 2D Human Pose Estimation pixel-joint positions, and enhancing that prediction with an extra depth value from the buffer at the joint's location, these inputs can the be used to retrieve a 3D location from the robot's frame of all the joints of a person. With these keypoints embedded in the surrounding 3D space (Figure 5.12), the robot behaviour can be adapted in more intelligent ways (having the head or hands position trigger the safety stop, etc.).

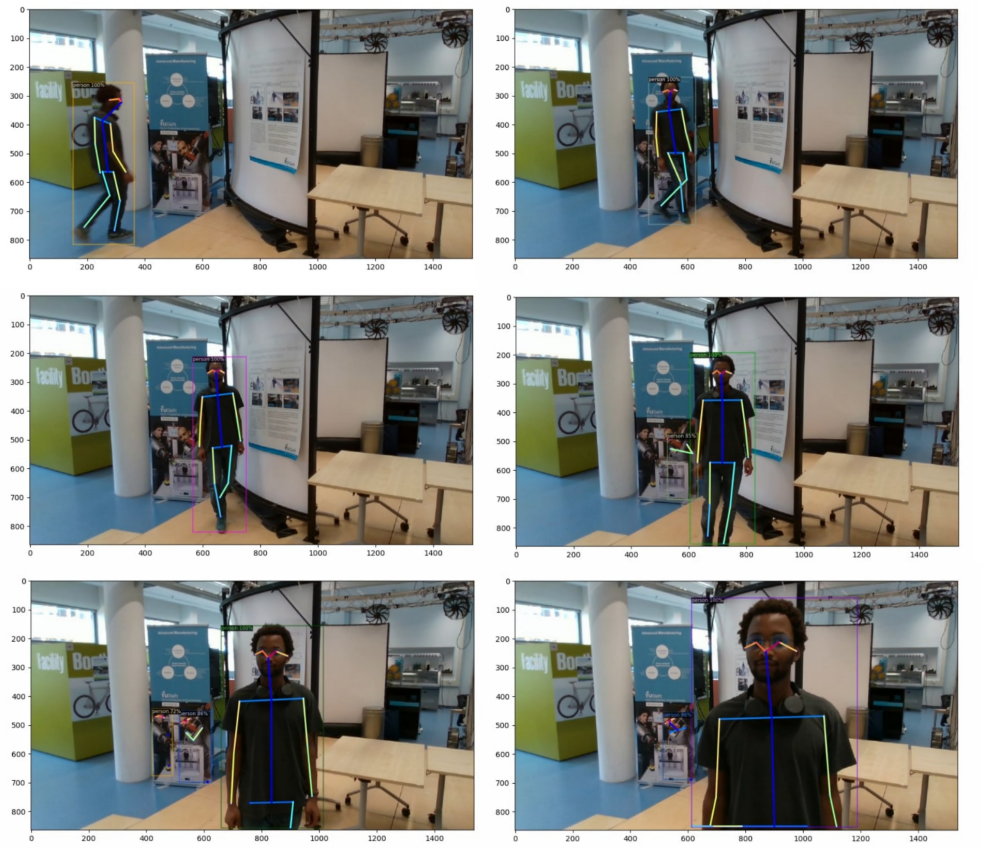


**Figure 5.12:** Safety Track where a person's skeleton is tracked through 3D space around the robot

## 5.7 PERFORMANCE BREAKDOWN

By summarizing all of the previously mentioned performance details for these models, along with the overheads associated with the visual and console debugging tools, the following table condenses the system's main quantitative benchmarks:

| **Model** | mAP | theoretical inference (s) | inf. (s) | inf. (FPS) | viz. (s) | Viz+Inf | FPS (w/debug) |
|---|---|---|---|---|---|---|---|
| **MobileNet** | 19.3 | 0.0169 | 0.0394 | 25.4 | 0.113 | 0.152 | 8.2 |
| **mask_rcnn_R_50_FPN (instance segmentation)** | 38.6 | 0.043 | 0.161 | 6.2 | 0.297 | 0.458 | 2.2 |
| **panoptic_FPN_R_101** | 36.5 | 0.053 | 0.23 | 4.36 | 0.533 | 0.762 | 1.31 |
| **keypoint_RCNN_R_50_FPN** | 55.4 | 0.066 | 0.154 | 6.5 | 0.173 | 0.33 | 3.06 |

**Table 5.1:** Aggregated model benchmarks, both on expected theoretical metrics, aswell as system results

# 6 CONCLUSION

Having a feature complete prototype system (Chapter 4) and after initial testing of the first "proto-interactions" (Chapter 5), we come to the conclusions of this project where the major findings are presented, followed by a reflection upon the current system weaknesses, and ending with recommendations for future work which remains to be done.

## 6.1 MAIN FINDINGS

First of all, the literature review (Chapter 2 and Chapter 3) reinforced the vision-led approach as a strong, scalable, and cheaper one, when compared to alternatives, and why such human-hybrid systems will have increased relevance in the future. Moreover, within vision, this review already shows there are useable models for deploying ML-based algorithms for these vision tasks into actual production workflows (In section 4.4.3 of Chapter 4). A further review also justified how a vision approach could be built to effectively use this captured data to enable human-cobot interaction (Section 3.6), ending with a prototype showing simple initial examples (Section 5.4, Section 5.5, and Section 5.6), that already provide basic workflows on un-optimized hardware systems.

Moreover, this project implemented several state of the art vision models (in section 4.4.3), successfully capturing both 2D low level human feature data (Section 5.3), merging this information with the depth buffer (Section 5.2) and achieving an average "full-system" framerate of 3-5 FPS (Section 5.7).

The final system was implemented in a modular structure (Figure 4.4.1), allowing these decoupled modules to effectively communicate (Chapter 5) and trigger simple interactions in a target **UR5 cobot**. This control module, through the **robodk** APIs, allows for a configurable workspace, one that is not limited to a preset configuration, nor a preset cobot model.

Finally, we come back to research question 1.4:

RRQ-1. How can we create a Vision based system such that a Cobot (collaborative robot) can react to a human within its environment?

This thesis has presented research into vision, how it can be applied to this problem domain, together with current and relevant HRI research. A concept system was developed (available as an open-source repository) given all of this, with the final concept choice being a flexible software prototype that is both low-to-medium cost and able to effect simple human-robot interactions.

Given these preliminary results, all of these interacting blocks provide a solid baseline system, which can be developed into a specific vision-based HRI application successfully.

## 6.2 WEAKNESSES

From this proof-of-concept prototype, several conclusions can already be drawn. In literature, an average humans' speed of movements circles around 1.4 metres per second for walking, to 0.25-0.5 seconds for arm and other more fine-grained motions (Buzzell et al. [2013]; DeGoede et al. [2001]). This means that a human-cobot system framerate should support at least 3 to 15 FPS in order to be synced with/allow productive interaction in HRI systems.

Again, the core prototype system still only runs at an average 3-5 FPS, which is not enough for enabling the most complex interactions (Section 3.2.1). Moreover, on the cobot's reaction side, the fact that once an instruction is sent, the only immediate change that can be effected is stopping all movements and locking the brakes (Section 5.4), which is a severe limiting factor. In case a cobot action lasts for longer than this system's budget of 200ms, altering it requires waiting for the current movement to be completed first, and clearing the current robot's task queue, leading to the "sluggishness" seen in the safety slowdown demo (Section 5.5). As such, robot movements should be scheduled/broken down into smaller chunks, to allow for dynamic path/velocity constraints in line with system performance.

On the perception side, the implemented Neural Networks showed the expected results in accuracy (by proxy of the "confidence" parameter of the output tensor, and their qualitative results), however their inference speeds paled in regards to the reference inference speeds stated for the models (Section 5.7), these being ∼25-40% faster. This indicates that the current GPU solution (Section 4.2) is underpowered.

Also, it is important to note that the models are used "as-is" (section 4.4.3 of Chapter 4). Proper optimization (such as extra, transfer learning to tune the network's accuracy for a specific scenario, or "pruning" (setting certain unnecessary pathways of the network to zero) of the network, to remove structures trained on irrelevant objects) is necessary for the extra performance and robustness. Nonetheless, the demos and preliminary results show the feasibility of a purely vision-driven system, and its implementation on cobot systems.

One point that came from the initial demos is that the relatively narrow FOV of a single camera system resulted in the "perception cone" having quite a few "dead angles". Though realsense provides some software features to increase the FOV slightly, these would come with a cost in frame resolution (Section 5.2). Indeed, for a complete overview of the human/objects related to them, a multi camera system seems more appropriate. However these extra cameras do not need to be realsense depth cameras as well, cheaper monocular cameras could be added to the current system, perhaps with depth estimation models running on their outputs, to help the estimations with the ones from the realsense camera, besides providing a more complete view of the 3D environment, with software based stitching.

Another point to note about the depth buffer is its presence of "gap" pixels, due to its disparity estimation algorithm not being able to perfectly synchronize the multiple views in the realsense camera. This is exacerbated for motions that occur too fast for the system framerate to handle, leading to blurriness, but also if the objects/people are at an extreme orientation from the center of the bisector line coming out of the realsense camera. This is also something that can could be aided by a multi-view camera setup. The software hole-filling filters provided by realsense help with this, but they introduce aberrations in the depth data, which are hard to evaluate.

On the HPE side, given that the system only runs perception "one frame at a time", the networks cannot take advantage of previous frames for the inference tasks of the next ones, which could be used to maximize performance, as well as determine higher levels of awareness, and capture "intent", such as the "actions sequences" mentioned in Chapter 3.

On the control side, though **pyro** as a system provided an agnostic way of passing messages, tests (Section 5.1) showed that there was some level of interference between the **UR5** controller, **robodk**, and the pyro messages. Since there is a single object controlling the **UR5** cobot arm, use of the system showed that calling a function interacting directly with the controller (safety stop) could bring crashes, presumably due to the asynchronous system behaviour that could lead to several simultaneous safety stop requests being present simultaneously in the response queue of the robot controller object. These events were rare, but nonetheless the system was changed so that only program parameters were sent between nodes, and the actual function calls were isolated to only "inside" each relevant node.

Finally, while there was an initial setup of the realsense camera, this parameter calibration was far from extensive, and the intrinsic parameters used were the ones preset at the factory for the used device. Extra tuning and calibration would be needed, tuned to a specialized calibration setup, especially for ensuring the precision of the augmented human skeleton in the 3D environment. Also, the current default model evaluation metrics (Section 2.2.5) do not explore how well this 3D skeleton reconstruction actually is, so this part of the evaluation is lacking.

## 6.3 FUTURE WORK AND IMPROVEMENTS

Throughout the project, several ideas for better approaches/alternative to the aforementioned weaknesses were thought of. Some of these can be summarized by the following points, starting with perception:

- By integrating the different subtask networks into a shared backbone network, we allow the system to have extra paralelization advantages, as well as performance, instead of merging the results of the disparate networks after the fact;
- Applying transfer learning with actual datasets of workers' environments, so that the networks are "fine-tuned" to the specific workspaces, rather than only the generic larger datasets;
- Prune base networks for unnecessary functionality, to increase performance;
- Implement proper realsense camera calibration, image pre-processing, and system calibration in general, exploring the setup parameters (camera position, and number of camera array, etc), tuning the post-processing filters, and finally devise a calibration test for the 3D HPE method, perhaps by using mannequins/cardboard cutouts with known measurements, so that the 2D to 3D skeleton conversion accuracy can be validated under the specific constraints of the scenarios, also alternative 3D camera systems can be explored as well;
- Add an action sequencing system, that also takes sequences of frames, for **higher level perception tasks** (Section 3.2.1), which would allow the system to become even more **predictive**, instead of solely **reactive**, thus allowing more complex interactions.
- Instead of applying naive stitching/object detection algorithms with every received frame, introducing more dynamically intelligent framerates, such as withholding/skipping bottleneck prone activities in very similar frames, or where nothing of interest can be detected initially;

There are also needed improvements on the robot action side:

- Implementing an actual task system, with an event handler/dispatcher, that would not only allow more generic scalable tasks (leaving the hard coded

ones behind), but also be able to break down **intra-task** actions into time segments such that a single movements' duration does not bottleneck the system perception response;

- Integrate more intelligent planning methods, such as handling new movement targets dynamically given the robot's constraints, or being able to catch simpler action errors, instead of these simply cancelling a movement.

More broadly, now that the proof of concept system is proven, more effort is need to turn it into a full prototype robot station, such as moving the current code into a more performant compute architecture (some 30% of performance gains could be gained from using better GPU compute, e.g. the NVIDIA® Jetson Nano NN optimized computers), that could be used to trial and tune the system with specific automation tasks that require human and robot hybrid-cooperation. While accuracy is already good enough for the system, increasing average system performance to at least 12 FPS, is essential for this task.

While there are several parts of it that require more work, these are approachable engineering problems, and both the research and the proof of concept did not present any major flaws in the underlying approach. While there is more research that needs to be done, as well as engineering, Vision appears as **a viable, lower cost, scalable and performant** software-based approach for computational human awareness, when compared to other sensor based methods, and there are promising results from using such awareness data for successful Human-Cobot interactions, both in literature, as well as the proto-interactions implemented as part of this un-optimized and limited proof of concept, which nonetheless still managed to provide useful behaviours from the cobot's perspective given human presence, **Q.E.D**.

# BIBLIOGRAPHY


Afzal, M. R. (2015). A portable gait asymmetry rehabilitation system for individuals with stroke using a vibrotactile feedback. *BioMed research international*, 2015:375638.

Aksoy, E. E., Abramov, A., Dörr, J., Ning, K., Dellen, B., and Wörgötter, F. (2011). Learning the semantics of object–action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249.

Althaus, P., Ishiguro, H., Kanda, T., Miyashita, T., and Christensen, H. (2004). Navigation for human-robot interaction tasks. In *Proceedings—IEEE International Conference on Robotics and Automation*.

Amidi, A. and Amidi, S. (2020). Cs 230 deep learning, stanford.

Amoruso, L., Finisguerra, A., and Urgesi, C. (2016). Tracking the time course of top-down contextual effects on motor responses during action comprehension. *Journal of Neuroscience*, 36(46):11590–11600.

Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ardon, P., Pairet, E., Petrick, R., Ramamoorthy, S., and Lohan, K. (2019). Learning grasp affordance reasoning through semantic relations. *IEEE Robotics and Automation Letters*.

Asadi-Aghbolaghi, M., Clapés, A., Bellantonio, M., Escalante, H. J., Ponce-López, V., Baró, X., Guyon, I., Kasaei, S., and Escalera, S. (2017). A survey on deep learning based approaches for action and gesture recognition in image sequences. In *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, pages 476–483.

Attention, I. S., Performance, P. H., Rossetti, Y., and Kawato, M. (2008). *Sensorimotor foundations of higher cognition: Attention and Performance XXII*. Oxford University Press, Oxford.

Avelino, J., Garcia-Marques, L., Ventura, R., and Bernardino, A. (2021). Break the ice: a survey on socially aware engagement for human–robot first encounters. *International Journal of Social Robotics*, pages 1–27.

Balit, E., Vaufreydaz, D., and Reignier, P. (2018). Pear: prototyping expressive animated robots a framework for social robot prototyping. In *VISIGRAPP 2018—Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*.

Bartneck, C., Kanda, T., Mubin, O., and Mahmud, A. (2009). Does the design of a robot influence its animacy and perceived intelligence? *International Journal of Social Robotics*.

Bauer, W., Bender, M., Braun, M., Rally, P., and Scholtz, O. (2016). Lightweight robots in manual assembly–best to start simply. examining companies' initial experiences with lightweight robots. *Frauenhofer-Institut für Arbeitswirtschaft und Organisation IAO, Stuttgart*, 63.

BCG Group, G. (2015). Report on man and machine in industry 4.0: How will technology transform the industrial workforce through 2025?

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Breazeal, C. (2003). Toward sociable robots. *Robotics and Autonomous Systems*, 42(3–4):167–175.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Buzzell, G., Chubb, L., Safford, A. S., Thompson, J. C., and McDonald, C. G. (2013). Speed of human biological form and motion processing. *PLOS ONE*, 8(7):null.

Casser, V., Pirk, S., Mahjourian, R., and Angelova, A. (2019). Unsupervised monocular depth and ego-motion learning with structure and semantics.

Chau, A., Sekiguchi, K., Nugraha, A. A., Yoshii, K., and Funakoshi, K. (2019). Audio-visual slam towards human tracking and human-robot interaction in indoor environments. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8.

Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915.

Chen, Z. and Liu, B. (2016). Lifelong machine learning. In *Synthesis Lectures on Artificial Intel- ligence and Machine Learning*.

Ciolek, T. and Kendon, A. (1980). Environment and the spatial arrangement of conver- sational encounters. *Sociological Inquiry*, 50(3–4):237–271.

Coleman, D., Șucan, I. A., Chitta, S., and Correll, N. (2014). Reducing the barrier to entry of complex robotic software: a moveit! case study. *Journal of Software Engineering for Robotics*, 5(1):3–16.

Colgate, J. E., Edward, J., Peshkin, M. A., and Wannasuphoprasit, W. (1996). Cobots: Robots for collaboration with human operators.

Colley, M., Bräuner, C., Lanzer, M., Walch, M., Baumann, M., and Rukzio, E. (2020). Effect of visualization of pedestrian intention recognition on trust and cognitive load. *12th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*.

Colley, M., Eder, B., Rixen, J. O., and Rukzio, E. (2021). Effects of semantic segmentation visualization on trust, situation awareness, and cognitive load in highly automated vehicles.

CoRo Laboratory (2015). Robodk.

Cretu, A. L., Ruddy, K., Germann, M., and Wenderoth, N. (2019). Uncertainty in contextual and kinematic cues jointly modulates motor resonance in primary motor cortex. *Journal of Neurophysiology*, 121(4):1451–1464. PMID: 30811258.

Csáji, B. C. (2001). *Approximation with Artificial Neural Networks; Faculty of Sciences*. Eötvös Loránd University, Hungary.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1.

Dalenogare, L. S., Benitez, G. B., Ayala, N. F., and Frank, A. G. (2018). The expected contribution of industry 4.0 technologies for industrial performance. *International Journal of Production Economics*, 204:383–394.

Darling, K. (2017). *Who's Johnny? Anthropomorphic framing in human–robot interaction, integration, and policy. Robot Ethics 2.0: From autonomous cars to artificial intelligence (Vol.1*. Oxford University Press, Oxford.

DeGoede, K. M., Ashton-Miller, J. A., Liao, J. M., and Alexander, N. B. (2001). How Quickly Can Healthy Adults Move Their Hands to Intercept an Approaching Object? Age and Gender Effects. *The Journals of Gerontology: Series A*, 56(9):M584–M588.

Di Cesare, G., Di Dio, C., Rochat, M., Sinigaglia, C., Bruschweiler-Stern, N., Stern, D., and Rizzolatti, G. (2014). The neural correlates of 'vitality form' recognition: an fmri study: this work is dedicated to daniel stern, whose immeasurable contribution to science has inspired our research. *Social cognitive and affective neuroscience*, 9(7):951–960.

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

Duarte, N. F., Raković, M., and Santos-Victor, J. (2019). Coupling of arm movements during human-robot interaction: The handover case. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, page 1–6. IEEE Press.

Eldardeer, O., Sandini, G., and Rea, F. (2020). A biological inspired cognitive model of multi-sensory joint attention in human robot collaborative tasks.

Elfaramawy, N., Barros, P., Parisi, G. I., and Wermter, S. (2017). Emotion recognition from body expressions with a neural network architecture. In *Proceedings of the 5th International Conference on Human Agent Interaction*, HAI '17, page 143–149, New York, NY, USA. Association for Computing Machinery.

Esterwood, C., Essenmacher, K., Yang, H., Zeng, F., and Robert, L. (2021). A meta-analysis of human personality and robot acceptance in human-robot interaction. *Available at SSRN 3768022*.

Facebook-AI-Research (2021). Papers with code - cv state of the art.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *Int. J. Comput. Vision*, 61(1):55–79.

Fortun, D., Bouthemy, P., and Kervrann, C. (2015). Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 134:1–21. Image Understanding for Real-world Distributed Video Networks.

Fuchs, S. and Belardinelli, A. (2021). Gaze-based intention estimation for shared autonomy in pick-and-place tasks. *Frontiers in Neurorobotics*, 15:33.

Gerstner, W. and Kistler, W. M. (2002). *Spiking neuron models: single neurons, populations, plasticity*. Cambridge University Press, Cambridge, U.K.

Giraud, T., Focone, F., Isableu, B., Martin, J., and Demulier, V. (2016). Impact of elicited mood on movement expressivity during a fitness task. *Human Movement Science*.

Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., and He, K. (2018). Detectron. https://github.com/facebookresearch/detectron.

Gkioxari, G. and Malik, J. (2015). Finding action tubes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 759–768.

Gong, W., Zhang, X., Gonzàlez, J., Sobral, A., Bouwmans, T., Tu, C., and Zahzah, E.-h. (2016). Human pose estimation from monocular images: A comprehensive survey. *Sensors*, 16(12).

Gorecky, D., Schmitt, M., Loskyll, M., and Zühlke, D. (2014). Human-machine-interaction in the industry 4.0 era. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 289–294.

Gray, J., Hoffman, G., and Adalgeirsson, S. (2010). Expressive, interactive robots: Tools, tech- niques, and insights based on collaborations. In *HRI 2010 Workshop: What Do Collaborations with the Arts Have to Say about HRI*.

Grunnet-Jepsen, A. and Tong, D. (2020). Depth post-processing for intel® realsense™ depth camera d400 series.

Guerrero, D. S. E. (2015). Tutorial on datasets, annotations and metrics in computer vision.

Hafiz, A. M. and Bhat, G. M. (2020). A survey on instance segmentation: state of the art. *International Journal of Multimedia Information Retrieval*, pages 1–19.

Hancock, P. A. (2011). A meta-analysis of factors affecting trust in human-robot interaction. *Human factors*, 53,5:517–27.

HAVEP Workwear (2020). Havep® smart workwear.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2020). Mask r-cnn. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):386–397.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Hemeren, P. and Thill, S. (2011). Deriving motor primitives through action segmentation. *Frontiers in Psychology*, 1:243.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

Hussein, A., Gaber, M., Elyan, E., and Jayne, C. (2017). Imitation learning. *ACM Computing Surveys*, 50(2):1–35.

IFR, I. F. o. R. (2018). How connected robots are transforming manufacturing.

Inagaki, T. (2001). Adaptive automation: Sharing and trading of control. *The Proceedings of the Transportation and Logistics Conference*, 2001.10.

Inagaki, T. et al. (2003). Adaptive automation: Sharing and trading of control. *Handbook of cognitive task design*, 8:147–169.

Intel (2020). Intel® realsense product family d400 series datasheet.

Intel Corporation (2021). Intel® realsense $^{TM}$ d400 series calibration tools.

Ittermann, P. and Niehaus, J. (2018). Industrie 4.0 und wandel von industriearbeit –revisited. forschungsstand und trendbestimmungen. In *Digitalisierung industrieller Arbeit*, page 33–60. Nomos Verlagsgesellschaft mbH & Co. KG.

Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., and Qu, R. (2019). A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868.

Johansson, G. (1973a). Visual perception of biological motion and a model for its analysis. *Perception & Psychophysics*, 14:201–211.

Johansson, G. (1973b). Visual perception of biological motion and a model for its analysis. *Perception & Psychophysics*, 14:201–211.

Kashi, S. and Levy-Tzedek, S. (2018). Smooth leader or sharp follower? playing the mirror game with a robot. *Restorative Neurology and Neuroscience*, 36(2):147–159.

Kavraki et al (2021). Open motion planning library:a primer.

Kidd, C. and Breazeal, C. (2005). Human-robot interaction experiments: Lessons learned. In *AISB'05 Convention: Social Intelligence and Interaction in Animals, Robots and Agents—Proceedings of the Symposium on Robot Companions: Hard Problems and Open Challenges in Robot-Human Interaction*, page 141–42.

Kleinsmith, A. and Bianchi-Berthouze, N. (2013). Affective body expression perception and recognition: A survey. *IEEE Transactions on Affective Computing*, 4(1):15–33.

Kopf, J., Rong, X., and Huang, J.-B. (2021). Robust consistent video depth estimation.

Kratzer, P., Bihlmaier, S., Midlagajni, N. B., Prakash, R., Toussaint, M., and Mainprice, J. (2021). Mogaze: A dataset of full-body motions that includes workspace geometry and eye-gaze. *IEEE Robotics and Automation Letters*, 6(2):367–373.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA.

Krüger, J., Lien, T., and Verl, A. (2009). Cooperation of human and machines in assembly lines. *CIRP Annals*, 58(2):628–646.

Kwan, J., Tan, C., and Cosgun, A. (2020). Gesture recognition for initiating human-to-robot handovers.

Lambert, J., Huzaifa, U., Rizvi, W., and LaViers, A. (2019). A comparison of descriptive and emotive labels to explain human perception of gait styles on a compass walker in variable contexts. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8.

Lambrecht, J. and Nimpsch, S. (2019). Human prediction for the natural instruction of handovers in human robot collaboration. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–6.

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., and Hager, G. D. (2017). Temporal convolutional networks for action segmentation and detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1003–1012.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, Z., Uemura, A., and Kiya, H. (2009). An fft-based full-search block-matching algorithm with ssd criterion.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). Feature pyramid networks for object detection.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, C. (2016). Ssd: Single shot multibox detector. In Leibe, B., Matas, J., Sebe, N., Welling, M., and eds), editors, *Computer Vision – ECCV 2016*, pages 21–37,. Springer International Publishing.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2015). SSD: single shot multibox detector. *CoRR*, abs/1512.02325.

Longuet-Higgins, H. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135.

Lorenz, T., Mortl, A., Vlaskamp, B., Schubo, A., and Hirche, S. (2011). Synchronization in a goal-directed task: Human movement coordination with each other and robotic partners. In *2011 RO-MAN*, page 198–203. IEEE, New York.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2.

Ludwig, T., Kotthaus, C., Stein, M., Pipek, V., and Wulf, V. (2018). Revive old discussions! socio-technical challenges for small and medium enterprises within industry 4.0. In *Proceedings of 16th European Conference on Computer-Supported Cooperative Work-Exploratory Papers*. European Society for Socially Embedded Technologies (EUSSET).

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation.

Malik, A. and Bilberg, A. (2019a). Developing a reference model for human-robot interaction. *International Journal on Interactive Design and Manufacturing*, 13(4):1541–1547.

Malik, A. A. and Bilberg, A. (2019b). Developing a reference model for human–robot interaction. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 13(4):1541–1547.

Marsland, S., Shapiro, J., and Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in neurons activity. *Bull. Math. Biophys*, 5:115–133,.

Mubin, O., Stevens, C., Shahid, S., Al Mahmud, A., and Dong, J.-J. (2013). A review of the applicability of robots in education. *Technology for Education and Learning*, 1(1).

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

Musić, S., Prattichizzo, D., and Hirche, S. (2019). Human-robot interaction through fingertip haptic devices for cooperative manipulation tasks. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–7.

Nelson, C. (2000). *Neural plasticity and human development: The role of early experience in sculpting memory systems*. Developmental Science.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA.

Noceti, N., Sciutti, A., and Rea, F. (2020). *Modelling Human Motion From Human Perception to Robot Design: From Human Perception to Robot Design*.

OLDER, M. T., WATERSON, P. E., and CLEGG, C. W. (1997). A critical assessment of task allocation methods and their applicability. *Ergonomics*, 40(2):151–171.

Pairet, E., Ardon, P., Mistry, M., and Petillot, Y. (2019). Learning generalizable coupling terms for obstacle avoidance via low-dimensional geometric descriptors. *IEEE Robotics and Automation Letters*, 4(4):3979–3986.

Papandreou, G., Zhu, T., Chen, L., Gidaris, S., Tompson, J., and Murphy, K. (2018). Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. *CoRR*, abs/1803.08225.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Peng, X. and Schmid, C. (2016). Multi-region two-stream r-cnn for action detection. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 744–759, Cham. Springer International Publishing.

Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, page 143–156, Berlin, Heidelberg. Springer-Verlag.

Piana, S., Staglianò, A., Odone, F., and Camurri, A. (2016). Adaptive body gesture representation for automatic emotion recognition. *ACM Transactions on Interactive Intelligent Systems*, 6(1):1–31.

Pratusevich, M., Chrisos, J., and Aditya, S. (2019). Quantitative depth quality assessment of rgbd cameras at close range using 3d printed fixtures.

Raina, D., Mithun, P., Shah, S. V., and Kumar, S. (2019). A novel image-based path planning algorithm for eye-in-hand visual servoing of a redundant manipulator in a human centered environment. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8.

Ranftl, R., Bochkovskiy, A., and Koltun, V. (2021). Vision transformers for dense prediction.

Redd, C. B. and Bamberg, S. J. M. (2012). A wireless sensory feedback device for real-time gait feedback and training. *IEEE/ASME Transactions on Mechatronics*, 17(3):425–433.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788,.

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149,.

Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.

Rizzolatti, G. and Sinigaglia, C. (2010). The functional role of the parieto-frontal mirror circuit: interpretations and misinterpretations. *Nature reviews. Neuroscience*, 11:264–74.

Romero, D., Bernus, P., Noran, O., Stahre, J., and Fast-Berglund, Å. (2016). The operator 4.0: Human cyber-physical systems & adaptive automation towards human-automation symbiosis work systems. In Nääs, I., Vendrametto, O., Mendes Reis, J., Gonçalves, R. F., Silva, M. T., von Cieminski, G., and Kiritsis, D., editors, *Advances in Production Management Systems. Initiatives for a Sustainable World*, pages 677–686, Cham. Springer International Publishing.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

Sandini, G. and Sciutti, A. (2018). Humane robots—from robots with a humanoid body to robots with an anthropomorphic mind. *J. Hum.-Robot Interact.*, 7(1).

Sciutti, A., Mara, M., Tagliasco, V., and Sandini, G. (2018). Humanizing human-robotinteraction: On the importance of mutual understanding. *IEEE Technology and Society Magazine*, 37(1).

Sherer, K. (1984). On the nature and function of emotion: A component process approach. In *Approaches to Emotion*, page 31.

Sheridan, T. B. (2016). Human–robot interaction: status and challenges. *Human factors*, 58(4):525–532.

Singh, P., Singh, V., Dutta, S., and Kumar, S. (2019a). Model amp; feature agnostic eye-in-hand visual servoing using deep reinforcement learning with prioritized experience replay. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8.

Singh, Y., Kher, M., and Vashista, V. (2019b). Intention detection and gait recognition (idgr) system for gait assessment: A pilot study. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, page 1–6. IEEE Press.

Sivic and Zisserman (2003). Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2.

Stanford Artificial Intelligence Laboratory et al. (2018). Robotic operating system.

Stäubli (2016). Trends: Man and machine.

Sultana, F., Sufian, A., and Dutta, P. (2020). Evolution of image segmentation using deep convolutional neural network: a survey. *Knowledge-Based Systems*, 201:106062.

Sze, V. (2020). Efficient computing for deep learning, ai and robotics.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329.

Tanaka, F., Isshiki, K., Takahashi, F., Uekusa, M., Sei, R., and Hayashi, K. (2015). Pepper learns together with children: Development of an educational application. In *IEEE-RAS International Conference on Humanoid Robots*.

Ullman, S. (1979). The Interpretation of Structure from Motion. *Proceedings of the Royal Society of London Series B*, 203(1153):405–426.

van den Bosch, A., van Dijck, J., Helberger, N., Heylen, D., Hindriks, K., Hoos, H., and Vossen, P. (2019). Artificial intelligence research agenda for the dutch research agenda for ai.

Vannucci, F., Sciutti, A., Lehman, H., Sandini, G., Nagai, Y., and Rea, F. (2019). Cultural dif- ferences in speed adaptation in human-robot interaction tasks. *Paladyn, Journal of Behavioral Robotics*, 10(1):256–266.

Varela, F., Thompson, E., Rosch, E., and Kabat-Zinn, J. (1991). *The embodied mind: Cognitive science and human experience. The embodied mind: Cognitive science and human experience*. MIT Press, Cambridge.

Varni, G., Volpe, G., and Camurri, A. (2010). A system for real-time multimodal analysis of nonverbal affective social interaction in user-centric media. *IEEE Transactions on Multimedia*, 12(6):576–590.

Wang, L., Gao, R., Váncza, J., Krüger, J., Wang, X., Makris, S., and Chryssolouris, G. (2019a). Symbiotic human-robot collaborative assembly. *CIRP Annals*, 68(2):701–726.

Wang, L., Gao, R., Váncza, J., Krüger, J., Wang, X., Makris, S., and Chryssolouris, G. (2019b). Symbiotic human-robot collaborative assembly. *CIRP Annals - Manufacturing Technology*, 68:701–726.

Weinland, D., Ronfard, R., and Boyer, E. (2011). A survey of vision-based methods for action representation, segmentation and recognition. *Computer Vision and Image Understanding*, 115(2):224–241.

Widder, D. G., Dabbish, L., Herbsleb, J., Holloway, A., and Davidoff, S. (2021). Trust in collaborative automation in high stakes software engineering work: A case study at nasa.

Wilhelm, B., Manfred, B., Braun, M., Rally, P., and Scholtz, O. (2016). Lightweight robots in manual assembly – best to start simply! examining companies' initial experiences with lightweight robots.

Willshaw, D. and Von Der Malsburg, C. (1976). How patterned neural connections can be set up by self organization. In *Proceedings of the Royal Society of London - Biological Sciences*.

Xu, G. and Zhang, Z. (2013). *Epipolar geometry in stereo, motion and object recognition: a unified approach*, volume 6. Springer Science & Business Media.

Yang, Y. and Ramanan, D. (2013). Articulated human detection with flexible mixtures of parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2878–2890.

Yuguchi, A., Inoue, T., Ricardez, G. A. G., Ding, M., Takamatsu, J., and Ogasawara, T. (2019). Real-time gazed object identification with a variable point of view using a mobile service robot. *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–6.

Zagoruyko, S., Lerer, A., Lin, T., Pinheiro, P. H. O., Gross, S., Chintala, S., and Dollár, P. (2016). A multipath network for object detection. *CoRR*, abs/1604.02135.

Zhang, L., Li, S., Xiong, H., Diao, X., Ma, O., and Wang, Z. (2019). Prediction of intentions behind a single human action: An application of convolutional neural network. In *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 670–676.

Zheng, C., Wu, W., Yang, T., Zhu, S., Chen, C., Liu, R., Shen, J., Kehtarnavaz, N., and Shah, M. (2021). Deep learning-based human pose estimation: A survey.

Zhou, D.-X. (2020). Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794.

# A APPENDIX

**Figure A.1**: More Vision Edge Cases

## COLOPHON

This document was typeset using LaTeX. The document layout was generated using the arsclassica package by Lorenzo Pantieri, which is an adaption of the original classicthesis package from André Miede.