UiA Faculty of Engineering and Science

Monocular Position Estimation of Occluded Industrial Workers

Andreas J. Akselsen

Harald I. Moldsvor

Supervisors

Atle Aalerud - Ph.D. research fellow at UiA

Joacim Dybedal - Ph.D. research fellow at UiA

Andreas Gromsrud - National Oilwell Varco

Erind Ujkani - National Oilwell Varco

University of Agder, 2019

Faculty of Engineering and Science

Department of Engineering Sciences

Contents

1 Introduction **3**
- 1.1 Problem description . . . 3
- 1.2 Requirements . . . 4
- 1.3 Known limitations . . . 4

2 State of the art **5**
- 2.1 Human detection . . . 5
- 2.2 Human pose estimation . . . 7

3 Theory **9**
- 3.1 Homogeneous coordinates . . . 9
- 3.2 Pinhole camera model . . . 9
- 3.3 Camera distortion . . . 12
- 3.4 Perspective-n-Point problem . . . 13
- 3.5 Position estimation . . . 13
- 3.6 3D position estimation . . . 14

4 Method **16**
- 4.1 Preprovided code . . . 16
- 4.2 Test site and equipment . . . 16
- 4.3 Camera calibration . . . 17
- 4.4 Human Pose Estimation . . . 19
- 4.5 Human body proportions . . . 24
- 4.6 Relative height estimation . . . 25
- 4.7 Weighted sum position . . . 26
- 4.8 Sensitivity analysis of position estimation . . . 28
- 4.9 Edge Computing . . . 28

5 Results **30**
- 5.1 Human Pose Estimation (HPE) . . . 30
- 5.2 Position sensitivity . . . 36
- 5.3 Position estimation . . . 37
- 5.4 Edge Computing - Nvidia AGX Xavier timing test . . . 44

6 Personnel tracking using human pose estimation in multi-camera setup **46**
- 6.1 Method and purpose . . . 46
- 6.2 Setup and results . . . 46

7 Future work **49**

8 Discussion **52**

9 Conclusion **54**

Bibliography **55**

A Calibration **58**

B 2D to 3D position estimation **59**

List of Figures

1.1 Conventional safety zone with break laser system . . . 3
1.2 Adaptive safety zone for the proposed system . . . 4

2.1 Working principle of the YOLO detector . . . 6
2.2 Working principle of the SSD detector . . . 7
2.3 Working principle of R-CNN networks . . . 7
2.4 Example of OpenPose results . . . 8
2.5 Example of results and dataset from DensePose . . . 8

3.1 Illustration of a pinhole camera . . . 10
3.2 Illustration of the common types of lens distortion . . . 12
3.3 World and image points correspondence . . . 13

4.1 UiA Industrial Robotics Lab . . . 17
4.2 HIKvision camera . . . 17
4.3 Checkerboard pattern used for calibration . . . 17
4.4 Intrinsic calibration images . . . 18
4.5 Reprojection errors . . . 18
4.6 Checkerboard placements . . . 18
4.7 Placements of room markers visible from the camera . . . 19
4.8 COCO Keypoints provided by OpenPose and its derivatives . . . 20
4.9 Architecture of the OpenPose neural network setup . . . 21
4.10 Illustration of the OpenPose process . . . 21
4.11 Failure to detect partially obscured person, when using bounding box detector for segmentation 22
4.12 A few frames from the 4.5 second video clip used for pose performance analysis . . . 23
4.13 Human proportions . . . 24
4.14 Fixed list of k elements sliding through measurements . . . 26
4.15 Deviations in position estimation for different keypoints . . . 26
4.16 Variance in position estimates . . . 27
4.17 Position estimates with biases included . . . 28
4.18 NVIDIA Jetson AGX Xavier embedded computing board . . . 29

5.1 Result of investigating relation between image scaling and time/accuracy . . . 30
5.2 Result of investigating relation between image area scaling and time/accuracy . . . 31
5.3 Example of false human detection . . . 32
5.4 Examples of false foot detection . . . 33
5.5 Resulting detection and confidence map in fake foot test, fake ankles circled . . . 34
5.6 Failure case: One person in front of another, different poses . . . 35
5.7 Inconsistency in detection when torso is occluded . . . 36
5.8 Position estimate, no occlusion . . . 37
5.9 Position estimate . . . 37
5.10 Pose estimate occlusion . . . 38
5.11 Position estimate . . . 38
5.12 Coveralls and helmet test, no occlusion . . . 38

5.13 Position estimate 38
5.14 Coveralls and helmet test, occlusion 39
5.15 Position estimate 39
5.16 Illustration of how new landmarks were calculated 40
5.17 Landmark verification 40
5.18 Position estimates, landmark verification 40
5.19 Results from dynamic test 41
5.20 Results from dynamic test 41
5.21 Error and velocity plot 42
5.22 Position estimation with short strides 43
5.23 Position estimation with long strides 43
5.24 Error and velocity plot for short stride 44
5.25 Error and velocity plot for long stride 44
5.26 Frequency test performed on the Xavier 45

6.1 Room setup and traced path during multicam test 47
6.2 Result after merging of systems with occlusions 47
6.3 Result from original multicamera setup with occlusions 48

7.1 Illustration of the PoseFlow tracking process 50
7.2 Illustration frames from the video with applied tracking 51

List of Tables

4.1 Ground truth values for landmarks 19
4.2 Measured keypoint heights 25
4.3 Measured keypoints ratio, keypoint/height 25

5.1 Result of testing effect on image scaling during pose processing, in the real program 31
5.2 Keypoint confidences from the fake ankle experiment 34
5.3 Result of pixel coordinate sensitivity test 36
5.4 Result of height sensitivity test 36
5.5 Estimated positions and absolute error 37
5.6 Estimated positions and absolute error 38
5.7 Estimated positions, no occlusion 39
5.8 Estimated position, verification test for landmarks 40

Nomenclature

ACS	Anti-Collision System
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CoP	Center of Projection
FAIR	Facebook AI Research
HPE	Human Pose Estimation
IRL	Industrial Robotics Lab
LPA	Low-Precision Arithmetic
MMC	Multi Machine Control
MPII	Max Planck Institut führ Informatik
NOV	National Oilwell Varco
PAF	Part Affinity Fields
PnP	Perspective-n-Point
R-CNN	Regional Convolutional Neural Network
SSD	Single Shot Detector
SVM	Support Vector Machine
YOLO	You Only Look Once

Preface

This thesis concludes a five year journey throughout our education at the University of Agder. The project has expanded our knowledge within the fields of computer vision and machine learning. We've enjoyed our time with this project and we are grateful for the opportunity to work with a such interesting problem.

The authors would like to extend our sincere gratitude to our supervisors, Ph.D research fellows Atle Aalerud and Joacim Dybedal. Their excellent guidance and assistance has helped us elevate the quality of this thesis. We would also like to thank Erind Ujkani and Andreas Gromsrud from National Oilwell Varco, for both providing us with this interesting thesis, and their support throughout the project. Finally we would like to thank Sondre Ripegutu for the collaboration where we merged ours and his thesis together. Sondre has also provided invaluable assistance in the form of discussions regarding problems we had and most importantly, friendly talks to take the mind of the thesis for a while.

Thanks to relatives and friends who kept me going, for the good introspective walks in the sun, and for promising to be my partners in workout, which I can pick up again now that the thesis is over. Thank you very much.

— Andreas J. Akselsen

I would like to thank the one who runs to meet me with a smile in the door everyday when I come home, my dog Lukas. Without him, finishing this thesis would be impossible. Oh, right, I should probably also thank my wife. Thanks.

— Harald I. Moldsvor

Abstract

In this thesis, a technique was developed to perform position estimation of occluded personnel in an industrial environment, using a single RGB camera. The system detects persons using Human Pose Estimation, which locates the joints of a person and determines their image coordinates in a given frame. An implementation of the OpenPose library was utilized for this purpose. It was demonstrated that the system can estimate 2D monocular human positions, even under occluded scenarios, with an accuracy of 20 cm. The system was implemented for a single person, with future extensions to multiple persons possible through added pose tracking. The system runs at 20 FPS on a workstation using an Nvidia 1080Ti GPU. It was demonstrated how the robustness can be improved by using multiple cameras and fusing their estimates, in a collaboration with the multicam project by Sondre Ripegutu. Pose inference was also tested on a Nvidia AGX Xavier, and it was demonstrated how it can perform pose inference at a rate of 10 FPS, increasing scalability by offloading processing, and adding redundancy to the system by using multiple nodes.

Chapter 1

Introduction

1.1 Problem description

National Oilwell Varco's (NOV) Multi Machine Control (MMC) is the state of the art drilling equipment [1], [2]. MMC enables one operator to operate a suite of fully automated drilling rig equipment. The machines on the platform are equipped with NOV's Anti-Collision Systems (ACS), [3] halting operation in case of emergency. However this is only for machine to machine collision. To avoid human and machine collision sensors are needed. A popular option is to utilize laser break beam sensors to create a bounding box in which humans are not allowed to operate within. This setup comes with a set of advantages and disadvantages. The disadvantage of being a fixed setup and safety zones cannot be changed depending on the situation. If the robot contains a traversing base, then the enclosed area will be excessively large compared to what is needed to secure the robot workspace. Figure 1.1 shows a person within the safety zone, which would cause a shutdown of operation. However, if coordinates of nearby persons are known, then an adaptive safety zone generated by a computer vision software, can be implemented, which moves along with the robot during operation. As shown in figure 1.2, the person is able to work in proximity to the robot without entering the safety zone. Another disadvantage with break laser beam is that it works in binary classification. It can't tell *what* has entered the area. With a computer vision based it is possible to classify that there is a human that entered the safety zone.

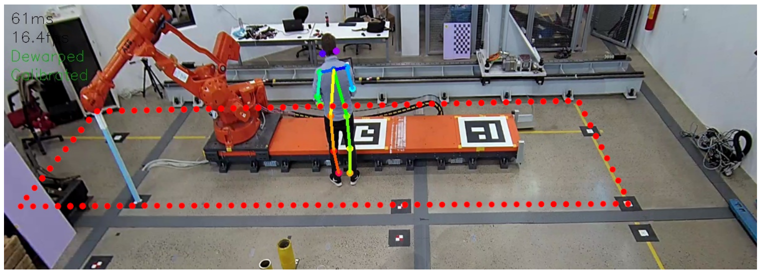
Figure 1.1: Conventional safety zone with break laser system

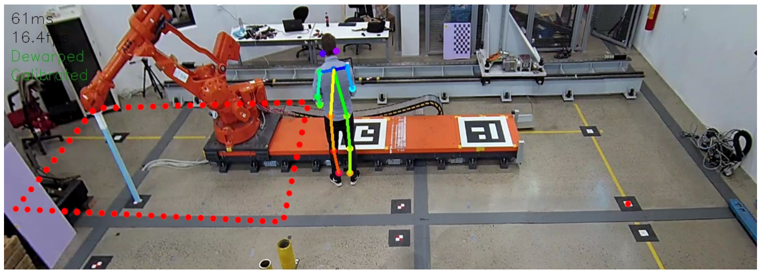

Figure 1.2: Adaptive safety zone for the proposed system

As of present, the system NOV has developed is capable of determining bounding boxes of persons in a 2D image, and calculates the planar coordinates based on the bottom of the bounding box, with an accuracy of less than 10 cm. However, this method assumes that the person is fully visible in the image, with no obstructions in the view. When the body is only partially visible, for instance, when blocked by equipment, crates or beams, the detection test fails.

The goal of this thesis is to develop a technique for estimating the planar coordinates of the person, even when the image view is partially obstructed. The estimation should be robust enough to be able to work in industrial settings where challenges such as persons wearing mono-colored coveralls and helmets might interfere.

1.2 Requirements

The position estimation must be able to work with the following criteria:

- Only a single camera is guaranteed to be present
- The person is partially obscured; for example, with only the upper body visible
- The system must work in an industrial environment
- The system ideally should work with a frame rate above 10 FPS

1.3 Known limitations

A system featuring only a single, stationary 2D camera cannot fully reconstruct a 3D position, unless assumptions about the environment and/or objects are made, given the lack of depth vision in such a setup. When the person is obstructed, this becomes even harder.

All tests in this thesis will concentrate on estimating the position of a single person. Ideas about extensions to multiple persons will be discussed, and the robustness of the human detector in the presence of multiple persons will be carried out as well.

Chapter 2

State of the art

2.1 Human detection

Human Detection is the problem of finding humans within a given image. It is a subclass of the more general problem of object detection, specifically trained to look for humans. In industrial applications, the solution algorithms should be capable of running in real time, so they can respond quickly to detected humans from a live video feed.

Real time object detection is a recent field of study, and all of these algorithms were developed in this millennium. The algorithms tend to use a machine learning component, which can be trained to recognize patterns in the data by feeding them categorized examples. In the classical algorithms developed in the 2000s, this usually involved a trainable classifier like AdaBoost or SVM (Support Vector Machine), which would process the data at the end of the pipeline. These classical algorithms were often based around the concept of *sliding windows*, where a frame is slided across multiple positions in the image, and performs feature detection on each obtained region [4].

The Viola-Jones algorithm was conceived in 2001, and was the first object detector capable of running in real time, although it was primarily aimed at face detection. It operates by sliding a window across the image and extracting features from each obtained region, before finally applying an AdaBoost classifier. [4] In 2005, Histogram of Oriented Gradients (HOG) [5] was published. This algorithm operates by calculating the gradient values in the image, and forming a histogram which can be processed by a trained classifier like SVM. This algorithm was developed with the goal of detecting pedestrians, making it well suited for human detection in general [5].

In recent developments and academic research, detection algorithms are usually based on deep learning. Unlike the simpler classifiers detailed above, deep learning uses a full *neural network* to process the data The most common type of neural network for object detection is the *convolutional neural network* (CNN). CNNs are well suited for image processing, as they can capture spatial relationships in two-dimensional data. [6] For more information of how neural networks learn and operate, the reader is advised to check out the overview at [7], or the MIT-published book on the subject, which is also freely available online [8].

The usage of deep learning and CNNs for object detection came into focus after the creation of AlexNet [9] in 2012, which went on to win the ILSVRC [10] (ImageNet Large Scale Visual Recognition Challenge) contest that same year. AlexNet combined CNNs with other recent tricks from the deep learning world, such as rectified linear units, pooling and dropout regularization, to cut down on processing time and classification errors. As a result, it was able to beat its competitors by a significant margin, and led to higher interest in deep learning as means of object detection [11].

Neural networks for object detection are commonly trained on publicly available sets of data, often for the purpose of comparing various approaches and algorithms. Examples of datasets include:

- Common Objects in Context (COCO) [12]
- Caltech Pedestrian Dataset [13]
- PASCAL Visual Object Classes (PASCAL-VOC) [14]

The deep learning detectors presented below can be broadly classified as *region based* or *single shot*. The region based detectors subdivide the image into a series of regions, which are then processed separately. The idea of region-based detectors was derived from the sliding window concept used in Viola-Jones, but lets the algorithm determine suitable regions at runtime, rather than investigating every possible region in the image. These networks tend to be more accurate than single-shot detectors, but at the cost of slower processing time. [15] Single shot detectors operate directly on the full image, and predict the bounding boxes and categories of objects in a single step. This cuts down on processing time at the cost of accuracy. In particular, these detectors often struggle to detect smaller objects [15].

You Only Look Once (YOLO) [16] is a single shot detection algorithm which is well suited for real time processing. YOLO operates by dividing the image into a uniform grid of cells. The grid is processed by a neural network, and for each cell, probabilities are calculated of whether it contains an object of a given category. If an object spans multiple sections, the middle section is considered the owner of the object. Coordinates and width/height of bounding boxes are also predicted as part of the process. The process is illustrated in figure 2.1. The original YOLO paper was published in 2015, the algorithm has since gone through several iterations. The latest revision, YOLOv3, was released in 2018 [17].

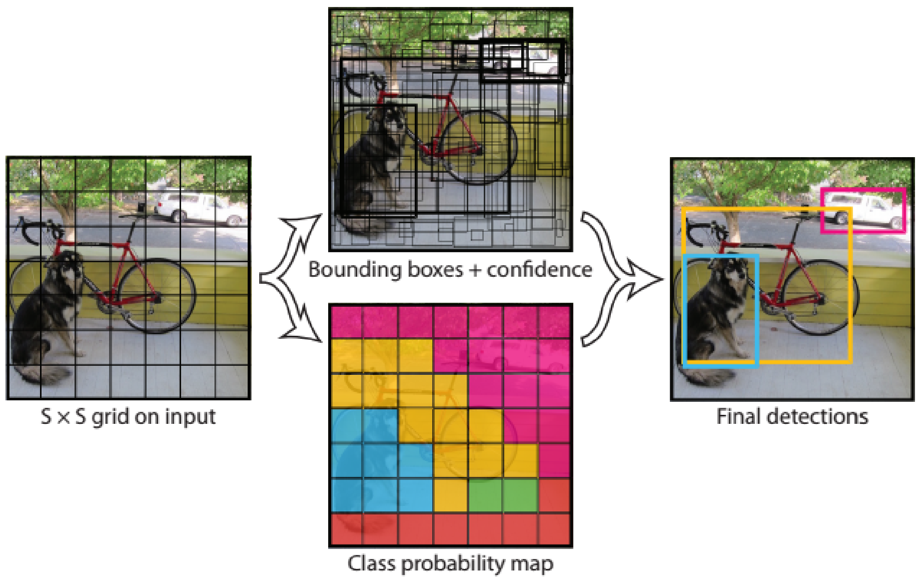

Figure 2.1: Working principle of the YOLO detector (grid size may be adjusted) [16] © 2011 IEEE

An alternative to YOLO is a detector simply named Single Shot Detector (SSD) [18], which was published in 2016. For each plausible location of an object in the image, SSD will attempt to fit multiple bounding boxes of various sizes and aspect ratios, assigning each box a confidence probability. A chain of CNNs process the image, with the later layers given most of the responsibility for feature detection. Versions of SSD are usually categorized by the accepted input dimension of the image, such as SSD300 or SSD512, where higher resolutions increase accuracy at the cost of slower speed. In general, SSD tends to run fast and perform well in real time on tested sets, as long as the objects are not too small [19].

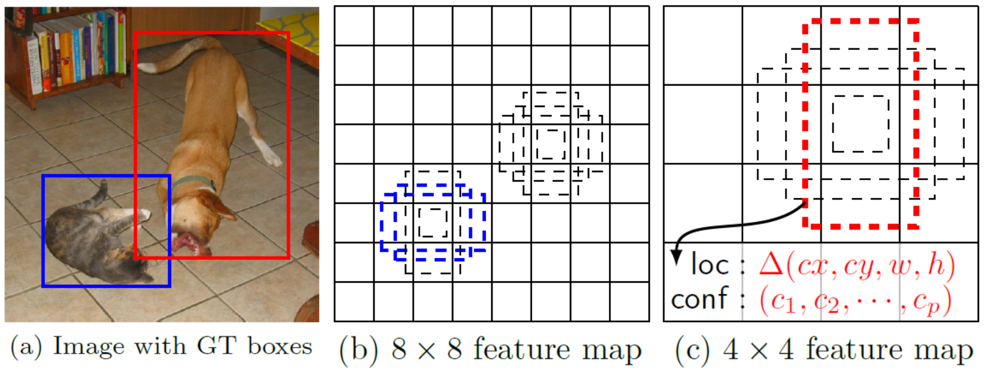
loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

Figure 2.2: Working principle of the SSD detector[18]

Of the region based detectors, one common class is the R-CNN (Regional Convolutional Neural Network) family of algorithms. The working principle of such networks is illustrated in figure 2.3. After dividing the image into regions, they are processed separately by neural networks to detect occurrences of predetermined categories. The regions are usually warped into squares before processing [20]. The original algorithm determined regions through selective search [20], but the latest revisions use a CNN to extract a feature map, which in turn hints at which regions to use [21], [22]. The original release of R-CNN in 2013 [20] ran very slowly, which made it unsuitable for realtime processing. Later updates have made huge improvements in processing speed, and the 2017 revision, Faster R-CNN [22], can run several checks per second, making it possible to employ it for realtime detection [23]. Several offshoots of R-CNN exist, such as R-FCN [24] for slightly faster performance, and Mask R-CNN [25], which also determines which image pixels lie inside the person.

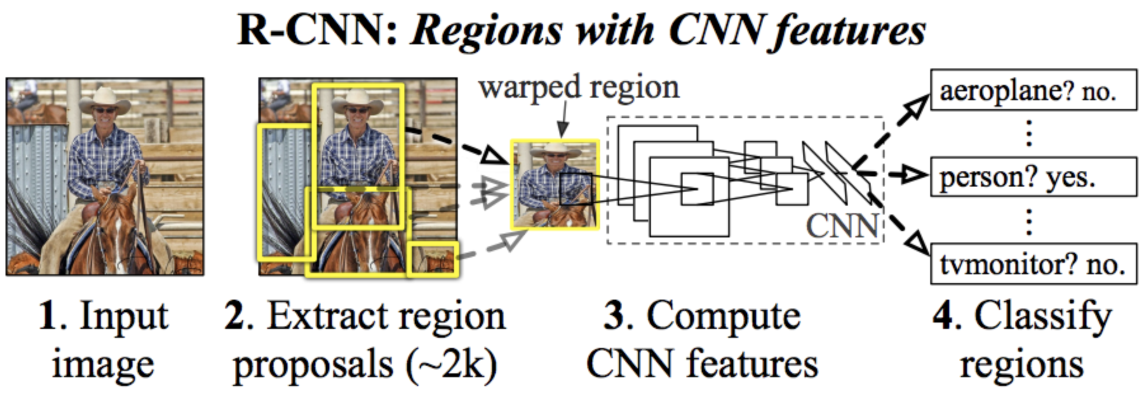

Figure 2.3: Working principle of R-CNN networks [20] © 2014 IEEE

2.2 Human pose estimation

Human Pose Estimation (HPE) is a sub field within human detection where the task is to correctly identify the joints and pose of one or multiple humans.

HPE has seen significant improvements in the recent years due to the use of CNNs and more high quality training data. With the rapid development, the state of the art changes frequently. Tracking what is the current state of the art is done by checking the leaderboards on the benchmarking sets. Two of the most popular benchmarking sets are the aforementioned COCO and the Max Planck Institute Informatik (MPII) human pose dataset [26], which is used in their project *PoseTrack* [27]. These two datasets are so called sparse datasets, meaning that they only point out a few keypoints like head, shoulder, knee, elbow etc. From this a "stickman" figure can be created to display the human pose estimate. Several academically developed frameworks have surfaced in recent years to solve human estimation problems, including OpenPose [28] and AlphaPose [29].

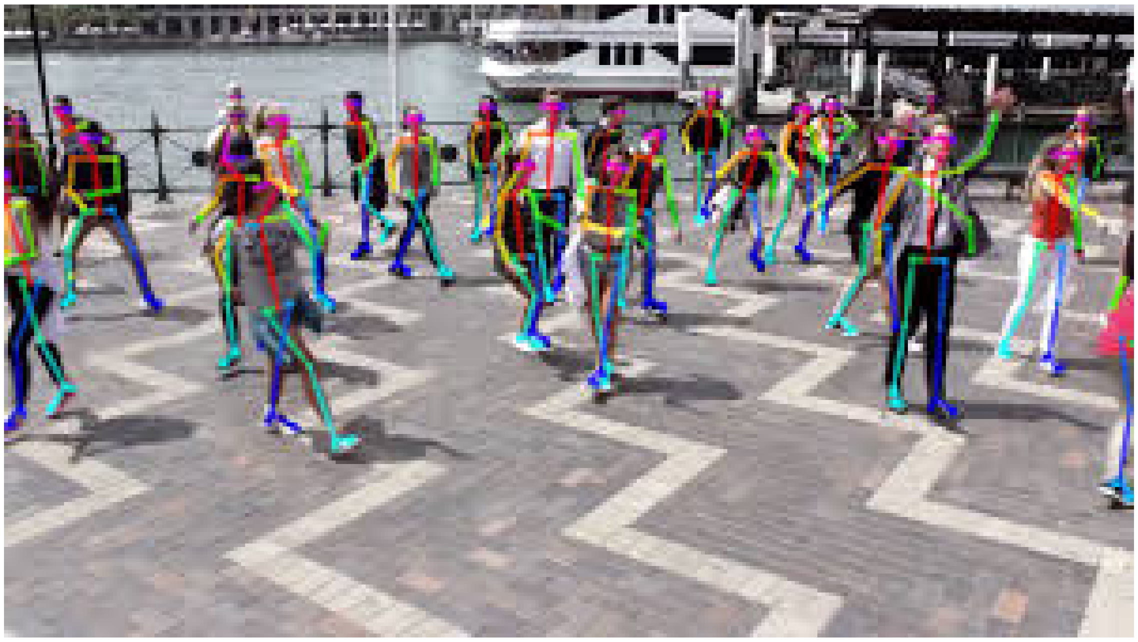

Figure 2.4: Example of OpenPose results [28, p. 1] @ 2018 IEEE

Recently Facebook AI Research (FAIR) released DensePose-COCO dataset and their proposal DensePose-RCNN. As described in their article, "DensePose-COCO dataset establishes a dense correspondence between points on a 2D image and points on a 3D surface-based representation of the human body." [30]. This means that from a 2D image, it is now possible to create a 3D mesh of a person.

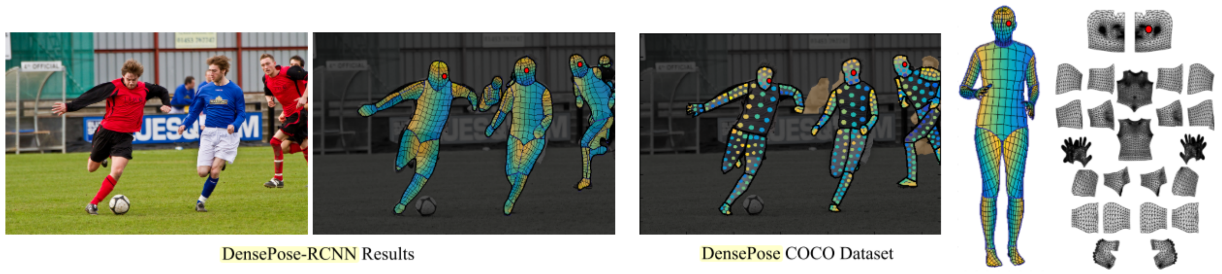

Figure 2.5: Example of results and dataset from DensePose [30]

Chapter 3

Theory

3.1 Homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \tag{3.1}$$

Homogeneous coordinates are expressed as scaled versions of their Cartesian equivalents, with the scaling factor provided as a fourth component. Depending on the type of calculation, the scaling factor may be constant, or it may be variable and calculated as part of the process. In cases where the factor can be set freely, it is usually equal to 1.
Homogeneous coordinates have certain advantages in this context:

- It allows for translations to be described using matrices
- When performing distance calculations derived from scaled triangles, as is done in 2D projection, the resulting formula contains a division by a variable length. By choosing this length to be the scaling factor and working with the scaled values, the transformations can be expressed using matrices, only dividing by the factor at the very end of the process.

3.2 Pinhole camera model

The Pinhole camera model is often used for a simple and yet efficient camera model. It works on the assumption that the aperture is a infinitely small point and no lenses are used to focus the incoming light. Figure 3.1 shows a simplified camera. At the center of the coordinate axis is the Center of Projection (CoP), which represents the aperture. Perpendicular to the principal axis is the imaginary image plane, the distance between CoP and image plane is called the focal length (f). The reason for it being imaginary is due to the actual image plane is behind the CoP and the image is formed upside down. To aid visualization the imaginary image plane is used. Inspiration for figure 3.1 is from Peter Corke's book *Robotics, Vision and Control: Fundamental Algorithms in MATLAB* [31].

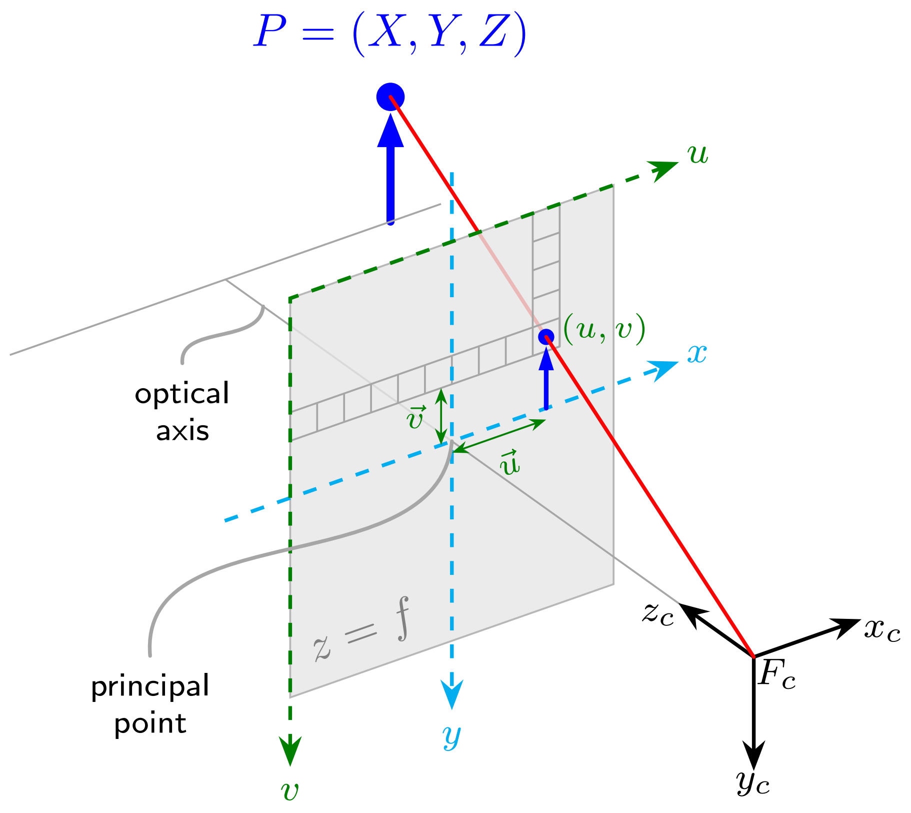

Figure 3.1: Illustration of a pinhole camera

The 3D point $P(X, Y, Z)$ is projected onto the image plane at point $P_c(u, v)$. Both of these points form similar triangles in respect to the CoP and principal axis. Therefore, using the laws of similar triangles the following relations can be found.

$$\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y} \tag{3.2}$$

Which in return can be solved for both u and v

$$u = \frac{fX}{Z} \tag{3.3}$$

$$v = \frac{fY}{Z} \tag{3.4}$$

Now using homogeneous coordinates for P_c, it can be rewritten as

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.5}$$

This transformation does not account for the origin in the image plane. Therefore it is needed to offset the point with the translation vector $[c_x, c_y]^T$.

$$u = \frac{fX}{Z} + c_x \tag{3.6}$$

$$v = \frac{fY}{Z} + c_y \tag{3.7}$$

Resulting in the following equation

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.8}$$

Now equation (3.8) transforms a 3D position to 2D image coordinates. However the 2D image coordinates are expressed in meters. To convert the position into pixels the pixels per meter (m) constant is needed. Typically, the pixels in a modern camera are square. However to generalize the problem, the pixels can be assumed to be rectangle and therefore it is needed one constant for each dimension; m_u and m_v.

$$u = m_u \frac{fX}{Z} + c_x \tag{3.9}$$

$$v = m_v \frac{fY}{Z} + c_y \tag{3.10}$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} m_u f & 0 & m_u c_x \\ 0 & m_v f & m_v c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.11}$$

Or in a more easily read format.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.12}$$

$$P_c = KP \tag{3.13}$$

The K matrix is the intrinsic matrix of the camera.
The second important parameter of the pinhole camera model is the extrinsic matrix. Extrinsic parameters describe the pose of the camera relative to the global origin. If the camera is moved or rotated, these parameters must be recalculated.
The overall camera model can be described in the form shown in equation 3.14.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Intrinsic matrix}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{\text{Extrinsic matrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.14}$$

where:

- (x,y,z) is the position of the point in global frame
- (u,v) is the projected position of the point in the image, measured in pixels
- s is an unknown scaling factor
- (f_x, f_y) are the focal lengths of the lens in x and y direction, measured in pixels
- (c_x, c_y) is the principal point of the image, usually near the center, measured in pixels
- (r_{11}, r_{12}, ..., r_{33}) make up a rotation matrix, describing the rotation of the camera relative to global frame
- (t_x, t_y, t_z) is the position of the camera in global frame

3.3 Camera distortion

The above camera model assumes that the transformation maps straight lines in the scene to straight lines in the image. However, for the kind of surveillance cameras employed in this project, this is not the case. Real lenses and cameras do not behave completely according to the assumptions of the pinhole camera model, causing deviations between assumed and actual behavior. This difference is called *distortion.* Wide angle lenses are especially vulnerable to this kind of behavior.

Distortion can often be corrected in software, assuming that a model of the distortion displacement can be created. In the OpenCV and MATLAB calibration tools, this is performed as part of the intrinsic calibration.

The most common type of distortion, radial distortion, is caused by the curvature in the image. The most common types of distortion are:

- Barrel distortion: Straight lines in the scene bend away from the center of the image. Also known as fish-eye
- Pincushion distortion: Straight lines in the scene bend towards the center of the image

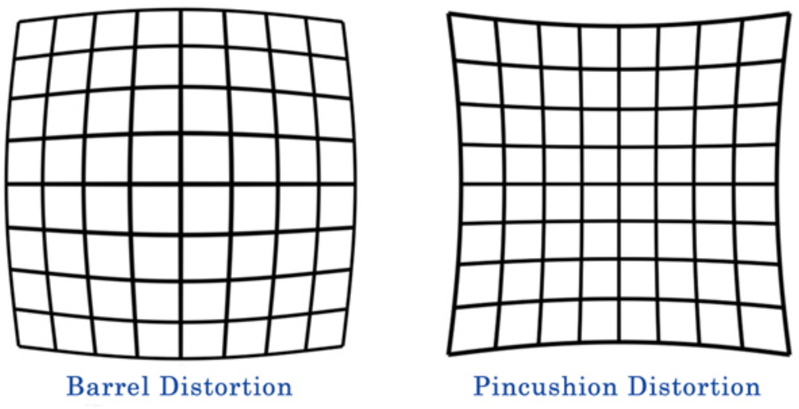

Figure 3.2: Illustration of the common types of lens distortion [32]

In addition, tangential distortions happen when the lens plane is not parallel to the image plane, causing the image to get stretched in some direction. OpenCV and MATLAB calibration tools will usually assume it to be zero, unless especially told otherwise.

Overall, the distortion model can be parameterized as follows: [33]

$$x_u = x\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6} + 2p_1xy + p_2(r^2 + 2x^2) \tag{3.15}$$

$$y_u = y\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6} + p_1(r^2 + 2y^2) + 2p_2xy \tag{3.16}$$

where

- (x_u, y_u) is the desired, undistorted position
- (x, y) is the originally obtained, distorted position
- r is the distorted distance from the principal point, $r = \sqrt{x^2 + y^2}$
- $(k_1, k_2, k_3, k_4, k_5, k_6)$ are radial distortion coefficients, to be determined during calibration
- (p_1, p_2) are tangential distortion coefficients, to be determined during calibration

3.4 Perspective-n-Point problem

The Perspective-n-Point problem (PnP) is the problem of estimating the extrinsic parameters of a camera, using a set of visible points with known global positions. This has the advantage of aligning the global coordinate system according to these points, making it easier to perform accuracy tests.

To solve the PnP problem, a set of n 3D points in the world frame and their corresponding points in the image plane is required. Figure 3.3 shows correspondence between world and image points where a camera has been rotated and translated. The minimal form of PnP problem is when $n = 3$, then it is called the P3P. This is solvable, but yields multiple solutions, so to remove ambiguity using $n \geq 4$ is advised. For further information on how the PnP is solved, the reader is advised to read the paper *EPnP: An Accurate O(n) Solution to the PnP Problem* by *V. Lepetit, et. al.* [34].

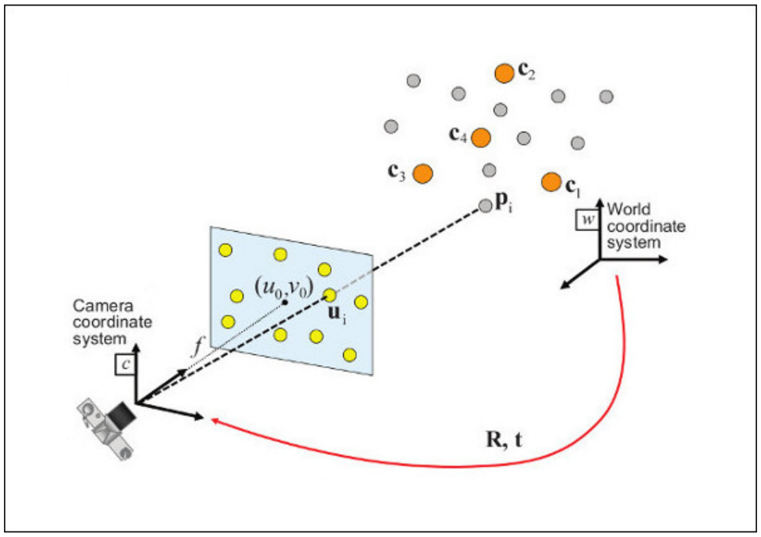

Figure 3.3: World and image points correspondence [35]

3.5 Position estimation

As previously discussed in section 3.2, the pinhole camera model is a function that maps points in $\mathbb{R}^3$ down to an image in $\mathbb{R}^2$. However the inverse is not directly possible, due to the loss of a dimension when the function went from $\mathbb{R}^3$ to $\mathbb{R}^2$. In order to estimate a position, one of the coordinates must be known. Since it is desired to estimate the x, y coordinates, the z coordinate must be known.

One common method is setting the z value to 0, which represents the floor. This is a good method when it is established that the measured object is on the floor. However when the object is partly occluded and the lower part cannot be seen, this method fails. Therefore a generalized solution that can take an arbitrary z value is needed. Equation 3.17 shows the pinhole camera model equation presented with slight modifications in notation.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = C \left(R \begin{bmatrix} X \\ Y \\ Z_{const} \end{bmatrix} + t \right) \tag{3.17}$$

Where:

- s - Scaling factor
- C - Intrinsic matrix

- R - Rotation matrix
- t - Translation vector

Firstly, the inverse of the rotation and intrinsic matrix is multiplied on both sides.

$$R^{-1}C^{-1}s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}X\\Y\\Z_{const}\end{bmatrix} + R^{-1}t \tag{3.18}$$

To ease readability, both left- and right-side of the equation are stored in variables $\mathbf{A}, \mathbf{B}$

$$\mathbf{A} = R^{-1}C^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix} \tag{3.19}$$

$$\mathbf{B} = R^{-1}t \tag{3.20}$$

$$\mathbf{A}s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}X\\Y\\Z_{const}\end{bmatrix} + \mathbf{B} \tag{3.21}$$

Now the scaling factor s can be calculated using the last row in the matrices.

$$s = \frac{Z_{const} + \mathbf{B}_{2,0}}{\mathbf{A}_{2,0}} \tag{3.22}$$

Finally the world coordinates X,Y can be calculated.

$$\begin{bmatrix}X\\Y\end{bmatrix} = R^{-1}\left(s\,C^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix} - t\right) \tag{3.23}$$

3.6 3D position estimation

The aforementioned position estimation method only works when the person stands on the floor. If the person stands on a chair or ladder or in more general terms, the person stands with a distance of Δz above the ground, then position estimate will be wrong. However, with a few assumptions about the measured person, then the full 3D position of the person can be estimated by measuring two points. The following assumptions are:

- The points have the same x and y coordinates.
- The z distance between the points are known.
- Intrinsic and extrinsic parameters are known.

In this example the two measured points will be on the top of the head and the bottom of the feet of the person. Since the person stands above the ground, the feet coordinates will be at height Δz and the head at $\Delta z + z$. Where z is the known height of the person.

Now both positions of the coordinates can be calculated with the following equations:

$$p_1 = R^{-1} \left((\Delta z + B[2,0]) \, C^{-1} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} - t \right) \tag{3.24}$$

$$p_2 = R^{-1} \left((z + \Delta z + B[2,0]) \, C^{-1} \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} - t \right) \tag{3.25}$$

Now the output from p_1 and p_2 will look the following:

$$p_1 = \begin{bmatrix} \alpha_1 \cdot \Delta z + \beta_1 \\ \alpha_2 \cdot \Delta z + \beta_2 \\ \Delta z \end{bmatrix} \qquad p_2 = \begin{bmatrix} \alpha_3 \cdot \Delta z + \beta_3 \\ \alpha_4 \cdot \Delta z + \beta_4 \\ \Delta z + z \end{bmatrix} \tag{3.26}$$

Here α_n and β_n are stand in variables for calculated coefficients. Since it is assumed that p_1 is in the same x and y position as p_2, then the it is possible to take the first element of each vector and calculate Δz.

$$\alpha_1 \cdot \Delta z + \beta_1 = \alpha_3 \cdot \Delta z + \beta_3 \tag{3.27}$$

$$\Delta z = \frac{\beta_3 - \beta_1}{\alpha_1 - \alpha_3} \tag{3.28}$$

Now this Δz variable can be used to estimate the full 3D position, which is done by inserting the now known value into equation 3.24 or 3.25.

Chapter 4

Method

4.1 Preprovided code

As part of the project material, a Python deep learning library for human detection was supplied by NOV. The library comes with two detectors, one based on Faster R-CNN and one based on SSD. The library estimates the bounding boxes of objects in the picture, and provides confidence values for each bounding box. The image coordinates of the boxes are also available, as normalized values between 0 and 1.
If the body is fully visible, the body coordinate corresponds to the middle of the bottom of the bounding box. This is used by the prewritten code to produce an estimate of the 2D floor position of the person.

4.2 Test site and equipment

It is desired to test the system in an environment that resembles an industrial workplace. Therefore a test site was setup in the UiA Industrial Robotics lab (IRL). This lab contains three industrial robot arms which can be used to occlude a human and provide test scenarios to see how well the system can detect humans near dangerous robot arms.

In IRL there are several landmarks which have been precisely measured. Which enables the group to have a reference to a coordinate system. The landmarks can be seen on the floor in figure 4.1.

Video of the IRL lab is captured with three HikVision RGB IP cameras (figure 4.2) that are installed in different corners in the IRL. The cameras stream at a resolution of 1920x1080 pixels and 20 FPS with an 80° field of view. The cameras can output at a higher resolution, but due to computation power limits, it was chosen not to.

The entire system runs on a computer running a Windows 10 operating system, with an Intel Xeon W-2123 CPU, Nvidia GTX 1080Ti GPU and 32 GB of RAM. The software is programmed in Python 3.6, and deep learning calculations are performed using the TensorFlow library (version 1.12), which employs the NVIDIA CUDA 9.0 and cuDNN 7.1 libraries for GPU calculations.

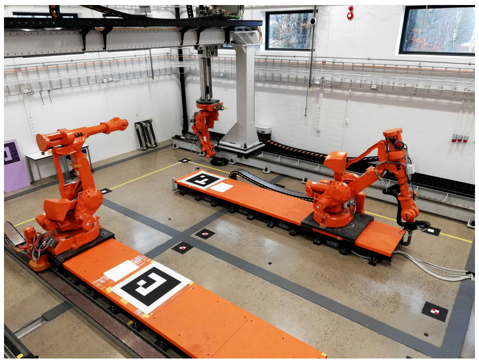

Figure 4.1: UiA Industrial Robotics Lab

Figure 4.2: HIKvision camera

4.3 Camera calibration

Camera calibration requires two steps: Intrinsic and extrinsic calibration, which respectively determines the intrinsic and extrinsic parts of the camera matrix. Intrinsic calibration depends only on the lens, and remains constant as long as the camera zoom/focal length is not changed. Distortion coefficients are also calculated as part of this process. Extrinsic calibration depends on the placement and orientation of the camera, requiring the process to be repeated over if the camera is moved and/or rotated.

Intrinsic calibration can be done with different types of patterns, this project uses a checkerboard of white and black fields, as shown in figure 4.3. The checkerboard features a 9x6 layout, where each field has a side length of 150.2 mm. A series of images must be taken, where the checkerboard is placed in different parts of the image, as well as with different orientations. A calibration tool can detect the relative corner positions between the fields of the checkerboard, and use this to estimate lens properties. A total of 100 images were taken in the IRL lab, simulating various checkerboard placements and orientations was performed by moving the checkerboard around and panning/tilting the lens. A selection of some of the images are shown in figure 4.4.

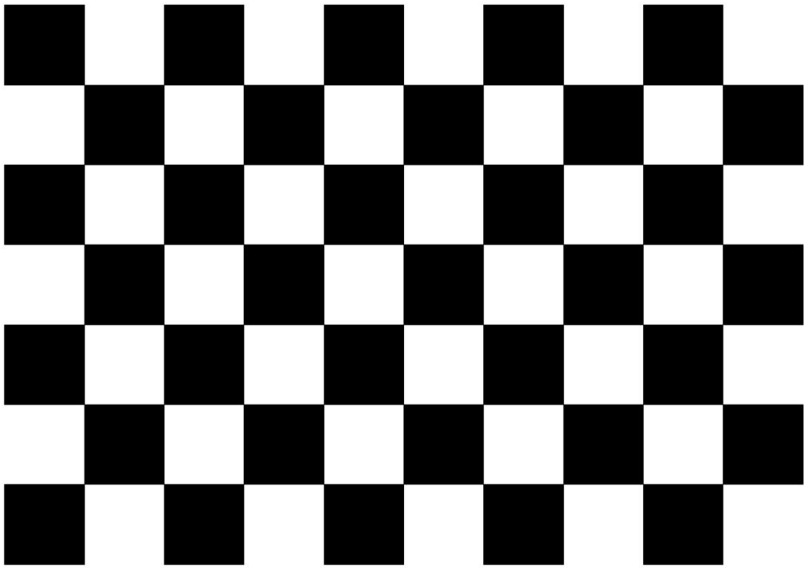

Figure 4.3: Checkerboard pattern used for calibration

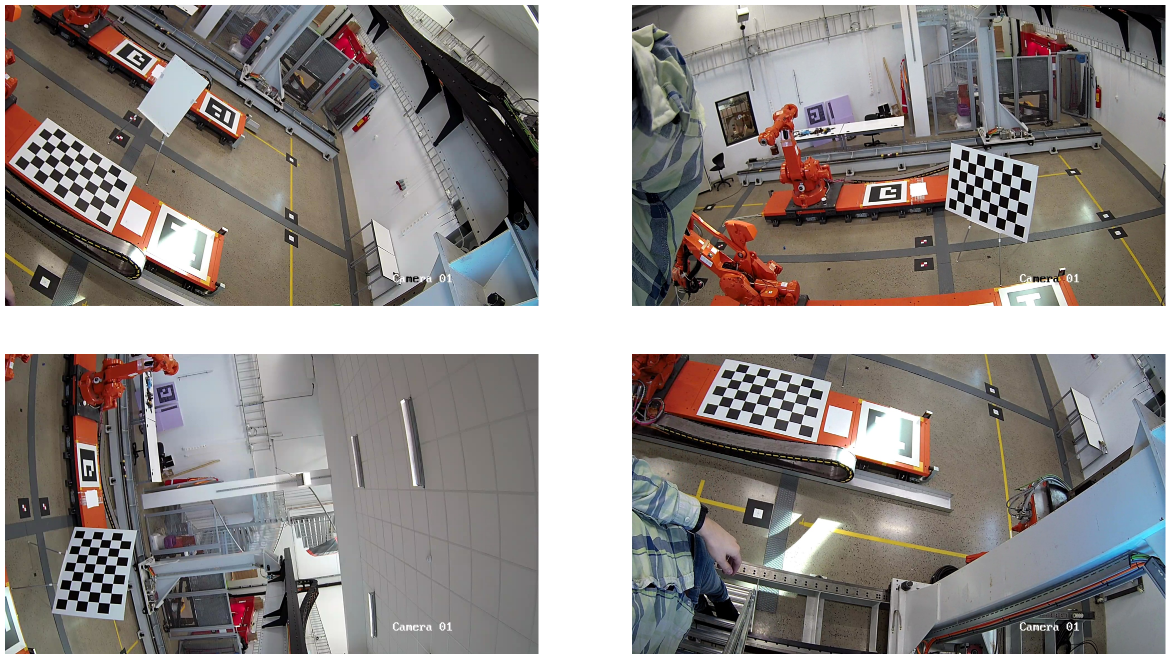
Figure 4.4: Intrinsic calibration images

Afterwards, the images were analyzed using the MATLAB calibration tool. In addition to performing intrinsic calibration, this tool analyzes the precision of the images and calculates the reprojection error for each image. The reprojection error is the average 2D distance between each checkerboard point, and the predicted position of that point in the model. Images with high error were automatically rejected and omitted from the result. The resulting reprojection errors and checkerboard placements are shown in figures 4.5, 4.6

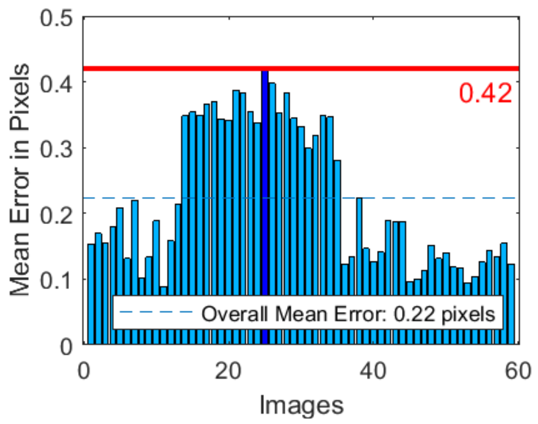

Figure 4.5: Reprojection errors

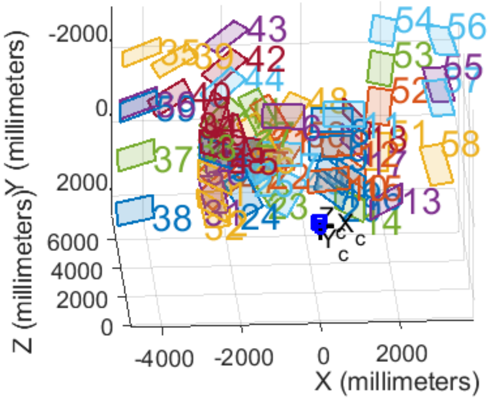

Figure 4.6: Checkerboard placements

Extrinsic calibration can be performed in a variety of ways. In this project, it is advantageous to have a known global reference frame which the accuracy can be tested against. Those extrinsic parameters are found by solving the PnP problem as described in the theory section, using the set of markers in the room. The coordinates of the markers were known in advance with high precision, and image coordinates were obtained manually from the resulting camera images. From this process, extrinsic parameters with respect to the global system can be obtained. Figure 4.7 shows the relative placements of the visible markers; actual visibility varied over the course of the project, depending on the placement of robots and other equipment in the lab. Table 4.1, shows the ground truth values for each keypoint.

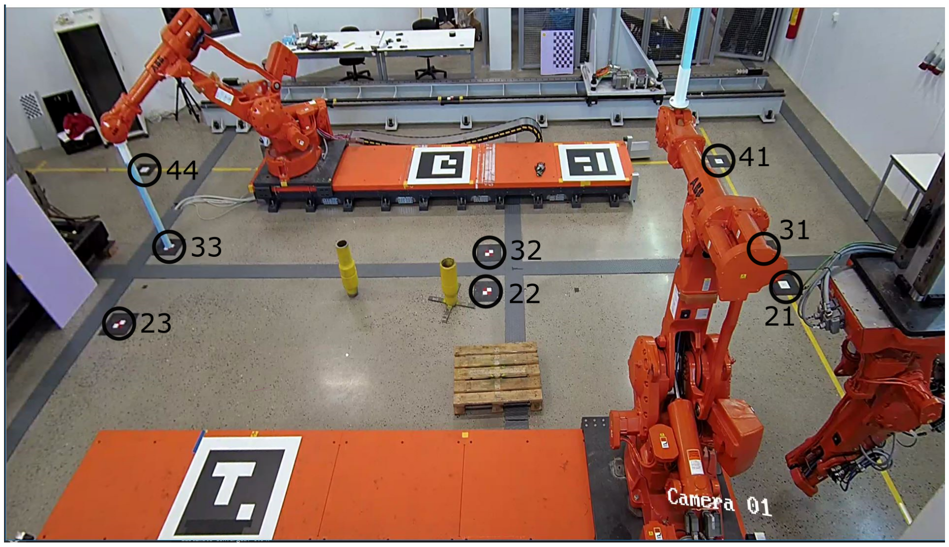

Figure 4.7: Placements of room markers visible from the camera

ID	x	y	unit
21	4.712	1.999	$[m]$
22	4.693	5.299	$[m]$
23	4.280	9.204	$[m]$
31	5.252	2.000	$[m]$
32	5.275	5.288	$[m]$
33	5.417	9.235	$[m]$
41	6.999	1.999	$[m]$
44	6.996	10.380	$[m]$

Table 4.1: Ground truth values for landmarks

4.4 Human Pose Estimation

Human Pose Estimation (HPE) is the process of discovering the location of humans and their various body parts in an image. In other words, in addition to discovering persons in the image, it also finds the precise image locations of feet, hands, shoulders and other parts. Based on the results of pose estimation, a stickman figure of the person can be drawn onto the image, so it matches the pose of the actual person. In theory, this process can be used to estimate pose for any object with a predictable shape, including various types of animals and machines, even bicycles. That said, pose estimation systems are usually trained to detect humans and their various joints.

Pose estimators comes with differing amounts of detail, with the most detailed ones being able to produce a full 3D mesh of a person. The simpler and faster ones look for a set of predetermined *keypoints*, which correspond to the set of joints/body parts implemented in the HPE. Out of these estimators, one of the most commonly used family of HPEs is OpenPose[28]. OpenPose has multiple, independently developed implementations, including the reference implementation [36], the simpler and lighter tf-pose-estimation [37] and the newer OpenPose Plus [38].

Several models for body keypoints exist, with the commonly used COCO and MPII datasets comes with their own custom keypoint models. The OpenPose HPE and its derivatives use a custom variant of the COCO [39] keypoint set, containing the following parts and their respectively keypoint identifiers:

• Nose	[0]
• Neck	[1]
• Shoulders	[2, 5]
• Elbows	[3, 6]
• Wrists	[4, 7]
• Hips	[8, 11]
• Knees	[9, 12]
• Ankles	[10, 13]
• Eyes	[14, 15]
• Ears	[16, 17]

For the keypoint types that come in both left and right versions, such as ears and shoulders, their side will also be classified by the HPE. These points are labelled as left and right according to what the persons themselves considers to be *their* left and right parts, and this is done correctly regardless of which side the person is seen from. All of the keypoints and their ordering are illustrated in figure 4.8.

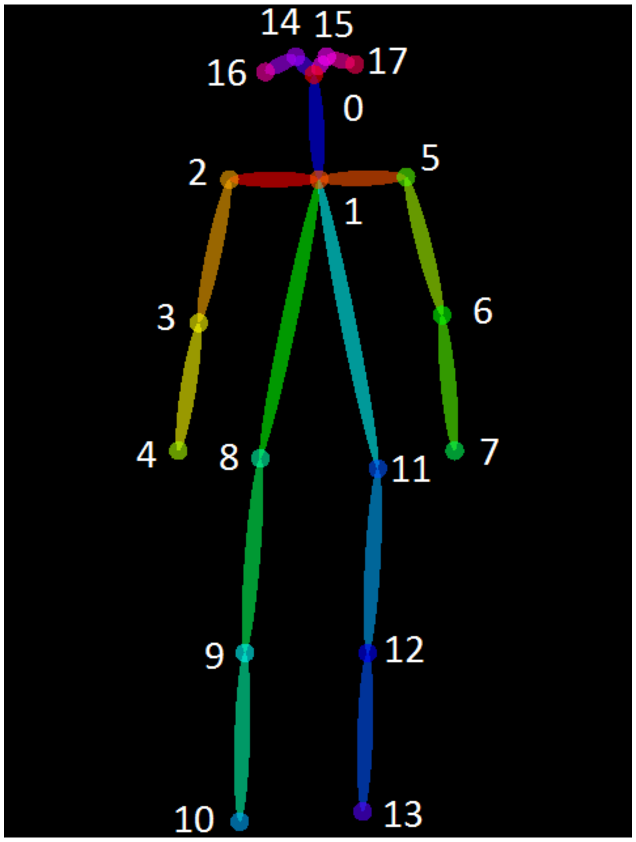

Figure 4.8: COCO Keypoints provided by OpenPose and its derivatives [36] © 2018 IEEE

In the OpenPose architecture, there are two general stages to the pose detection process. It must detect the various joints, identifying the positions and types of each joint. Second, it must estimate the limbs between joints, tying the parts together and distinguishing different humans.

To estimate keypoints, the pose estimator produces a set of confidence maps, one for each type of keypoint. Each pixel in the image is assigned a confidence value, corresponding to the likelihood of the pixel belonging to a keypoint of given type. Small regions of high confidence are likely to contain the desired keypoint. If there are multiple persons in the image, the confidence map may contain several regions where the confidence is high, corresponding to the likely location of each keypoint type for each person [28].

Grouping related keypoints is performed using the technique of Part Affinity Fields (PAF) [28]. One affinity field is created for each possible type of limb, where a limb corresponds to a connected pair of keypoints. For each pixel in the image, the affinity field is assigned a 2D vector indicating the strength and orientation of a limb present at that position. As with keypoints, there may be more than one limb of each type if the image contains multiple persons.

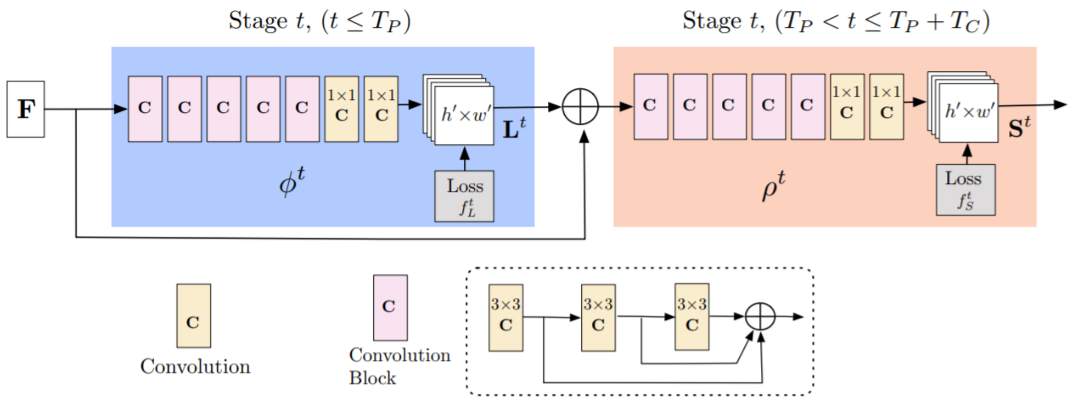

Figure 4.9: Architecture of the OpenPose neural network setup [28, p. 3] © 2018 IEEE

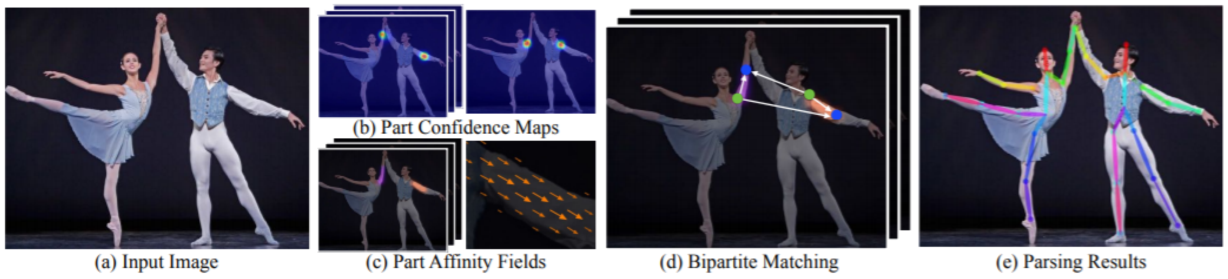

Figure 4.10: Illustration of the OpenPose process [28, p. 2] © 2018 IEEE

In the OpenPose architecture, the confidences and affinity fields are formed in a multistage CNN neural network. This is illustrated in figure 4.9. The network goes through the same stages multiple times, refining the accuracy of the predictions at each step. Greedy bipartite matching is carried out on the resulting data to form the final human and pose estimations [28]. The full process is illustrated visually in figure 4.10.
This project uses the tf-pose-estimation HPE [37], which is a small and fast Python implementation of the OpenPose architecture. The library is simple to set up and use, and integrates well with the existing Python/TensorFlow setup, while still running fast and efficiently.

tf-pose-estimation can be trained using a multitude of datasets, although it comes with two pretrained models: CMU and MobileNet-thin. CMU is the model which shipped with the original OpenPose implementation. MobileNet-thin is a small and very fast model, which is well suited for embedded and/or time-critical applications. According to the tf-pose-estimation module developer [37], mobileNet-Thin features a CNN structure optimized according to the MobileNet paper [40], and the result runs about seven times faster than the CMU model under tf-pose-estimation. Using the 1080Ti, and slightly shrinking the image before pose processing, tf-pose-estimation with MobileNet-Thin can run at a processing time of 30 ms per frame. The fast performance of the library makes it possible to run the full program at an average of 20 FPS, which is well suited for real time processing, and leaves room for more features to be added without the program becoming too slow for real-time usage. The latter model also have less restrictive terms concerning commercial usage, as discussed in section 8.

For each frame, the HPE outputs a series of entries corresponding to each detected person in the image. Each entry contains the following:

- A list of keypoints detected on the person, including the type of keypoint, the normalized location in the image, and a confidence value between 0 and 1. Only the discovered keypoint types are returned
- A score indicating the likelihood that the detected entity is a person. This value is not normalized and will usually take on values higher than 1. The true nature of this score is not fully known, but is assumed to grow with the number of detected keypoints and limbs on a person, as well as their confidence values

It should be noted that some competing HPEs, like AlphaPose [29], will always return an estimate of every possible keypoint even when they are invisible. But for those invisible keypoints, the confidence scores will be very low.

Image size and pose performance

Inference time of the HPE is highly dependent on the image size during pose processing, with bigger images taking longer to process than small ones. As the image obtained from the camera is large (1920x1080), pose estimation on the full, unedited image slows the process to a crawl. This problem must be mitigated for the algorithm to be able to run in real time.

A possible strategy is to use a bounding box detector to determine which parts of the image contains persons. Areas inside and near the boxes can be segmented into separate images and processed separately. This removes the need to process the entire image; however, it comes with its own set of problems. First, it makes the detection accuracy highly dependent on the bounding box detector. Compared to the pose detector, bounding boxes struggle more when the person is occluded, which lowers detection accuracy in many of the critical scenarios in this project. This is demonstrated in figure 4.11, where the system fails to detect a person partially hidden behind a robot but still visible, because the bounding box cannot see the person.

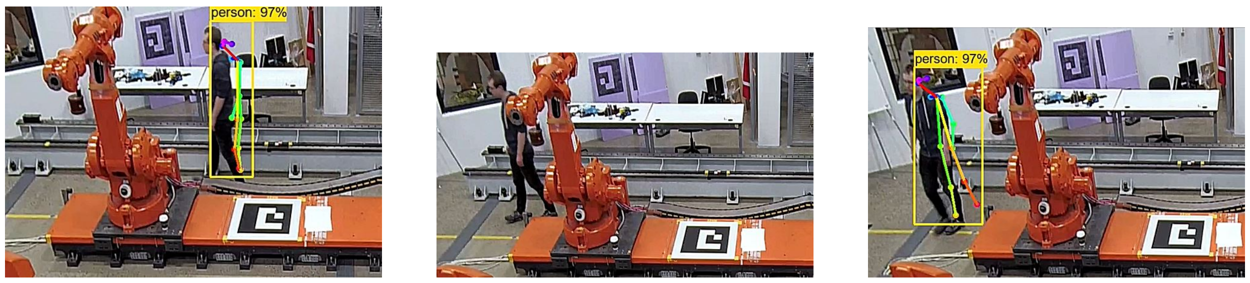

Figure 4.11: Failure to detect partially obscured person, when using bounding box detector for segmentation

Second, as the number of persons in the frame increases, performance slows down drastically. tf-pose-estimation does not support parallel processing of multiple images out of the box, so the pose estimation must be rerun for every candidate segment in the frame. Combined with the issue demonstrated above, this made bounding box segmentation unsuitable for the problem, and the method was quickly dropped once a better solution was found.

Another, easier method is to reduce the size of the full image before pose processing, which speeds up the program drastically compared to using the full image. This must be carefully weighted against another trade-off, however: If the image is downsized too much, detection accuracy will decrease. The low resolution makes persons harder to detect, as well as harder to separate when multi-person estimation is to be implemented.

To test the effect of image size on inference time and accuracy, an experiment was carried out. The basis of the experiment is a short video clip, including a total of 90 frames recorded at 20 FPS (4.5 seconds of footage in total). For the sake of future proofing, the video contains two persons. Both persons are always visible, although they cross very close to each other at one point, potentially leading to the estimator interpreting the result as a single person if the image resolution is too low. The video was recorded and stored using the full 1920x1080 resolution of the original camera. A selection of frames from this video are shown in figure 4.12.

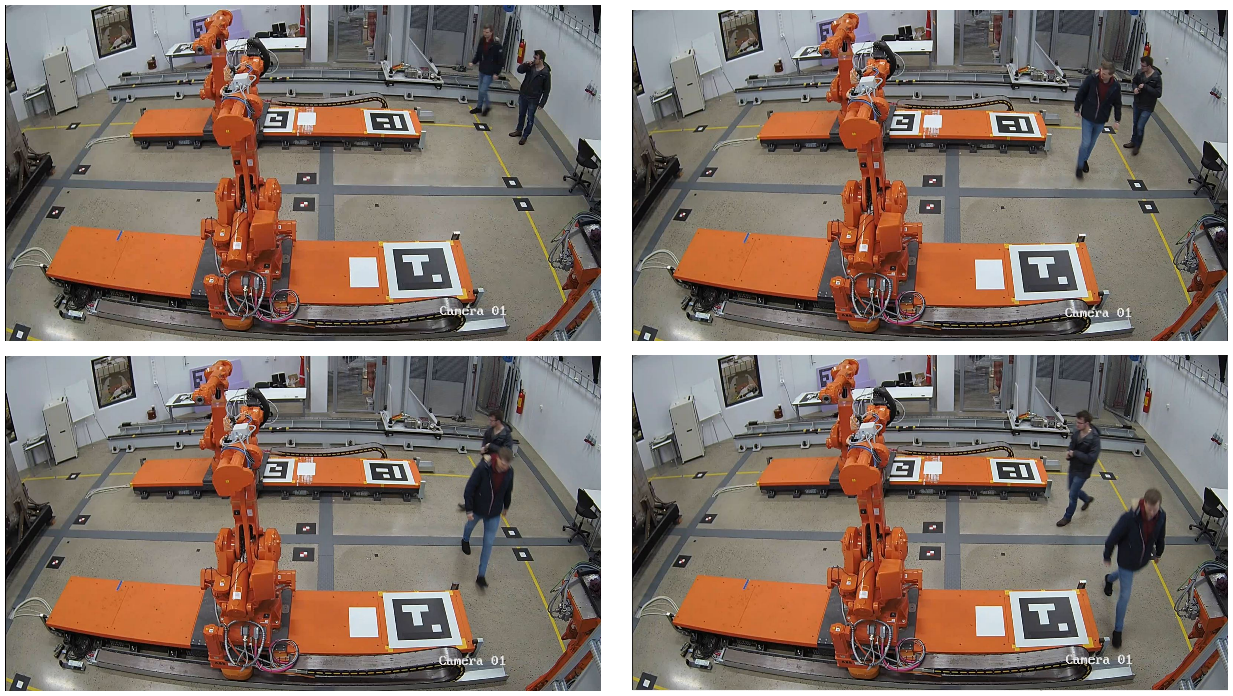

Figure 4.12: A few frames from the 4.5 second video clip used for pose performance analysis

At each iteration of the experiment, the image size was reduced, using a scaling factor that was varied on each step. Both axes were scaled equally so the aspect ratio of 16:9 was preserved. Pose inference was performed over all 90 frames using the given scaling, and the average processing time and number of detected persons were recorded. The detection was carefully filtered during the process to remove all traces of false detection. Accuracy was obtained by the average number of persons discovered over all frames, where 100% corresponds to always discovering both persons in every frame.

This experiment was carried out on the target hardware, as well as on the Xavier, to gain an understanding of typical pose inference time for multiple

Pose failure cases

OpenPose and its derivatives sometimes struggle to detect persons as expected. Some common issues include:

- Random background objects may be interpreted as humans
- When feet are occluded, the system may ”fill in the blanks” and predict erroneous ankle locations nearby
- Two overlapping persons may be detected as a single person
- Inconsistent detection when only legs are visible

These issues were gradually discovered over the course of the project, and tests were performed to document the issues. More on this in the result section.

4.5 Human body proportions

The post estimator identifies several keypoints on the human body. Where most of the points are used to estimate the persons position. Keypoints such as elbows and hands are excluded from the position estimation. This is due to the person can move the arms without changing position.

In order to use the keypoints properly it is needed to know at which height or z-plane the keypoint is at. To generalize the problem it was desired to have this as a ratio between keypoint height and total height.

To identify these ratios, a small test was executed. 6 individuals were measured and the measurements can be found in table 4.2, while the ratios can be found in table 4.3. It is important to note that these results are based on a small sample set with only male individuals. The numbers might not be representative for a larger population. Despite the low amount of samples, the standard deviation on the different ratios were sufficiently low.

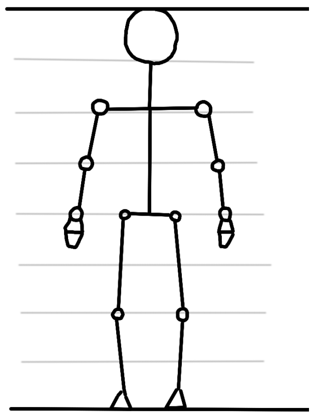

Figure 4.13: Human proportions

Name	Total Height	Eyes	Ears	Shoulders	Hips	Knees	Ankles	Unit
Person 1	185	173	170	150	97	59	20	[cm]
Person 2	178	168	162	145	96	56	19	[cm]
Person 3	190	178	175	153	104	60	22	[cm]
Person 4	190	180	169	156	100	60	20	[cm]
Person 5	186	175	172	150	96	56	18	[cm]
Person 6	191	180	177	158	102	61	21	[cm]
Average	186.7	175.7	170.8	152	99.2	58.7	20	[cm]
std	4.89	4.68	5.27	4.69	3.37	2.16	1.41	[cm]

Table 4.2: Measured keypoint heights

Name	Eyes	Ears	Shoulders	Hips	Knees	Ankles	Unit
Person 1	0.94	0.92	0.81	0.56	0.35	0.13	[-]
Person 2	0.94	0.91	0.81	0.57	0.35	0.13	[-]
Person 3	0.94	0.92	0.81	0.58	0.34	0.14	[-]
Person 4	0.95	0.89	0.82	0.56	0.36	0.13	[-]
Person 5	0.94	0.92	0.81	0.55	0.33	0.12	[-]
Person 6	0.94	0.93	0.83	0.57	0.34	0.13	[-]
Average	0.94	0.92	0.82	0.57	0.35	0.13	[-]
std	0.0041	0.0138	0.0084	0.0105	0.0105	0.0063	[-]

Table 4.3: Measured keypoints ratio, keypoint/height

4.6 Relative height estimation

Being able to estimate the height of a person is important to achieve best possible accuracy, a higher confidence in the height estimation will yield a higher confidence in the position estimation. The proposed method for height estimation utilizes the assumption that the measured person is completely visible and is standing straight up. Then the HPE will detect a full pose and return a list with image coordinates for each joint in the pose estimation. From the foot features, a good estimation of position in the world frame can be estimated. Since OpenPose detects the ankle joint which is above the ground, a small z-value must be added. Here a z-value of 20 cm is used. Empirical tests in section 4.5 showed that ankle heights usually tends to be at that value. Now the persons position and image coordinate for head features are known. Using the same method for position estimation as described in section 3.5, however with the modification of estimating the scaling factor s with either X or Y coordinates. Then calculating the Z-value with the following equation:

$$s = \frac{X_0 + \mathbf{B}_0}{\mathbf{A}_0} \tag{4.1}$$

$$Z_{\text{estimate}} = s\mathbf{A}_2 - \mathbf{B}_2 \tag{4.2}$$

It is important to note that this method does not provide the total height of the person. Only the height up to the eyes and ears. OpenPose does not give out the image coordinate to the top of the head and thus it is not possible to estimate the person height without using human proportions. The height up to the eyes and ears follows a known ratio (R_{eye}) and from this, the total height of the person can be estimated.

$$Z_{eye} = \mathrm{R}_{eye} \cdot Z_{person} \tag{4.3}$$

$$Z_{person} = \frac{Z_{eye}}{\mathrm{R}_{eye}} \tag{4.4}$$

While testing the height estimation, fluctuations in the height were witnessed. Even though they were not critical, it was decided to create a filter that finds an estimate with higher probability of being closer to the actual value. This was done by storing previous measurements in an list of fixed size. The newest value (Z_i)

is added to the right of the list (Z) and the oldest one removed, to keep the list at a fixed size. The variance of this list is then calculated and if it is sufficiently low (i.e. all the measurements are relatively consistent) then the average of the list is calculated and stored as variable $\hat{Z}_{person}$.

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|}\hline 1.815 & \multicolumn{5}{c|}{\overbrace{1.816 \;|\; 1.848 \;|\; 1.793 \;|\; \cdots \;|\; 1.803}^{k\text{-elements}}} & 1.770 & 1.771 & \cdots & 1.836 \\ \hline \end{array}$$

Figure 4.14: Fixed list of k elements sliding through measurements

$$\hat{Z}_{person} = \frac{1}{k}\sum_{i=0}^{k} Z_i \tag{4.5}$$

Humans can move their bodies into many types of postures, sitting, squatting, standing and kneeling. The proposed position estimation method works for all the previous methods, although it is needed to know the relative height of the person. What is meant with the relative height is the distance between the feet and head in different postures. A persons height does not change with different postures but their relative height changes.

4.7 Weighted sum position

If the full person is detected, there will be a total of 13 position estimates, using each of the keypoints which are likely to move smoothly as the person moves. To gain a better overall position estimate, the average of the positions can be used.

$$P(x, y) = \frac{1}{n}\sum_{i=1}^{n} P_n(x, y) \tag{4.6}$$

However, this will only work optimally when the noise in the system is evenly distributed. When the position estimate of each keypoint is graphed then a slight bias is seen, which increases with the height. Since it is difficult to show that the error grows further up, figure 4.15 shows how the position estimate for the keypoints. Note that the deviations have been artificially altered to emphasize the differences between real and expected position estimates.

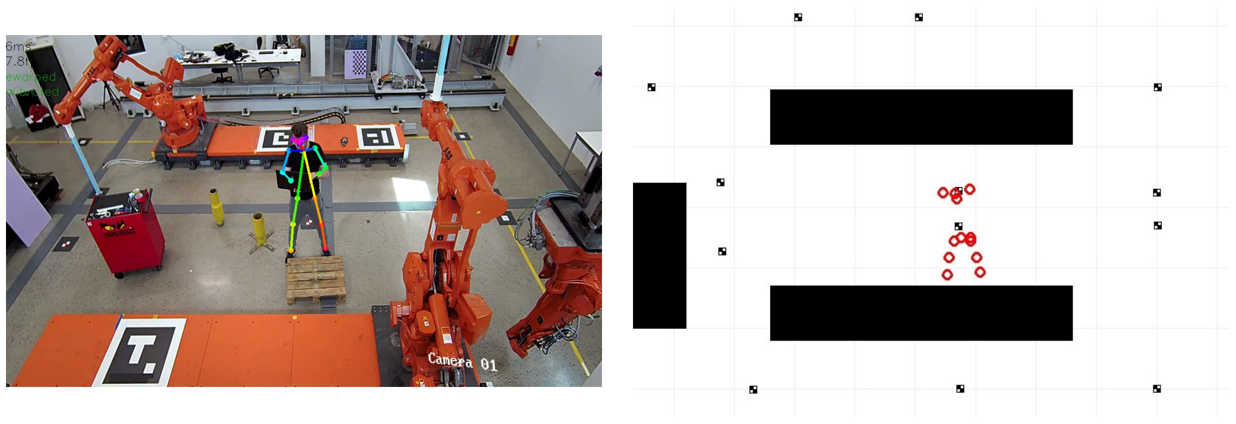

Figure 4.15: Deviations in position estimation for different keypoints

In order to estimate the keypoint biases, each keypoint and its position estimate was stored when a person stood on a landmark. To avoid that the biases represents only one orientation of the person, the position estimates were recorded while the person rotated 360° on the landmark. Furthermore, the same recordings were done on multiple landmarks to ensure spatial invariance. The bias was set to be the difference between estimated position and the known true position. Since there were numerous estimates from the different orientations and landmarks, the final bias value was set to be the average of all estimated biases for each keypoint.

$$\beta = \frac{1}{n}\sum_{i=1}^{n} P_i(x,y) - P_i^*(x,y), \tag{4.7}$$

Where:

- $P_i(x,y)$ is the known landmark position
- $P_i^*(x,y)$ is the estimated position
- n is the number of samples

Figure 4.16 shows the different uncorrected position estimates on landmarks: 21, 22, 23, 31, 32, 41. The different colors represents different keypoints. A legend in the figure is not provided due to the number of different keypoints and the goal of the figure is to illustrate drift in position estimation.

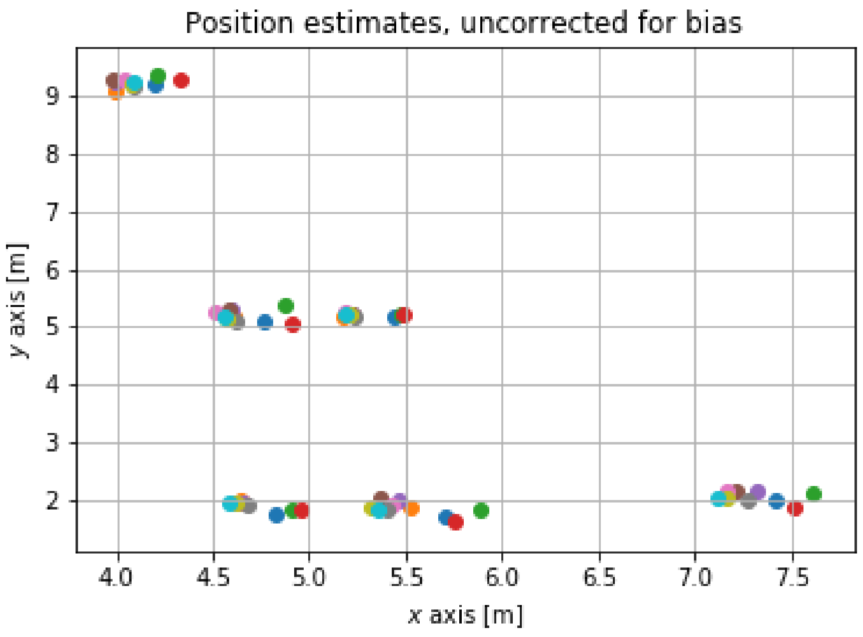

Figure 4.16: Variance in position estimates

From the empirical tests, it seems that keypoints closer to ground are more accurate. Therefore it is desired that these keypoints have a greater impact on the estimation rather than the more uncertain higher keypoints. The resulting method is a weighted mean function. Here the σ_i is a scalar ranging from 0 to 1. A high σ_i value means that the keypoint is more trusted and thus contributes more to the total sum. The β_i value is the bias for that keypoint.

$$P_{\text{WMS}}(x,y) = \frac{\sum_{i=1}^{n} \sigma_i (P_i^*(x,y) - \beta_i)}{\sum_{i=1}^{n} \sigma_i} \tag{4.8}$$

Figure 4.17 shows how the position estimates are after inclusion of biases. As shown, the spread in measurements are significantly lower.

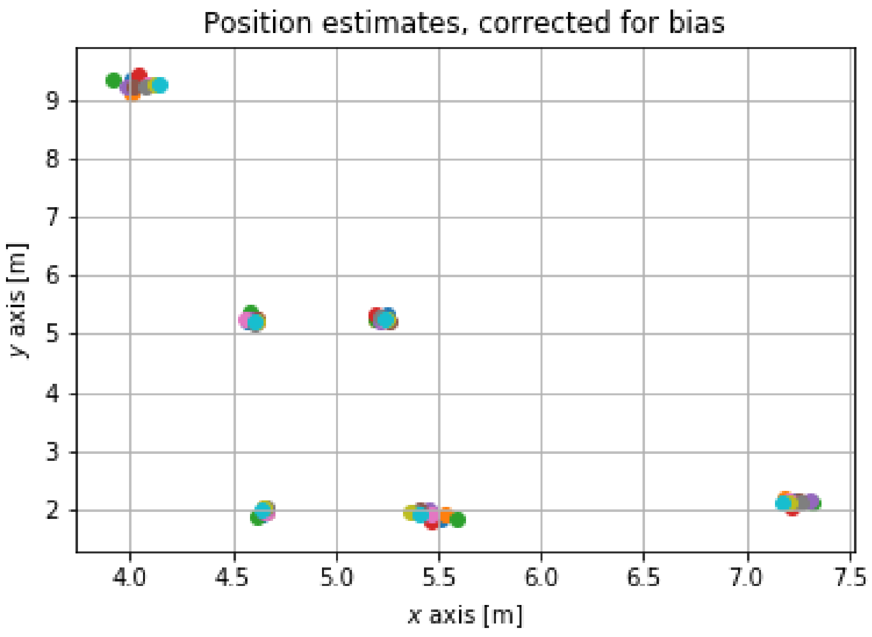

Figure 4.17: Position estimates with biases included

4.8 Sensitivity analysis of position estimation

The purpose of a sensitivity analysis is to gain knowledge of how a system reacts to uncertainties in the input. If a system is highly sensitive, then a small change in the input would change the output significantly. However if the system is robust, i.e. not sensitive, then the output would not change substantially. In regards of the position estimation, it is not known how sensitive the method is to changes in pixel coordinates or the height of the person. Therefore, sensitivity test will be carried out to measure this fact.

4.9 Edge Computing

In the recent years a new class of microprocessors called *AI accelerators* has emerged, designed as hardware acceleration for machine learning applications. Products such as the Nvidia Jetson series, Google Coral, Intel Movidius fall into this category. These products work similar to a GPU however with one large difference, GPUs work with high precision arithmetic while AI accelerators work on low-precision arithmetic (LPA) . Diving deeper into how these products differ and how high/low-precision arithmetic works, is beyond the scope of this thesis. A gross oversimplification is that machine learning networks does not always need the high precision arithmetic for inference and LPA also require less memory and computes faster.

The current setup computes everything on a centralized point. One computer that deals with all the computation and collection of videofeeds. This setup has the advantages by consisting of off-the-shelf hardware that could be easily bought in local computer hardware stores. However it has the negative sides of being harder to scale up to larger systems. Then there can be potential problems with receiving all the video feeds due to bandwidth limitations. Furthermore, running OpenPose on numerous videofeeds would require extreme computational power that would require multiple GPUs. How OpenPose works on multi GPU setup is not known. The largest disadvantage is that a centralized setup has no redundancy against hardware failure.

One single point of failure and the entire system collapses. On the other hand a decentralized system is implemented in the form of edge computing. Meaning that the position estimation is calculated on the edge, i.e. right next to the camera. Then the only information being sent a cross the communication network would be the position. Which is insignificant in size compared to a video feed. Now the hardware requirements for the main computer is reduced since it only needs to receive positions from the nodes, thus making it cheaper to have a another computer as redundancy. If the system is set up so there is sufficient overlap between the nodes, one can fail and the entire system will not collapse while the nodes awaits replacement.

Figure 4.18: NVIDIA Jetson AGX Xavier embedded computing board, image courtesy of Nvidia

Chapter 5

Results

5.1 Human Pose Estimation (HPE)

Image size and pose performance

As detailed in section 4.4 page 22, pose estimation was run over the 4.5 second video clip for multiple image scalings, measuring the average inference time and average number of detected persons. The latter was converted to a percentage where 100% corresponds to the detection of both persons in every frame. The results are shown in figure 5.1.

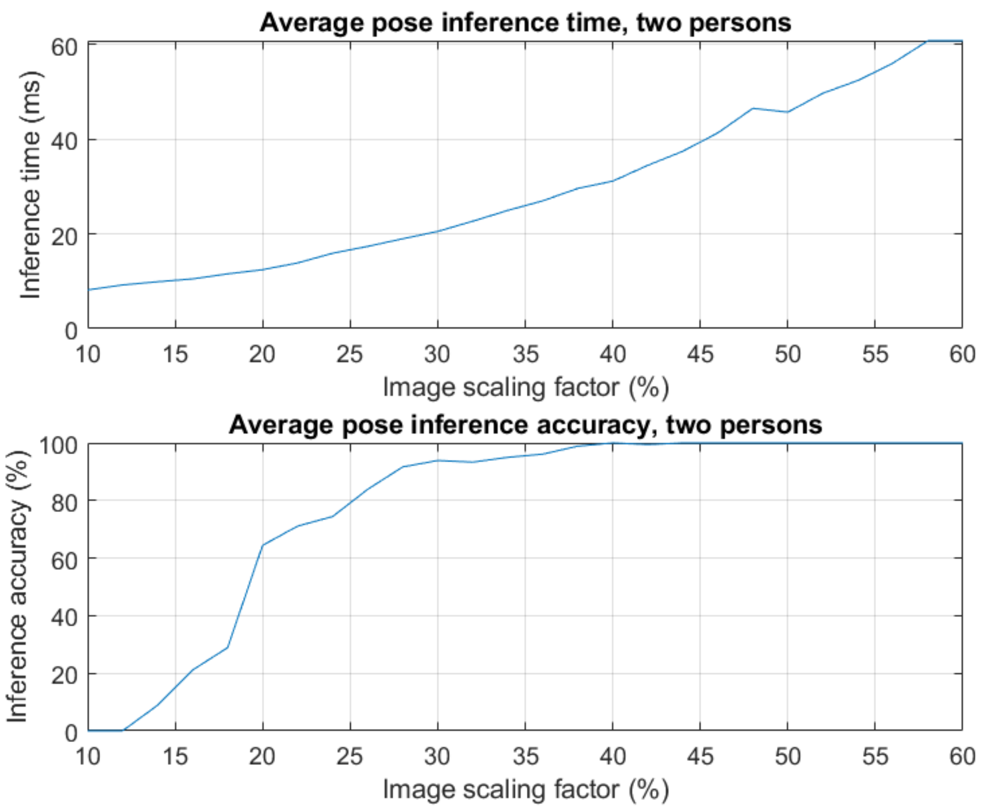

Figure 5.1: Result of investigating relation between image scaling and time/accuracy

As a result of this experiment, the image scaling was set to 40%, giving an inference image size of (768,432), This led to a good detection rate while still not using more processing time than required. At this scale, the program was able to find both persons in nearly all frames, while only taking just above 30ms on average to process each frame. As the pose detection is the most resource intensive component of the current implementation, the result should work well in real time. Keep in mind that this result may not match the real-time one.

To clarify the relation between image *area* and inference time, the time graph above can be expressed in terms of *area* scaling factor, which is achieved by squaring the image side scaling factor. The result is given in figure 5.2.

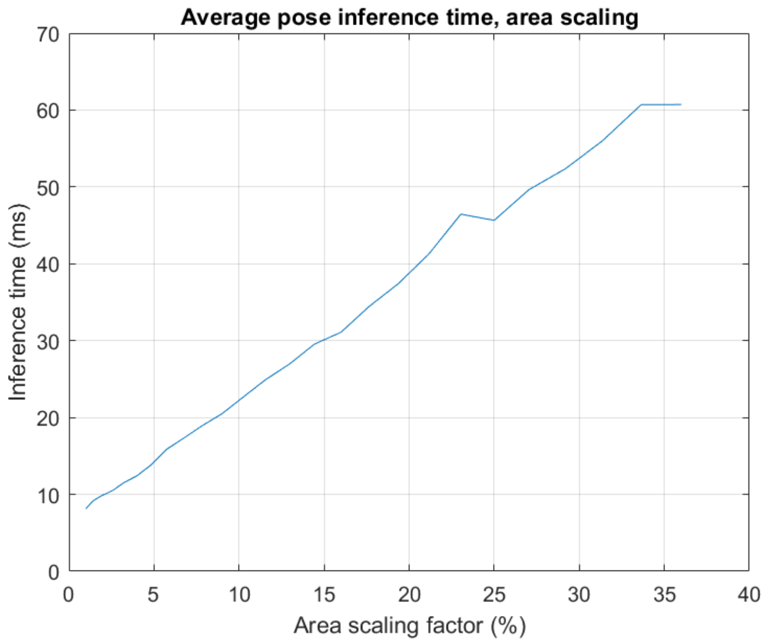

Figure 5.2: Result of investigating relation between image area scaling and time/accuracy

As can be seen from the graph, there is an approximately linear relationship between image area and average processing time, given the same underlying images. The values in the graph depend on the setup, software configuration and optimizations, but an underlying linear graph, given a constant setup, is expected to remain. Note that, according to documentation for the original OpenPose implementation [28], inference time under this architecture is almost independent on the number of persons in the image, compared to architectures based on R-CNN and AlphaPose.

The real program runs slower, because of the added processing complexity of running the entire program. To verify this, another test was performed. The real program was run for 30 seconds, with a single person always visible and walking around in the image. The pose processing time was recorded, as well the full execution time of a single loop in the position estimator. The test was performed with an image scaling of 35%, 40% and 100% respectively; where the two first values are close to the one chosen in the previous test. The result is given in table 5.1.

Image scaling	Average pose processing time [ms]	Average loop time [ms]	Average frame rate
100%	237	261	3.8
40%	44	51	19.8
35%	36	43	23.2

Table 5.1: Result of testing effect on image scaling during pose processing, in the real program

The pose processing time is slightly higher in the real program than in the experiment, for the scaling of 40%, and the additional processing done by the remainder of the program increases the time slightly further. However, the frame rate still remains at an average of just under 20 FPS, which is fast enough for real-time processing. The final image scaling is set to the value of 40%, leading to a pose processing image size of 768x432.

False human detection

Over the course of the project, the HPE would sometimes misclassify random background object as humans. The OpenPose paper [28] documents this as a common issue when the image contains statues or animals. In early stages of the project, chairs and ABB robots would sometimes be interpreted as humans, even though the detector would usually only find a few of their "keypoints". An example of such a detection is shown in figure 5.3.

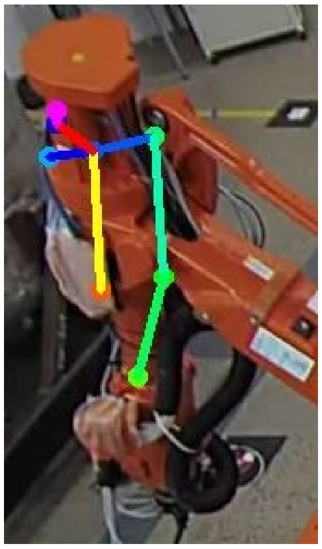

Figure 5.3: Example of false human detection

The amount and severity of false detections varied through different stages of the project. In some situations, they would barely show up, and only remain for a couple of detection cycles at most. At other times, the detections would linger and take on the shape of a human for several frames. These detections were often hard to reproduce in later stages, making them difficult to fully understand and document. For example, the ABB robots present in the room would occasionally show up as a human, as shown in figure 5.3. But this was rare, and in most other cases, they would not be "detected" at all. It is possible that, in addition to their relative placements, that lighting conditions also played a role.

The problem can be reduced by raising the human threshold score, keeping in mind that this score is not normalized, and will often take values higher than 1. In the tf-pose-estimation implementation itself, this value is set to 0.4. Through trial and error, the threshold was set to a value of 0.8, eliminating the vast majority of false detections while all humans still showed up as desired. A few fleeting ones can sometimes appear, although they only tend to persist for one or two frames and feature almost no keypoints, making them fairly easy to sort out. Higher values of the score would sometimes make it harder to detect persons that are partially occluded, so this must be done with care. For the remainder of single-human tests in the project, only the human with the highest score will be kept, filtering away the remaining ones.

For the most part, a brief detection with a low number of parts, as well as low human score (< 0.8) and low part confidences, can be considered untrustworthy, especially if it is far away from areas where occlusion is likely.

False ankles and keypoint confidences

If most of the upper body is visible while the legs are (partially) hidden, the pose estimator would sometimes make a guess of where the feet are. This may or may not be caused or worsened by nearby edges present at the locations where the feet realistically could have been. These fake feet may be too long, too short, or twisted into unrealistic positions. A couple of such cases are shown in figure 5.4. Because of this, care may have to be taken when evaluating the position of the ankles. Similar issues were less likely to happen with parts belonging to upper body, which can usually be trusted assuming that the human detection is not entirely false. In situations where these edge cases become prominent, the confidence values of the ankles can be checked, testing whether the ankles can be trusted. As part of the experiments, a test was be carried out where the confidences of false feet are compared to those of the rest of the keypoints.

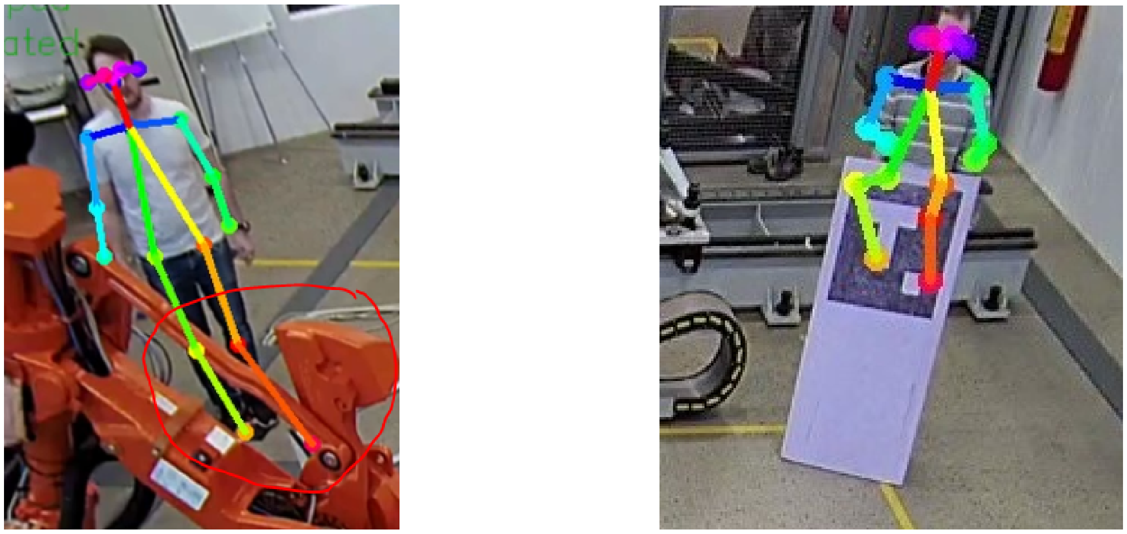

Figure 5.4: Examples of false foot detection

A somewhat contrived edge case was set up, which leads to the detection of a pair of very short ”ankles”. The resulting pose detection, along with its confidence map, are shown in figure 5.5, and the confidence values of each keypoint are listed in table 5.2. As can be seen from the image, the lower detected ankles do not really exist, leading to overly short legs. However, from the accompanied score map, the confidence values of the ankles are very low compared to nearly every other detected keypoint. The left and right ankles have confidence scores of respectively 22% and 31%, which are lower than for all other visible keypoints. By contrast, all body parts which are clearly visible, such as the left and right eyes and shoulders, all have confidences that exceed 60%, with the eyes and right ear exceeding 70%. The result also contains a fake wrist detection, but it only has a confidence value of 36%. In any case, wrists and arms are not used for position estimates, making this a non-issue. Out of all the detections that are not false positives, the lowest one is the left ear with 49%, which makes sense as the left ear faces away from the camera.

As can be seen from this test, fake ankles can be spotted and ignored in position estimates if their confidence values are very low compared to the rest of the values.

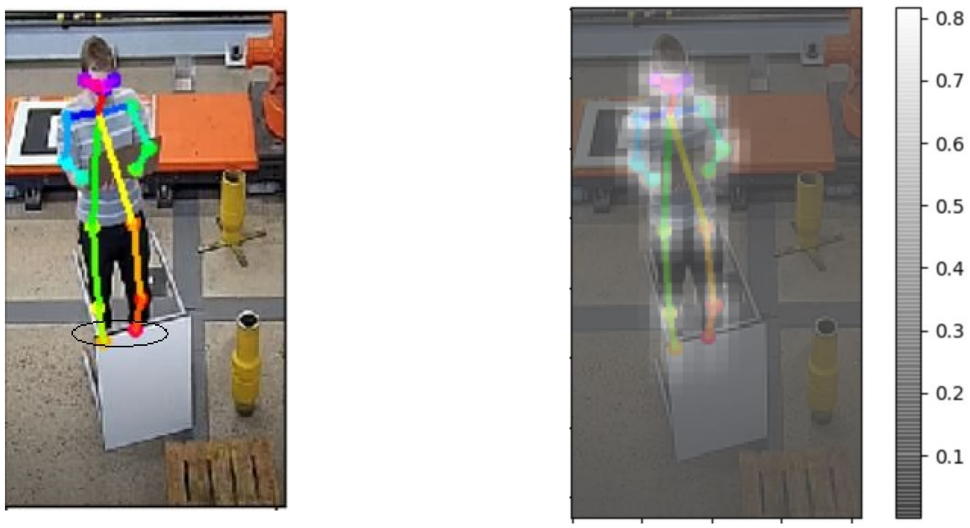

Figure 5.5: Resulting detection and confidence map in fake foot test, fake ankles circled

Keypoint type	Confidence
Nose	74%
Neck	75%
Right shoulder	77%
Right elbow	57%
Right wrist	67%
Left shoulder	64%
Left elbow	62%
Left wrist	36%
Right hip	67%
Right knee	55%
Right ankle	31%
Left hip	60%
Left knee	52%
Left ankle	22%
Right eye	77%
Left eye	72%
Right ear	82%
Left ear	48%

Table 5.2: Keypoint confidences from the fake ankle experiment

Incorrect merging of two persons

While the HPE performs well at separating different persons most of the time, as can be seen from the test in section 4.4, the aforementioned test also demonstrates how it can fail when one person is directly in front of another. This might be more likely to happen if the persons hold different poses, with one of them standing and another sitting in such a way that their heads align and confuse the system. This is illustrated in figure 5.6. As discussed in section 4.4, this issue also becomes more likely if the image size is reduced too much, for the sake of lowering inference time. So the problem can be mitigated by making sure the image used during pose inference has sufficiently high resolution to separate the persons. The pose scenario illustrated in the figure can be considered fairly rare and not applicable most of the time. The original clip where this scenario was observed, was specifically set up to be difficult to solve, and the persons were correctly seen as a single person as soon as they separated again.

The remainder of the results will center on single person estimation, so this scenario is mostly considered future proofing. If multi person estimation is implemented, edge cases could be solved using a conservative approach. If two persons is reduced to one near an obstacle, and the likelihood of the person being too close to the obstacle is high according to prior position estimates, it may be time to shut off the nearby machine.

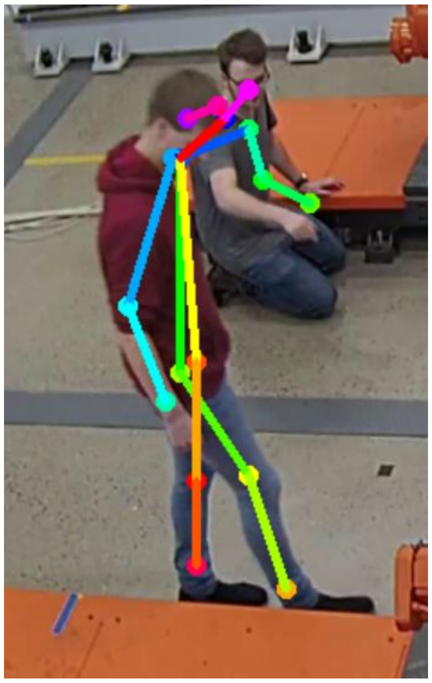

Figure 5.6: Failure case: One person in front of another, different poses

Failure to detect person

Perhaps the most serious issue of the HPE is inconsistent detections when only the legs are visible. As noticed during the experiments, the pose estimation usually runs well even when the feet and lower parts of the body are hidden, but detection becomes less reliable when all of the upper body is hidden. This issue is illustrated in figure 5.7.

Additionally, the HPE can struggle to detect persons lying on the ground, especially when lying parallel to the camera plane in a way that causes the feet to obscure each other. While this is not a major concern in this report, as position estimation breaks down regardless in this case, it is mentioned for completeness.

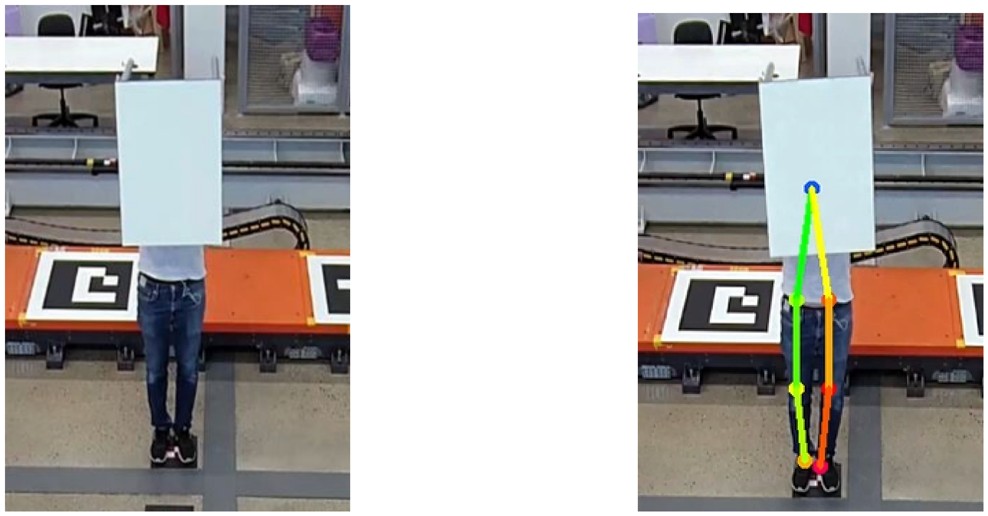
Figure 5.7: Inconsistency in detection when torso is occluded

5.2 Position sensitivity

The tests were carried out around the point with image coordinates (1000,500), which is approximately at the center of the image, in the region where most of the image processing will take place.

Deviation in uv-coord	Deviation in xy-coord [mm]	Absolute deviation [mm]
5, 0	4.12, -40.27	40.4
0, 5	-57.2, 9.92	58.1
-5, 0	-4.12, 40.30	40.5
0, -5	57.4, 9.96	58.3

Table 5.3: Result of pixel coordinate sensitivity test

For the height sensitivity test, calculations were carried out around the height of 180cm, with the uncertainty in z-value added to this value. The planar position with image coordinates of (1000,500) was once again used.

Deviation in z-value [mm]	Deviation in xy-coord[mm]	Absolute deviation[mm]
-150	149.0, 44.0	216.0
-100	99.5, 29.7	144.0
-50	49.8, 14.8	72.1
50	-49.8, -14.8	72.1
100	-99.5, -29.7	144.0
150	-149.0, -44.0	216.0

Table 5.4: Result of height sensitivity test

As can be seen from table 5.4, for a given planar position, the estimation error is proportional to the difference in height.

5.3 Position estimation

Static position, no occlusion

The first test is while standing still on the landmarks with no occlusion. This test will provide a baseline for future reference in regards of accuracy. The estimated position was recorded for each visible landmark and calculated the absolute deviation, results can be seen in table 5.3. A composite image has been created to visualize the results. As shown in figure 5.9, the system is able to estimate the position with great accuracy (< 10cm), even when the person is standing with their back towards to camera.

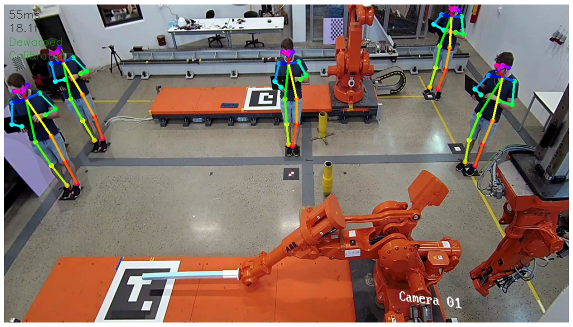

Figure 5.8: Position estimate, no occlusion

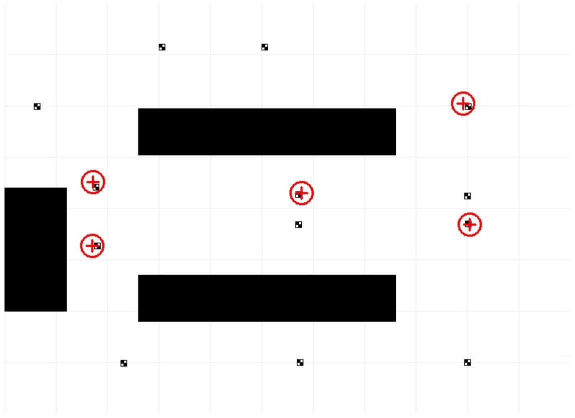

Figure 5.9: Position estimate

ID	x	y	$\lvert error\rvert$	unit
21	4.67	1.95	0.057	$[m]$
22	4.68	5.23	0.070	$[m]$
23	4.28	9.29	0.090	$[m]$
31	5.26	1.97	0.032	$[m]$
32	5.52	5.22	0.072	$[m]$
33	5.52	9.28	0.121	$[m]$
41	7.03	2.08	0.098	$[m]$
44	7.09	10.35	0.104	$[m]$

Table 5.5: Estimated positions and absolute error

In table 5.3, all estimated positions and the euclidean error are listed. Note that there are more entries than shown in the composite. The reduced number in figures was done to not overcrowd the image.

Static position with occlusion

In this test a simple occlusion was made by placing a tool trolley in front of the person. The trolley was chosen since it is easy to maneuver around and is a common object in a industrial setting.

From the figures 5.10 and 5.11, it can be seen that the system is still able to detect and estimate the position of the person when only upper torso is shown. The average error is around 30 cm, with a maximum error of 40 cm and minimum of 13 cm.

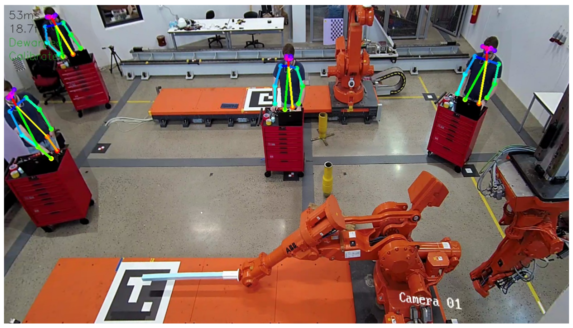
Figure 5.10: Pose estimate occlusion

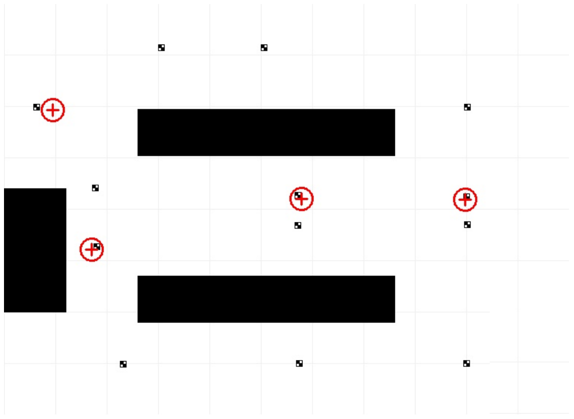
Figure 5.11: Position estimate

ID	x	y	$\|error\|$	unit
21	4.80	5.24	0.121	$[m]$
22	4.76	1.88	0.076	$[m]$
23	4.21	9.30	0.122	$[m]$
31	5.18	2.04	0.081	$[m]$
32	5.19	5.22	0.100	$[m]$
33	5.29	9.12	0.163	$[m]$
41	6.73	2.15	0.305	$[m]$
44	6.92	10.05	0.337	$[m]$

Table 5.6: Estimated positions and absolute error

Testing with industrial coveralls and helmet

Since the system is intended to work in an industrial setting, it is upmost important that it is able to detect the pose when the subject is wearing coveralls and helmet. Since most of the training data for OpenPose was not with uniform colored clothes and helmet it was feared that OpenPose would struggle with detection. Nonetheless, OpenPose detected the human without significant trouble and achieved similar results as without coveralls and helmet in fully visible and occluded scenarios. Even when the person is occluded and stands with the back against camera (figure 5.14, center), the accuracy is still quite good.

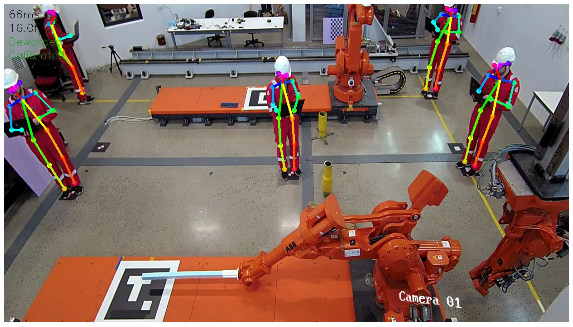
Figure 5.12: Coveralls and helmet test, no occlusion

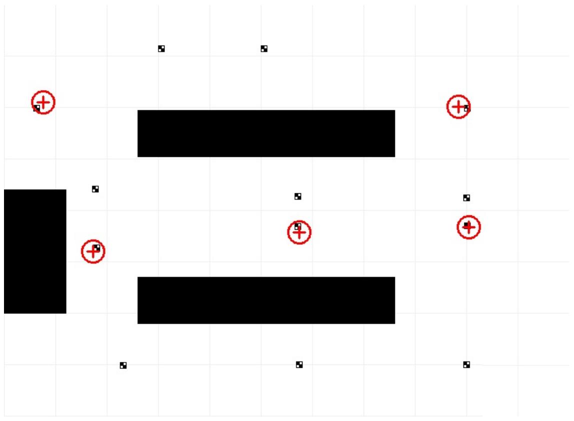
Figure 5.13: Position estimate

ID	x	y	$\lvert error \rvert$	unit
21	4.66	1.95	0.064	$[m]$
22	4.56	5.24	0.142	$[m]$
23	4.24	9.32	0.094	$[m]$
31	5.26	2.02	0.022	$[m]$
32	5.40	5.32	0.124	$[m]$
33	5.43	9.27	0.044	$[m]$
41	7.01	2.15	0.160	$[m]$
44	7.09	10.25	0.164	$[m]$

Table 5.7: Estimated positions, no occlusion

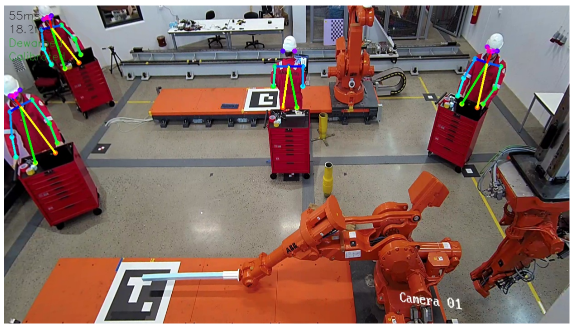

Figure 5.14: Coveralls and helmet test, occlusion

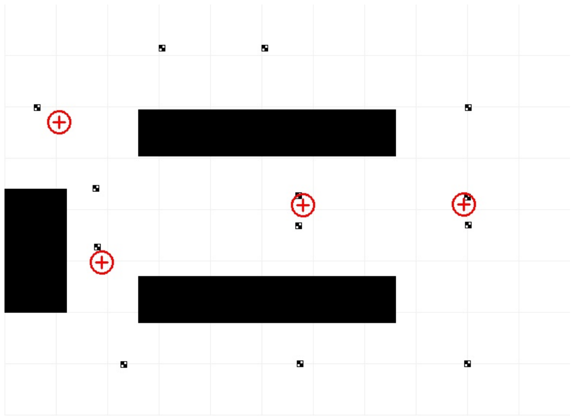

Figure 5.15: Position estimate

ID	x	y	$\lvert error \rvert$	unit
21	4.98	1.75	0.36	$[m]$
22	4.84	5.28	0.15	$[m]$
23	4.18	9.10	0.14	$[m]$
31	5.66	1.86	0.43	$[m]$
32	5.62	5.24	0.34	$[m]$
33	5.50	9.07	0.19	$[m]$
41	7.37	2.09	0.32	$[m]$
44	7.17	10.40	0.17	$[m]$

Verification of using landmarks

In this thesis the landmarks are used to solve the PnP problem and for ground truth in position estimation. This could be an issue due to the same points are now used for both creating the extrinsic matrix and estimating the position. One metaphor for this, a student who creates his own exam, will know the answers. To verify that the previous tests are valid, tests on other points that were not used in the PnP problem is needed.

In this test, 10 additional points have been created and measured up. The new landmarks were placed along a straight line between other landmarks with a fixed distance. Then the position of the landmarks could be calculated using trigonometry. The point P5 ended up being severly occluded and is therefore not included in the estimated positions. Calculations of the positions are trivial and therefore not shown.

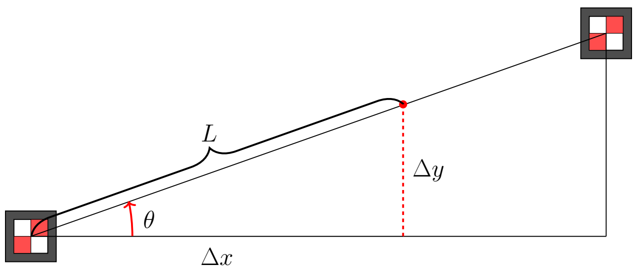

Figure 5.16: Illustration of how new landmarks were calculated

The results from the position estimation on these new points, is that they show similar accuracy results when compared to the landmarks. Therefore it is believed that the previous results are representable. Quantified results can be seen in table 5.8. Note that in one of the test results in image 5.18, the red marker is on top of the black part which represents the robot trolley axis. This is not an error, the trolley was moved closer to the wall and the map was not updated correctly. In the same figure two lines are drawn to aid visualization.

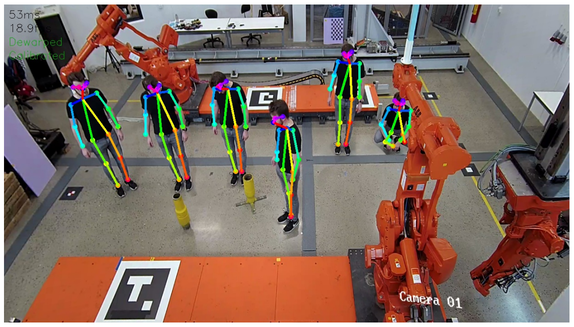

Figure 5.17: Landmark verification

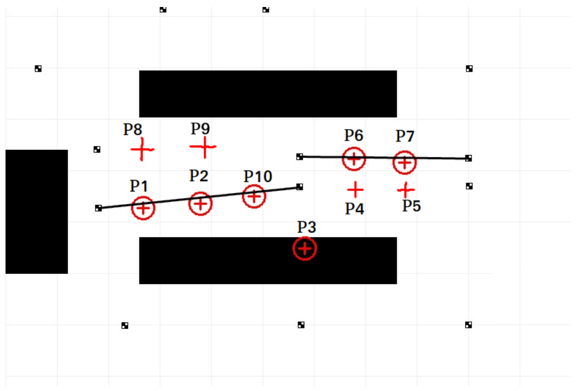

Figure 5.18: Position estimates, landmark verification

ID	x	y	$\|error\|$	unit
P1	4.38	8.21	0.12	$[m]$
P2	4.49	7.22	0.12	$[m]$
P3	3.69	5.30	0.23	$[m]$
P4	4.69	4.30	0.10	$[m]$
P6	5.28	4.29	0.07	$[m]$
P7	5.28	3.29	0.12	$[m]$
P8	5.38	8.23	0.06	$[m]$
P9	5.31	7.24	0.09	$[m]$
P10	4.51	6.19	0.09	$[m]$

Table 5.8: Estimated position, verification test for landmarks

Dynamic testing

To contrast all the previous tests, this test is taken while the person was walking. The test was executed with a person walking in straight lines between landmarks. The result can be seen in figure 5.19. Since there is no external position measurement systems in the lab, ground truth cannot be obtained. Nevertheless, from a purely visual standpoint, it shows that the system is able to estimate the position quite well, with little noise.

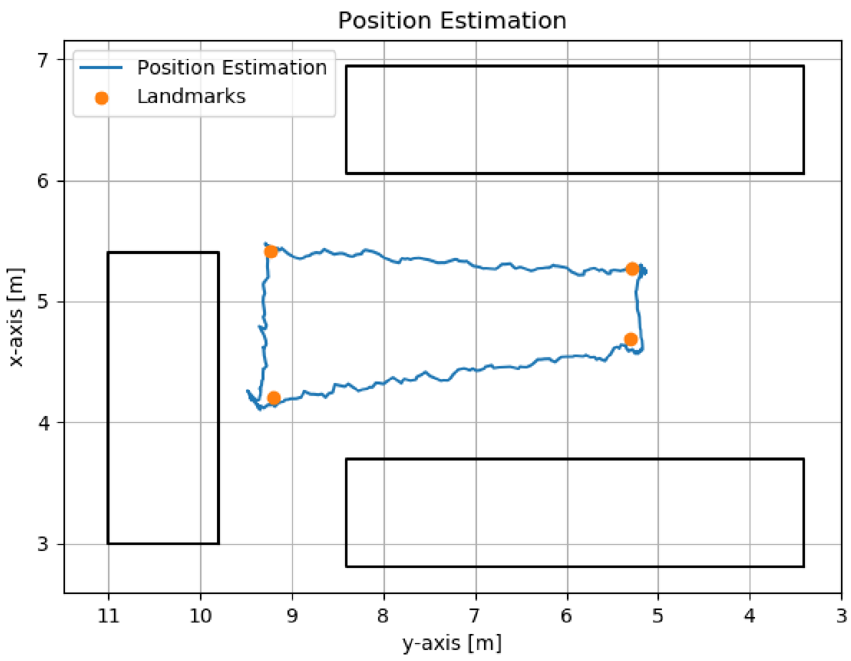

Figure 5.19: Results from dynamic test

To gain better information on how the system performs, a test where the person walks in only a straight line and then view the position estimates for each axis separately can provide some information. The test person walked along a line that was drawn on the floor at x value of 5.0 m. This provided the group a possibility to check error in the x axis

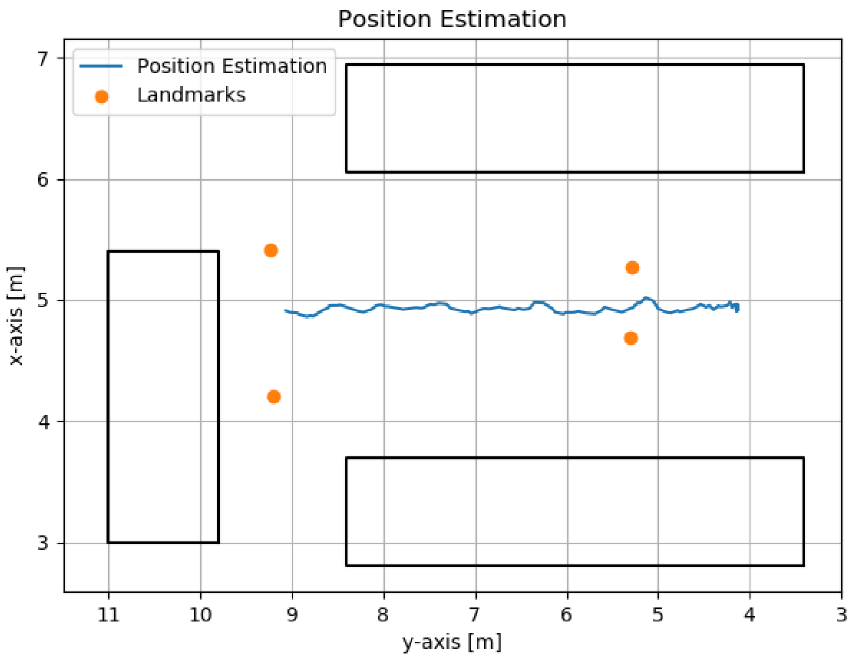

Figure 5.20: Results from dynamic test

Figure 5.21 shows both the position estimate error and the velocity for the test. Note that there is not given units for the velocity. This is due to how the velocity is calculated. It was calculated by taking difference of y values between measurements. The FPS varies slightly during operation, which in turn makes the velocity calculation invalid. Therefore the velocity plot is only meant to give an indication of velocity rather than actual values. The error in x axis is calculated by subtracting 5.0 m from the estimated value.

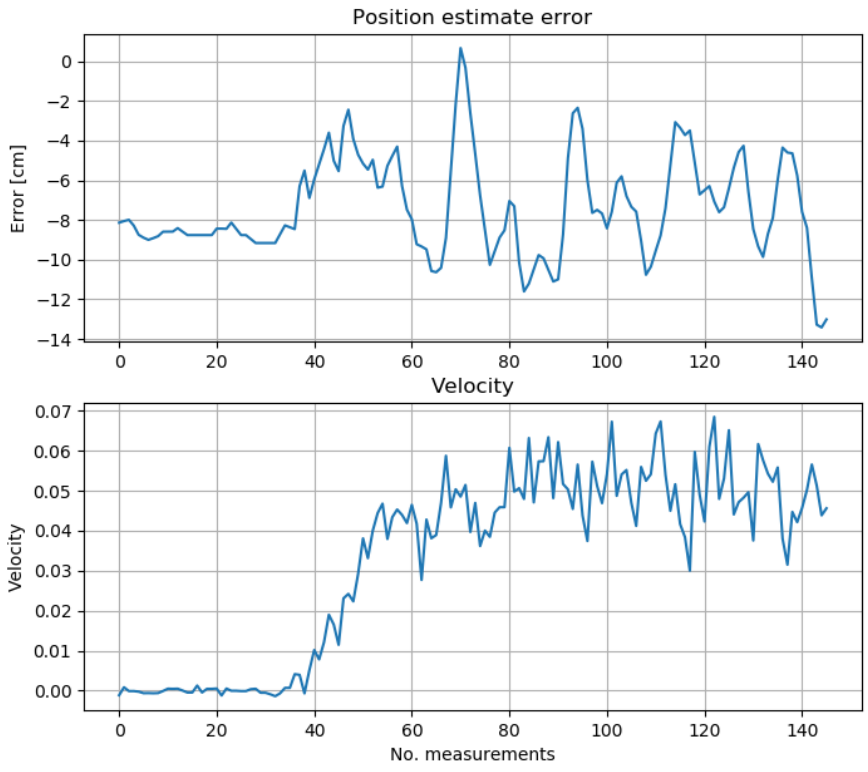

Figure 5.21: Error and velocity plot

From the previous test, an oscillating pattern can be seen, this could be from walking where the feet moving together and apart which could influence the position estimation. To test whether the stride length has an impact on the oscillation on the position estimation, additional tests with short and long stride length was completed. The person moved in the same straight line as the previous test. The position estimates can be seen in figures 5.22 and 5.23, estimation error for x axis and velocity can be seen in figures 5.24 and 5.25.

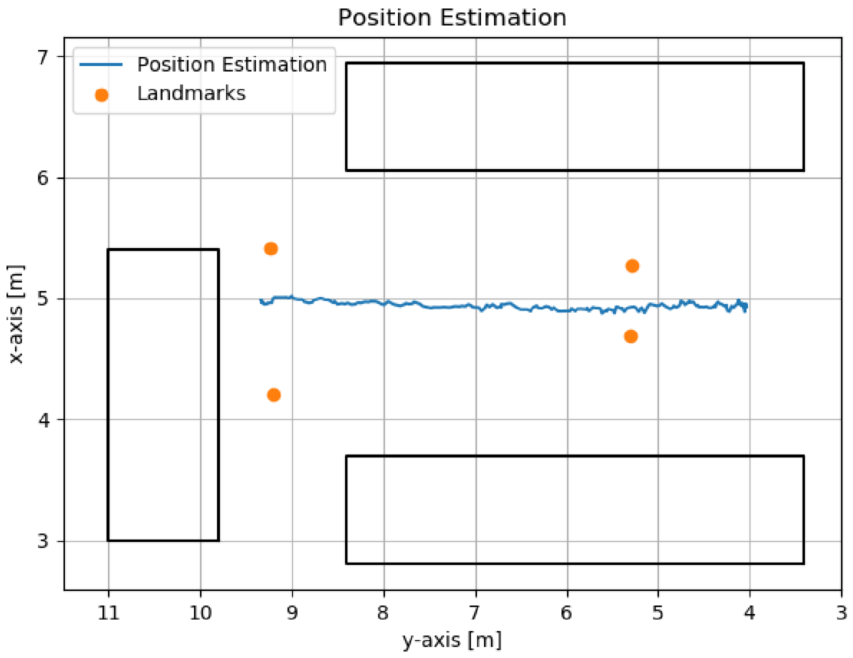

Figure 5.22: Position estimation with short strides

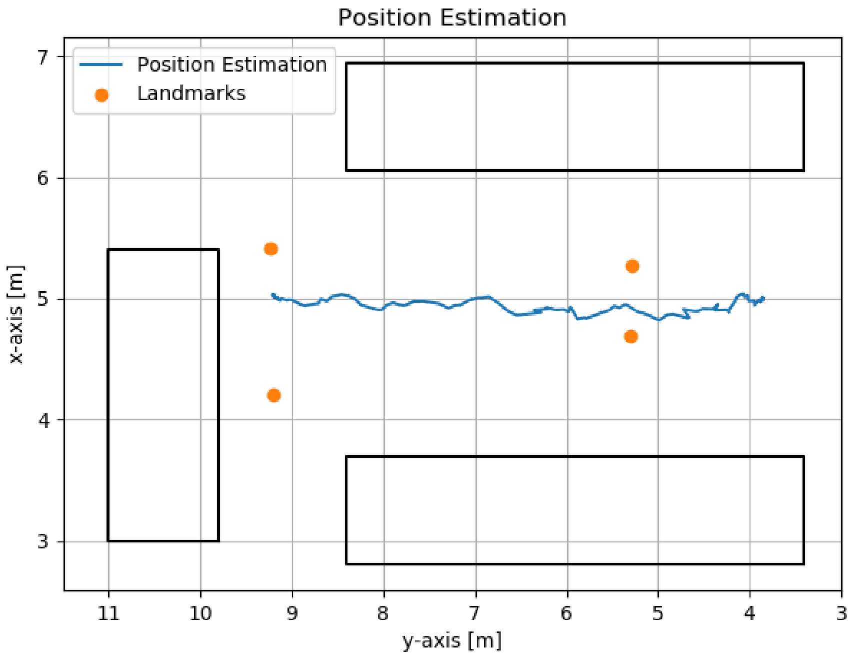

Figure 5.23: Position estimation with long strides

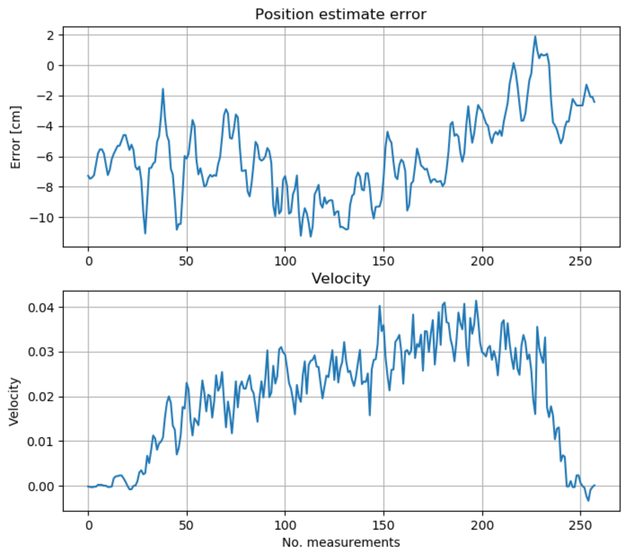

Figure 5.24: Error and velocity plot for short stride

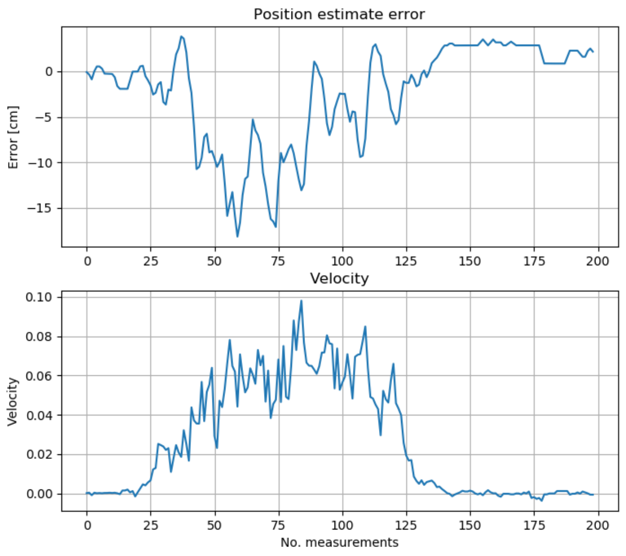

Figure 5.25: Error and velocity plot for long stride

From the tests it can be seen that the maximum error does increase with stride length. The most notable result is that the large oscillations in the short stride length is reduced. The end result is that the position estimation suffers slightly when the person is moving, although even with long strides the maximum deviation is around 20 cm.

5.4 Edge Computing - Nvidia AGX Xavier timing test

The inference time on OpenPose is heavily linked to the resolution, as shown in section 4.4 the inference time is linear in respect to the image area. In the same section the downsampling scale was also found. As shown there the image can be scaled down 40% without loss in accuracy. The original image is 1920x1080 before downsampling, after downsampling the image resolution will be 768x432. Creators of tf-pose-estimation have tested the software on a Nvidia Jetson TX2, where it was able to run at 10 FPS on a image resolution of 368x368. Our downsampled image will be ≈ 2.44 times larger in area than the image resolution which was tested on the Jetson TX2. Therefore it is believed that the Jetson TX2 is *not* powerful enough to run on the required image resolution and inference time. However Nvidia has released another AI accelerator which is called the Jetson AGX Xavier, which is claimed to be 20x more powerful than the Jetson TX2. If this is true and tf-pose-estimation runs with same computational efficiency on both sets of hardware, then it could be able to run the HPE with the required FPS and resolution. However this is all speculative and needs to verification by testing,

The pose inference test from section 5.1 on page 30, was rerun on the Xavier hardware. Detection accuracy was equal to the results on the workstation hardware, and the resulting average inference time at each scaling are given in figure 5.26.

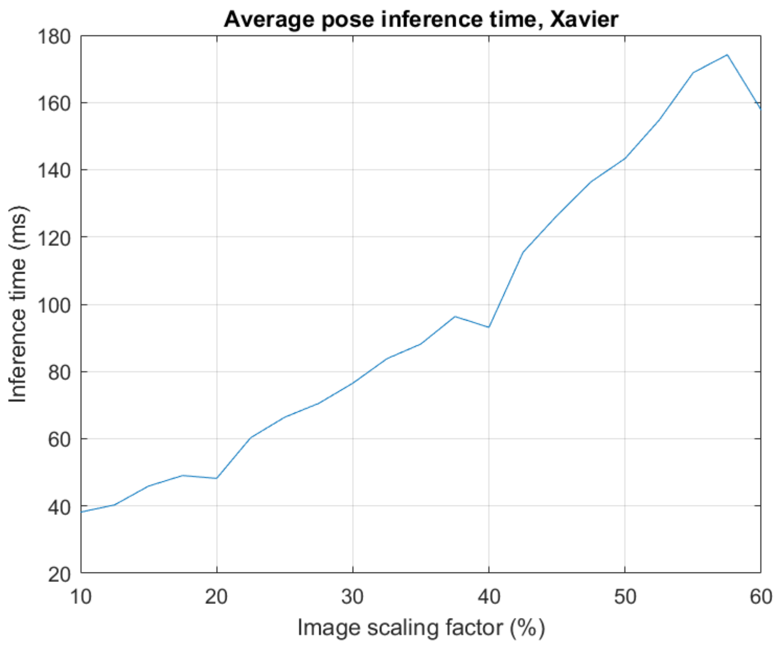

Figure 5.26: Frequency test performed on the Xavier

As can be seen from the graph, the Xavier performs slower than the workstation setup, taking about three times longer for the same scaling values. For a scaling of 40%, as is used in the original program, the Xavier barely remains below 100ms (just above 10 Hz). Since actual program performance might vary, and pose tracking functionality would be implemented on top of this in the future, the real performance frequency could easily drop below 10Hz for equivalent image sizes.

The original Jetson TX2 test performed on tf-pose-estimation by its creators [37] was run for a resolution of 368x368, and it was able to run the HPE at a frame rate of 10 FPS. It is useful to know the relative performance of an equivalent image on the Xavier, compared to the TX2 test given above. As inference time depends primarily on area, this involves testing an image of equal area to that used in the 368x368 test, calculating the resulting FPS, and contrasting with the above result. To produce this area, both sides of the image must be scaled, according to the factor s given by the equation

$$368 \cdot 368 = (s \cdot 1920) \cdot (s \cdot 1080), \tag{5.1}$$

which gives a resulting scaling factor of

$$s = \sqrt{\frac{368 \cdot 368}{1920 \cdot 1080}} \approx 0.256 = 25.6\% \tag{5.2}$$

Obtaining the resulting inference time from figure 5.26 the result would run at $\approx$ 66ms ($\approx$ 15 FPS). While this suggest faster performance on the Xavier compared to the TX2, it is still only a speedup of 50%, which is far lower than 20x. There are likely other, unknown bottlenecks at play.

Chapter 6

Personnel tracking using human pose estimation in multi-camera setup

6.1 Method and purpose

This chapter is a collaboration between two master theses that are both writing for NOV. The first group consists of Sondre Ripegutu, this project concerns with multi-camera person tracking. Second group consist of Andreas J. Akselsen and Harald I. Moldsvor and their project concerns with position estimation of occluded person. Further on in this section we will use "the multi-cam group" and "the occlusion group" to identify the different projects.

The purpose of this collaboration is to investigate the possibility of merging the two theses, where the symbiosis can be beneficial for both groups. The idea is to use the tracking implemented by the multi-cam group, and combine it with the detection framework constructed by the occlusion group. Since the reader might not read both of the theses, a short introduction for each project is given.

The multi-cam group code uses detections from multiple cameras to track personnel. The tracking algorithm retrieves detections from each camera and clusters it with a modified version of hierarchical clustering. New clusters are matched with previous clusters, saved as nodes, where a positive match will update the node, and no match will create a new node. The nodes locks on a unique ID to the matched cluster, where Kalman filter is the underlying algorithm.

The occlusion group code first estimates the human pose which is done by a machine learning application. This returns a "stick-figure" of the detected pose and gives out the image coordinate for each joint. Through projective geometry calculations using the image coordinates, intrinsic, and extrinsic matrices, the position can be estimated. In order for the code to work it needs an estimate of the persons height, which is automatically detected when the person is fully visible.

6.2 Setup and results

The multi-cam group code runs a bounding box detector on the camera image, and assumes that the lower middle part of the box corresponds to the position of the feet which is touching the ground. This makes the code run easily with multiple persons, as there is no need to keep track of individual heights. However, it also makes the code less robust in the presence of occlusions, making the positions unreliable when the feet are not visible.

For this chapter, the code was edited to use the pose based estimator instead of the bounding box, to make it more robust in the presence of occlusion. As with the occlusion code, this test is restricted to a single person. Extensions for multiple persons should work with this code as well.

To test the resulting code fusion, the position was tracked as a person walked along a predetermined path, consisting of straight lines where each corner has a known position. Various obstructions were added close to the path, ensuring that the person has multiple positions where feet and other lower limbs are obscured from view, sometimes in both cameras at the same time. The setup and the traced path are illustrated in figure 6.1.

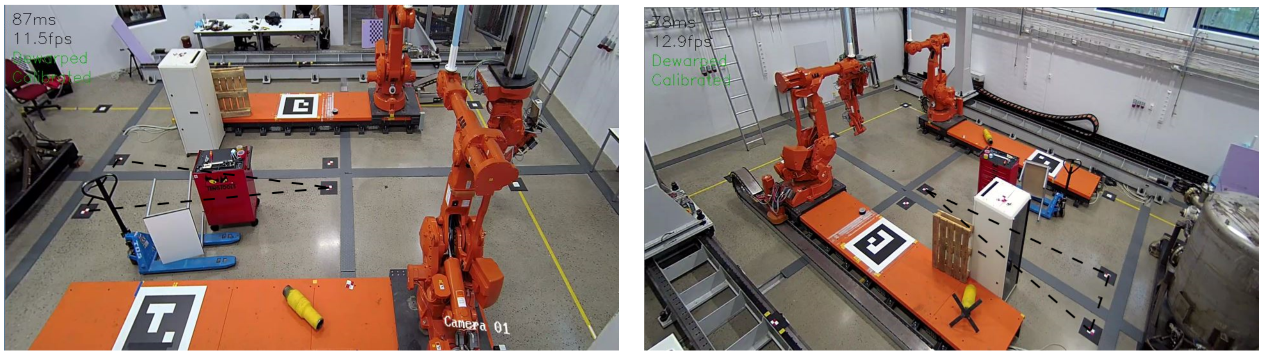

Figure 6.1: Room setup and traced path during multicam test

Illustrated in the figure 6.2, the merged system outperforms the original multicam system (figure 6.3) when occlusions are present. In figure 6.3, showing the bounding box detectors results, it is clearly shown where the position estimation went wrong for both cameras, as witnessed the dotted lines abruptly changes direction when the occlusion appears, which in return throws the merged estimation off.

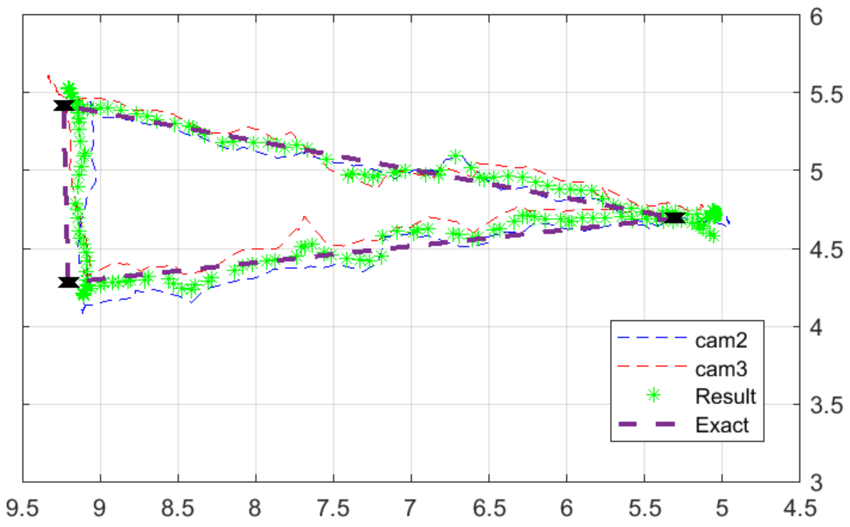

Figure 6.2: Result after merging of systems with occlusions

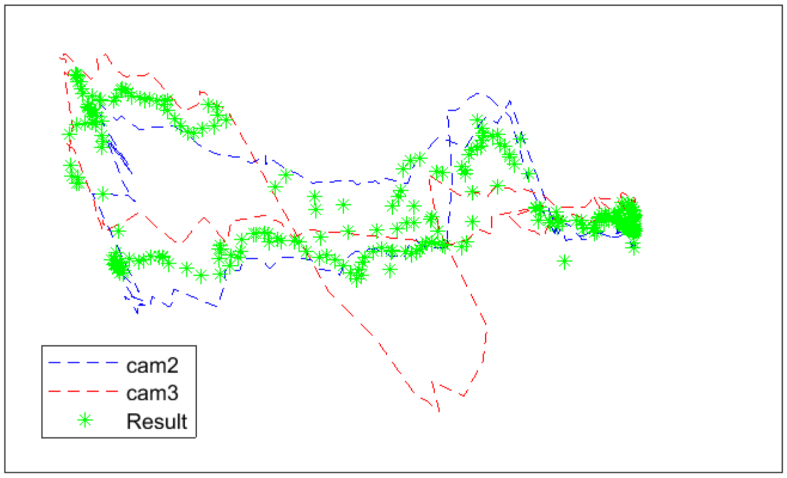

Figure 6.3: Result from original multicamera setup with occlusions

Ticks and grid is omitted in figure 6.3 because the multicam and occlusion systems work on different coordinate systems and to avoid confusion was removed.

In the end, the pose detector improves accuracy in the presence of occlusion, giving a position estimate even when the person is partially hidden, and the benefit of using a pose based detector is clear. This setup used two cameras, however adding more cameras can increase accuracy and robustness.

Chapter 7

Future work

Extensions to multiple persons

If the code is to be extended to multiple persons, one main challenge surfaces. In the current, single-person setup, the code makes an estimation of the person height, and stores it for further estimation in following frames. If the code is to be extended to multiple persons, this process must be repeated for every visible person. But this requires the ability to *track* the detected persons across frames, assigning an ID to each person, so each height can be assigned to the correct person in future estimates. Pose tracking uses the detected poses, and perhaps some form of visual data, to perform this ID assignment.

A simple scheme could be made by creating a cost/similarity function comparing the image locations of the keypoints in two given poses, such as a weighted average of differences between image locations. Matching could be carried out by combining the pairs in a way that minimizes this function for each pair, likely combined with some level of thresholding to prevent matching of widely different poses. As this does not rely on visual features, matching time would likely not be a major issue unless there are a massive number of persons in the frame, but the robustness of the matching would be in question.

A few research projects for pose tracking exist, one of which is the PoseFlow framework. PoseFlow ships with the AlphaPose HPE, but can be paired with any pose estimator which outputs data compatible with the common COCO or MPII keypoint formats. The tracking technique is described in a paper and implemented in Python as a separate GitHub repository. The tracker takes in a set of frames, as well as the corresponding poses from each frame, and forms pose IDs which persist across frames. In addition to pose data, the tracker makes use of visual data from each frame, to generate more robust poses than what is achieved by pose data alone. The overall pipeline and procedure of PoseFlow is illustrated in figure 7.1.

However, the currently implemented codebase of PoseFlow is primarily geared towards high accuracy, and as such, it is not optimized. It relies on feature matching functionality which, although in principle can be implemented on a GPU, is currently not. And the implementation also performs feature matching across the entire image, even though the paper itself suggests only running it in bounding boxes near the detected poses [41, p. 5]. Because of this, when performing tracking over a test video as part of this thesis, it runs at a very slow speed (over 700ms for a pair of frames). A tradeoff between speed and accuracy can be performed by tweaking the nFeatures parameter in the matching.py file, adjusting the complexity of feature matching. From the test below, it may be possible to lower the inference time to acceptable levels this way. However, a GPU implementation of the feature matching (or verifying whether it can be eliminated, or use velocity matching instead of visual matching) would likely be a better long-term solution.

For illustration purposes, PoseFlow tracking was performed on a prerecorded video, to show what can be achieved given the right implementation and/or tweaks to the code. Even with a highly reduced complexity of feature matching (reduced from 10000 to 300), while still running matching over the entire image, the code was able to track the persons well within this video. Some frames from the result, spread across in time, are

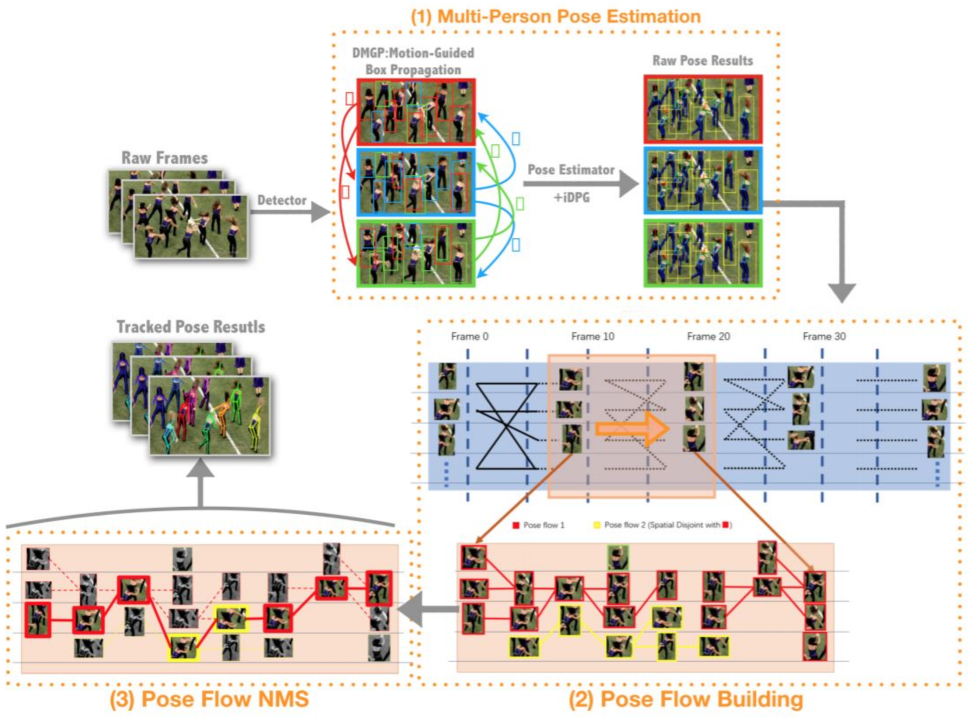

Figure 7.1: Illustration of the PoseFlow tracking process [41]

shown in figures 7.2. On average, it took 65ms to process each pair of frames; limiting the search to areas near the matched poses might make even lower feature numbers possible, and a GPU implementation would be even more ideal. The colors of the person poses correspond to their respective IDs. A cross-frame ID system could be built using such a tracking principle, assuming that a good balance of accuracy and performance can be obtained. Fully investigating this possibility is left as future work.

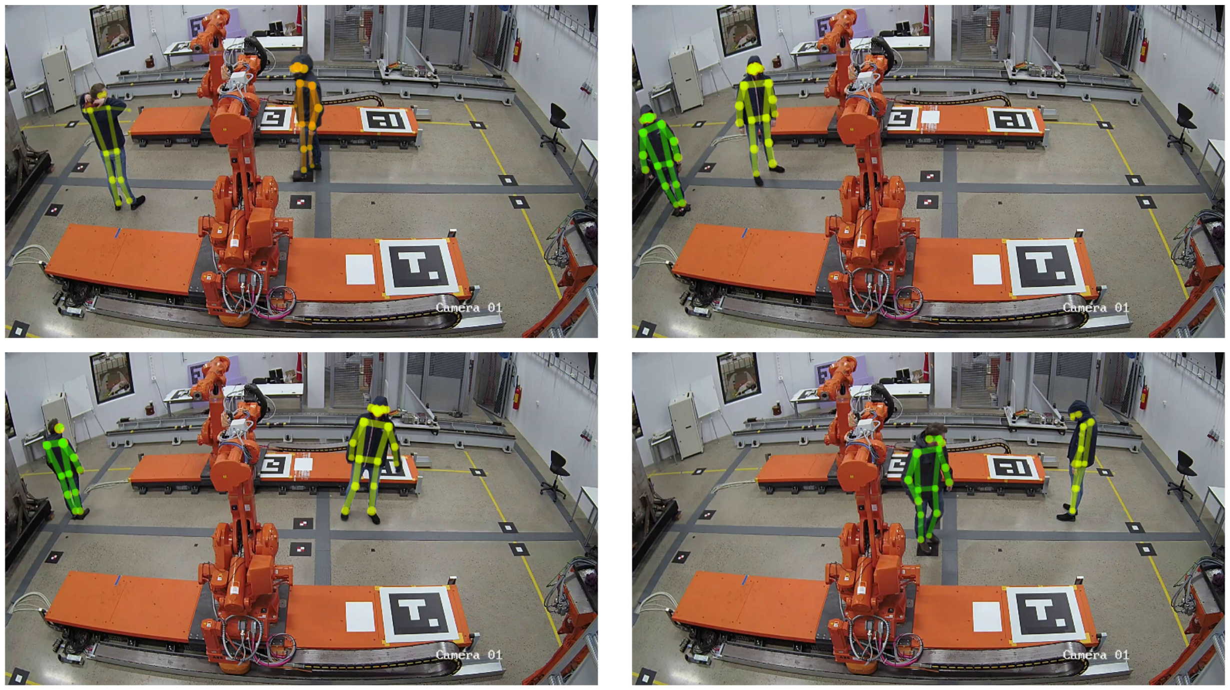

Figure 7.2: Illustration frames from the video with applied tracking

As PoseFlow ships as part of the AlphaPose pose estimation framework [29], it is bound by the same license terms as AlphaPose, with the package being free for non-commercial use. Commercial terms are not public and must be agreed upon with the researchers directly.

Retraining model for specific needs

Not all detected keypoints are used for positioning (like hands and elbows), and others are possibly redundant (eyes and nose). If the model can be retrained to only detect the useful keypoints, the model may perform faster. Investigations could be carried out to figure out exactly which keypoints are required for acceptable model accuracy, and the model would only have to determine the locations of these keypoints. Alternatively, the model could be trained to better recognize feet or persons lying down, to gain better performance in these areas if required.

Chapter 8

Discussion

Human Pose Estimators (HPE)

Compared to using bounding box detectors, a HPE performs better when trying to detect partially obscured persons, while also giving more information about which parts of the person are visible. HPEs still have problems in certain cases, nevertheless, the field has seen great advancements in recent years, and future HPEs might reduce or get rid of the current issues.

The HPE is robust enough to tackle humans wearing equipment, including helmets and jackets, as long as the result looks humanoid in appearance. This makes it suitable for industrial environments.
The human confidence score, which is useful for filtering away false human detection, is not well explained in the documentation. Both in the OpenPose architecture and in competing systems like AlphaPose, there is no clear upper limit to this score. The chosen threshold value of 0.8 may or may not generalize well to future scenarios.

Regarding the issue when detection fails when only legs are visible, the issue could be mitigated by training the network to recognize feet, but this could also harm detection accuracy on full body images. The reference OpenPose implementation, which was not utilized in this project, features a custom foot detection framework and dataset [28], which might be useful if this issue is to be improved on. For the current system, this is an inherent limitation in the way the HPE operates.

Dynamic position estimation

As shown in section 5.3 page 40, the position estimation oscillates in x axis when the person is moving. To gain a better position estimate a Kalman filter is adviced. This thesis focused on position estimation and not filtering, therefore the Kalman filter was not implemented, to keep the results unfiltered.

3D Position estimation

In the theory chapter a full 3D position estimation method was proposed. This method was tested, however the method was not stable enough to be implemented. One of the issues might be that it uses only two key-points, top of the head and feet. If more keypoints were used, it might stabilize the estimation. Nevertheless, to prove that the method works in theory with two keypoints, manual calculations have been done and are shown in appendix B.

Edge Computing

The overall performance is slower than on the workstation. Once multi-person tracking is implemented, smaller image sizes will likely be required to run the detector at required accuracy and speed. However, if the number of cameras increase or redundancy is required, a high number of nodes would perform faster than a similar computation on the workstation alone. Computer performance would gradually degrade with each camera added, while the equivalent degradation caused by more nodes would be much lower. If multiple nodes are to be implemented, each node should only cover a small area, allowing it to process the area quickly to within the desired detection accuracy. For large camera images, a single image could be divided among multiple nodes.

Licensing

Open-source HPEs like OpenPose [28] and AlphaPose [29] are freely available for non-commercial use and academic research, but come with restrictions for usage in commercial environments.

The license of the original OpenPose only allows for non-commercial use, and provides a contact link on its Github Page [36] for interested commercial parties. The commercial license of OpenPose carries costs of $25000 per year. Similar restrictions likely apply to the CMU pose model, which ships with OpenPose and can also be used as part of other pose libraries.

tf-pose-estimation is developed independently of the reference OpenPose implementation. According to its license page on GitHub [37], it is provided under the Apache License 2.0 [42], which allows for commercial usage as long as the original creators are credited and no claim to ownership is made. However, as the library can be considered a simplified subset of OpenPose, the restrictions of OpenPose may apply for this library as well. In particular, the OpenPose license considers all ”derivatives” of the software to be covered by its license, where a derivative is any piece of software that uses parts of the original code or its documentation. It is unclear whether this extends to the OpenPose paper itself. It should be noted that this report does not use the original, non-commercial OpenPose CMU model, but instead uses the simplified Mobilenet-thin model [40], which is also provided under the Apache 2.0 license.

Prior to further implementation, the OpenPose developers should be contacted, following the process listed on its official GitHub page [36].

Chapter 9

Conclusion

In this project a method for monocular real-time position estimation of occluded humans has been developed. The developed system is able to determine the position of a person with ≈ 10 cm accuracy when the person is not occluded, around ≈ 20 cm when the person is partly occluded. Even with coveralls and helmet, the position estimation maintained the same accuracy for both fully visible and occluded scenarios. Using projective geometry and human pose estimation has proven to be a good solution for this problem. The system has been tested on a workstation with a Nvidia 1080Ti GPU, which resulted in a frame rate of 20 FPS for the resulting program. As hardware and pose estimation libraries are further developed and optimized, this method will become more suitable for real-time applications.

The implementation of the method was programmed to only estimate the position of one person. If pose tracking is implemented, persons can be tracked across multiple frames, and their heights can be assigned to an ID. From this, the positions of multiple persons can be estimated.

In cooperation with Sondre Ripegutu, our position estimation was fused with Sondre's multicamera system. The result of this was an even more robust estimate. While the combined system ran more slowly, it still was able to run at 10 FPS.

The Nvidia Jetson AGX Xavier module has been tested to see if it is capable of running pose inference. Using the same image resolution as on the main hardware, it held an inference frame rate of 10 FPS. Xavier nodes can be used to scale up the covered area of the system, by offloading processing from the workstation to the nodes, making it possible to increase the number of cameras even further for the same workstation hardware. If multiple cameras cover the same areas, the positions can be fused using the aforementioned multicam system. This can also provide redundancy; if one node fails, there is still a camera covering the area.

Bibliography

[1] Kjell Rohde, Tore Berg, Thomas Yost, Svein Ove Aanesland, and Gregers Kudsk. Fully automatic pipehandling systems on a 6th-generation drilling vessel. 01 2010.

[2] Erlend A. Engum, Erik Haavind, Staale Enes, and Jesper Holck. Multi-machine control leads to improved rig layout. 03 2014.

[3] National Oilwell Varco. NOV, Anti-Collision System, 2019. [Online] Available at: https://www.nov.com/Segments/Rig_Systems/Offshore/Control_Systems/Machine_Control_Systems/Anti-Collision_System.aspx [Accessed: Apr-04-2019].

[4] Satya Mallick. Image Recognition and Object Detection, 2016. [Online] Available at: https://www.learnopencv.com/image-recognition-and-object-detection-part1 [Accessed: Jan-21-2019].

[5] N. Dalal and B. Triggs. 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. IEEE.

[6] Daphne Cornelisse. An intuitive guide to Convolutional Neural Networks, 2018. [Online] Available at: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050 [Accessed: Jan-16-2019].

[7] Michael A. Nielsen. Neural Networks and Deep Learning, 2015. [Online] Available at: http://neuralnetworksanddeeplearning.com/chap1.html [Accessed: Jan-16-2019].

[8] Ian Goodfellow and Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016. [Online] Available at: http://www.deeplearningbook.org [Accessed: Jan-17-2019].

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F Pereira, C J C Burges, L Bottou, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[11] Sik-Ho Sang. Review: AlexNet, CaffeNet - Winner of ILSVRC 2012 (Image Classification), 2018. [Online] Available at: https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160 [Accessed: Jan-16-2019].

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. 5 2014.

[13] Caltech Pedestrian Detection Benchmark. [Online] Available at: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians [Accessed: Jan-11-2019].

[14] The PASCAL Visual Object Classes Homepage, 2012. [Online] Available at: http://host.robots.ox.ac.uk/pascal/VOC [Accessed: Jan-11-2019].

[15] Jonathan Hui. What do we learn from single shot object detectors (SSD, YOLOv3), FPN & Focal loss (RetinaNet)?, 2018. [Online] Available at: https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d [Accessed: Jan-11-2019].

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. IEEE, 6 2016.

[17] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. Technical report, Washington University, 4 2018. [Online] Available at: http://arxiv.org/abs/1804.02767 [Accessed: Jan-21-2019].

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In *European Conference on Computer Vision 2016*, 12 2016.

[19] SH Tsang. Review: SSD — Single Shot Detector (Object Detection), 2018. [Online] Available at: https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11 [Accessed: Jan-23-2019].

[20] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. IEEE, 6 2014.

[21] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448. IEEE, 12 2015.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 6 2017.

[23] Jonathan Hui and Deep Learning Mar. Object detection : speed and accuracy comparison (Faster R-CNN , R-FCN , SSD , FPN , RetinaNet and YOLOv3), 2018. [Online] Available at: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 [Accessed: Jan-07-2019].

[24] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. *R-FCN: Object Detection via Region-based Fully Convolutional Networks.* PhD thesis, Microsoft Research, Tsinghua University, 5 2016.

[25] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988. IEEE, 10 2017.

[26] MPII Human Pose Database. [Online] Available at: http://human-pose.mpi-inf.mpg.de/ [Accessed: Jan-9-2019].

[27] Mykhaylo Andriluka, Umar Iqbal, Eldar Insafutdinov, Leonid Pishchulin, Anton Milan, Juergen Gall, and Bernt Schiele. PoseTrack: A Benchmark for Human Pose Estimation and Tracking. 10 2017.

[28] Zhe Cao, Tomas Simon, Shih En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:1302–1310, 12 2017.

[29] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. Rmpe: Regional multi-person pose estimation. In *ICCV*, 2017.

[30] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. DensePose: Dense Human Pose Estimation In The Wild. 2 2018.

[31] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* Springer Publishing Company, Incorporated, 1st edition, 2013.

[32] JEN BACHER. Lens Distortion: What Every Photographer Should Know, 2014. [Online] Available at: https://clickitupanotch.com/lens-distortion/ [Accessed: Feb-10-2019].

[33] Camera Calibration. Camera Calibration and 3D Reconstruction, 2014. [Online] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html [Accessed: Jan-28-2019].

[34] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*, 2009.

[35] OpenCV. Image- and World-coordinates correspondence. [Online] Available at: https://docs.opencv.org/3.3.0/dc/d2c/tutorial_real_time_pose.html [Accessed: Feb-20-2019].

[36] OpenPose GitHub repository, 2018. [Online] Available at: https://github.com/CMU-Perceptual-Computing-Lab/openpose [Accessed: Apr-04-2019].

[37] GitHub - ildoonet/tf-pose-estimation: Deep Pose Estimation implemented using Tensorflow with Custom Architectures for fast inference. [Online] Available at: https://https://github.com/ildoonet/tf-pose-estimation [Accessed: Feb-15-2019].

[38] GitHub - tensorlayer/openpose-plus: High-Performance and Flexible Pose Estimation Framework using TensorFlow, OpenPose and TensorRT. [Online] Available at: https://github.com/tensorlayer/openpose-plus [Accessed: Feb-22-2019].

[39] COCO - Common Objects in Context - Keypoint Evaluation. [Online] Available at: http://cocodataset.org/#keypoints-eval [Accessed: Mar-25-2019].

[40] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 4 2017.

[41] Yuliang Xiu, Jiefeng Li, Haoyu Wang, Yinghong Fang, and Cewu Lu. {Pose Flow}: Efficient Online Pose Tracking. In *BMVC*, 2018.

[42] Apache License, Version 2.0. [Online] Available at: https://www.apache.org/licenses/LICENSE-2.0 [Accessed: Mar-4-2019].

Appendix A

Calibration

Intrinsic parameters

The intrinsic matrix and distortion parameters:

$$\text{Intrinsic matrix} = \begin{bmatrix} 1317.07992 & 0 & 989.3528 \\ 0 & 1324.3600 & 581.9859 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

$$\text{Radial distortion parameters} = \begin{bmatrix} -0.3495 & 0.1053 \end{bmatrix} \tag{A.2}$$

The radial distortion parameters correspond to the k_1 and k_2 parameters from the OpenCV distortion model [33],as also explained in the theory section.

Extrinsic matrix

Matrix calculated from solvePnP using landmarks:

$$\begin{bmatrix} 0.03442472 & -0.99871261 & -0.03725673 & 4.98542743 \\ -0.71656598 & 0.00132136 & -0.69751806 & 3.30740039 \\ 0.69666932 & 0.05070877 & -0.71559799 & 3.9774644 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

Appendix B

2D to 3D position estimation

May 20, 2019

```
In [1]: from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.pyplot as plt
        import numpy as np
        import sympy as sp
        %matplotlib inline

In [2]: # Define camera parameters
        # Assume a image resolution of 640x480
        # Principal point is therefore 320x240
        C = np.array([[500, 0, 320],
                      [0, 500, 240],
                      [0, 0, 1]])
        # Set a seed to avoid potential errors with changing values
        np.random.seed(0)
        R = np.random.rand(3,3)
        t = np.random.rand(3,1)
        # Concatenate rotation matrix and translation vector
        extrinsic = np.hstack((R, t))
        # Define the global coordinates
        x, y = 2.0, 3.0
        z1, z2 = 1.0, 2.0
        # Z corresponds to the height of the person
        Z = z2 - z1
        # Define two global points
        P1 = np.array([[x], [y], [z1], [1]])
        P2 = np.array([[x], [y], [z2], [1]])
        # Calculate the uv-points
        uv1 = C @ extrinsic @ P1
        uv2 = C @ extrinsic @ P2
        uv1 = np.divide(uv1, uv1[2]).round()
        uv2 = np.divide(uv2, uv2[2]).round()
        print("Homogeneous pixel coordinates")
        sp.pprint([uv1, uv2])

Homogeneous pixel coordinates
[[739.] , [[722.]
 [617.]    [610.]
 [  1.]]   [  1.]]
```

1

```
In [3]: ## Now calculate back the position.
        # Calculate the inverse of rotation and intrinsic matrix
        invC = np.linalg.inv(C)
        invR = np.linalg.inv(R)
        A1 = invR @ invC @ uv1
        A2 = invR @ invC @ uv2
        B = invR @ t
```

```
In [4]: # Define everything as sympy variables and matrices
        dZ = sp.Symbol('dZ')
        invC = sp.Matrix(invC)
        invR = sp.Matrix(invR)
        A1 = sp.Matrix(A1)
        A2 = sp.Matrix(A2)
        B  = sp.Matrix(B)
        t  = sp.Matrix(t)
```

```
In [8]: p1 = invR @ ((dZ + B[2,0])/A1[2,0] * invC @ uv1 - t)
        p2 = invR @ ((Z + dZ + B[2,0])/A2[2,0] * invC @ uv2 - t)
        sp.pprint(p1)
        print("\n")
        sp.pprint(p2)
```

```
1.25914469013157dZ + 0.715888718009561

0.763586256576719dZ + 2.19831404062789

     1.0dZ + 2.66453525910038e-15


0.878132607928419dZ + 1.10768021697801

0.537174088710764dZ + 2.44648536552801

      1.0dZ + 0.999999999999999
```

```
In [7]: # Solve for delta Z
        dZ_estimated = sp.solve(p1[0] - p2[0], dZ)
        print("Estimated Points")
        print("Point 1\t\t       Point 2")
        sp.pprint([p1.evalf(subs={dZ: dZ_estimated[0]}), p2.evalf(subs={dZ: dZ_estimated[0]})]
        # Compare it to the original point
        print("Actual Values")
        print("\nPoint1\tPoint 2")
        sp.pprint([sp.Matrix(P1[0:3]), sp.Matrix(P2[0:3])])
```

```
Estimated Points
Point 1                      Point 2
```

```
2.0106565445631   2.0106565445631

2.98350332472096, 2.99885693392039

1.0282915352788   2.0282915352788
Actual Values

Point1        Point 2
2.0  2.0

3.0, 3.0

1.0  2.0
```