POLITECNICO DI TORINO

Master's Degree in MECHATRONIC ENGINEERING

Master's Degree Thesis

Efficient People 4D pose estimation and tracking for social controllers

Supervisors

Prof. Marcello CHIABERGE

PhD Mauro MARTINI

Dott. Chiara BORETTI

Candidate

Pietro VIGNINI

ACADEMIC YEAR 2023 - 2024

Abstract

As social robots increasingly engage with humans in dynamic environments, one of the main challenges is to develop a perception system that can detect and track people's position, velocity and orientation in real time.

By integrating RGBD data with multi-object tracking frameworks, this work seeks to provide a reliable solution for 4D pose estimation.

The tracking-by-detection paradigm, which separates the detection phase from the tracking phase, has established itself as one of the most used approaches for online, real-time multi-object tracking applications. Following this paradigm, three methods were studied, developed and tested, in a progressive approach to improve the tracking accuracy and the quality of the estimated 4D poses.

In the first method, YOLOv8 Segmentation was combined with a modified version of SORT tracking algorithm; the segmentation masks provided by YOLOv8 were used to extract the centroids of the people in the scene. These 2D centroids were then deprojected to 3D points using the depth map from the RGBD camera to obtain 3D positions to be fed to the tracking algorithm.

To improve identity association and reduce ID switches during the tracking phase, in the second method SORT was replaced with StrongSORT, a more advanced tracking algorithm that integrates a Re-ID model to exploit visual features to associate detections to tracks. In both methods, a person's orientation was obtained as the angle described by the estimated velocity vector.

To further refine orientation accuracy, in the third method YOLOv8 Pose was used to extract people body keypoints to directly estimate the orientation. Keypoints were also used to obtain the 3D positions of the detected people.

To evaluate and compare the different methods, a dataset consisting of multiple RGBD videos was recorded, capturing different levels of complexity in terms of number of people and occlusions. Ground truth data was obtained with a motion capture system to quantitatively determine the accuracy of estimated positions, velocities and orientations. The computational efficiency of each method was also measured to verify real-time capabilities.

Table of Contents

List of Figures

Chapter 1

Introduction

In recent years, social robotics, the branch of robotics that studies and designs robots that interact with humans in social contexts, has gained increasing importance in various fields, with applications ranging from logistics to healthcare, entertainment, sales support, and domestic care. In these environments, robots are required to navigate around humans, and safe and smooth Human-Robot Interaction (HRI) is critical to ensure socially acceptable, efficient and accident-free autonomous navigation [1].

To make robots successfully navigate around humans in dynamic environments, one of the main challenges is to develop a perception system that can detect and track people's position, velocity, and orientation in real time. This knowledge is fundamental to enable the two subsequent steps: prediction, which aims to model how humans will move into the future, and planning, which determines the future actions towards the robot's goal [2]. The ability of a robot to understand not only where people are, but also in which direction they are moving and their speed, allows for more informed decisions and more natural behaviors.

To address this challenge, this thesis work, developed at PIC4SeR (PoliTo Interdepartmental Centre for Service Robotics), aims to study and develop efficient methods for real-time 4D pose estimation and tracking that can be deployed on social robots operating in dynamic environments with multiple people.

Following a progressive approach to improve tracking accuracy while maintaining real-time performance, three different methods were developed and tested. The first method combines YOLOv8 Segmentation with a modified version of the SORT tracking algorithm to establish a baseline approach for integrating RGBD data with multi-object tracking frameworks. The second method enhances tracking consistency by replacing SORT with StrongSORT, a more advanced tracking algorithm that integrates visual re-identification features. The third method further refines the system by replacing the segmentation model with YOLOv8 Pose to directly estimate orientations through body keypoint analysis.

To quantitatively evaluate and compare the methods, a dataset consisting of multiple RGBD videos was recorded, and the motion capture system installed at PIC4SeR laboratory was used to obtain ground truth data for people's positions, velocities, and orientations.

1.1 Thesis Organization

The thesis is organized as follows:

- **Chapter 2** presents the core computer vision tasks used to detect, locate, analyze and recognize people in images and videos. These include object detection, instance segmentation, pose estimation, and person re-identification.
- **Chapter 3** introduces the multi-object tracking problem and the tracking-by-detection paradigm. It then describes the original formulation of the two tracking algorithms, SORT and StrongSORT, that were modified and integrated into the developed methods.
- **Chapter 4** details the three methods developed in this work, describing how RGBD data was integrated with the tracking frameworks to obtain people 4D pose estimation and tracking.
- **Chapter 5** presents the experimental setup, including the optimization process for the deep learning models, the hardware components and sensors used, and software framework used to develop and test the algorithms.
- **Chapter 6** discusses the experimental testing phase, describing the evaluation metrics, the recorded dataset, and presenting the results of the evaluation of the three methods.
- **Chapter 7** concludes the thesis by summarizing the main results and proposing potential directions for future work.

Chapter 2

Visual Perception

This chapter presents an overview of the fundamental computer vision tasks and methods that form the foundation of the tracking system developed in this work. We first introduce object detection and instance segmentation, which enable the system to identify and locate people in images. We then discuss pose estimation for understanding human body configurations, and person re-identification for maintaining consistent identities across frames. These elements represent the essential visual perception components that, when integrated with tracking algorithms, allow the development of a complete people 4D pose estimation and tracking system.

2.1 Object Detection

Object detection represents a fundamental task in computer vision. Its main goal is to find and recognize one or more items within an image or video, by combining object localization and classification.

Image classification is the task where given an input image, the classification algorithm outputs the class of object shown. In short, it answers the question of "what" objects are present. On the other hand, object localization is the task of identifying the position of an object by drawing a bounding box around it. It answers the question of "where" the object is [3].

Unlike image classification, which assigns a single label to an entire image, object detection must handle multiple instances of potentially different object classes simultaneously. This makes it a more complex task that requires different approaches with respect to the two tasks it builds on. Historically, various fields including Machine Learning, Artificial Intelligence, Pattern Recognition, and Image Processing have all provided methods that have been used extensively to perform object detection. The evolution of these approaches has led to significant improvements in detection accuracy and efficiency, particularly with the advent of deep

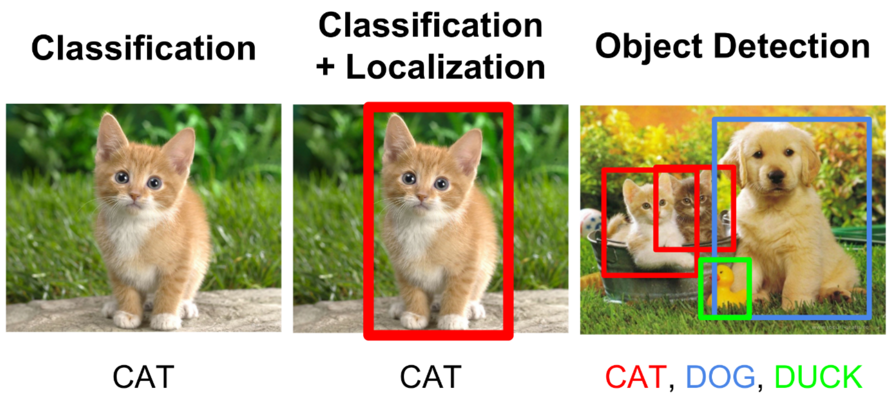


Figure 2.1: Difference between Object Classification, Object Localization and Object Detection [4].

learning techniques [5].

The output of an object detection system typically includes:

1. **Bounding Boxes**: Rectangular coordinates (u, v, width, height) that define the spatial location of each detected object in the image plane. (u, v) are usually the pixel coordinates of the bounding box center.
2. **Class Labels**: The category assigned to each detected object.
3. **Confidence Scores**: A measure of the model's certainty for each detection.

To evaluate detection performance, several metrics are used:

- **Intersection Over Union (IOU):** this metric measures the overlap between predicted and ground truth bounding boxes (Figure 2.2). Given a predicted bounding box B_p and the corresponding ground truth B_{gt}, IOU is calculated as:
$$IOU = \frac{\text{Area}(B_p \cap B_{gt})}{\text{Area}(B_p \cup B_{gt})} \tag{2.1}$$

- **Precision and Recall:** precision measures the ability of the detector to produce predictions that correspond to true instances, while recall measures the ability to find all instances. They are defined as:
$$Precision = \frac{TP}{TP+FP}, \qquad Recall = \frac{TP}{TP+FN} \tag{2.2}$$
where TP represents the number of true positives, FP the number of false positives, and FN the number of false negatives. A prediction is considered a TP if its IOU with the B_{gt} exceeds a predefined threshold (commonly 0.5).

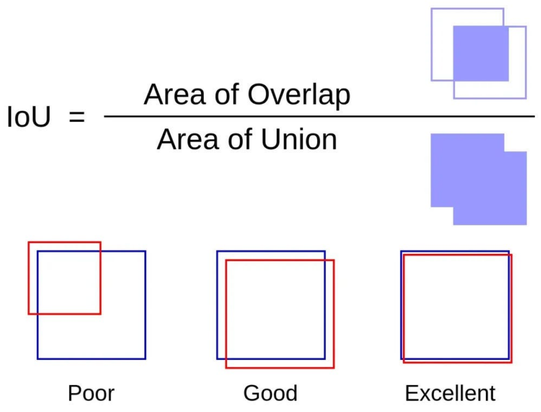


Figure 2.2: Intersection Over Union (IOU) metric [6].

- **Average Precision (AP):** This metric combines both precision and recall, and it is defined as the area under the precision-recall curve. A high AP value indicates that the detector maintains high precision (low false positives) even at high recall levels (finding most true objects).

  For multi-class object detection, to provide a comprehensive evaluation of the model's performance, the mean average precision (mAP) is commonly adopted, which averages the AP across all classes. mAP is usually computed at different IOU thresholds:

  mAP50: mean average precision calculated at an IOU threshold of 0.5. It measures the model's accuracy on easy detections.

  mAP50-95: : mean average precision over different IoU thresholds from 0.5 to 0.95 in steps of 0.05. This provides a more comprehensive evaluation of detector performance across different levels of prediction confidence.

2.1.1 Deep Learning for Object Detection

In recent years, deep learning techniques, in particular Convolutional Neural Networks (CNNs), have transformed the object detection task, leading to a significant improvement in performance. Unlike traditional methods based on hand-crafted features and simple classifiers (such as Support Vector Machines (SVM), and Adaboost) [7], CNNs can automatically extract hierarchical, high-dimensional features from the image, recognizing patterns like edges and corners in their first layers and more complex and abstract features such as shapes or object parts in the deeper layers. The key elements of a CNN are (Figure 2.3):

- **Convolutional layers:** these apply a series of filters that the network learns during training to the input image to detect features like edges, textures, and complex patterns depending on the location of the layer in the network.

- **Non-linear activation functions:** functions like the ReLU (Rectified Linear Unit), process the feature maps produced by the convolutional layers to introduce non-linearity in the network. This allows the model to learn complex patterns.

- **Pooling layers:** they reduce the spatial dimension of the feature maps while maintaining the important features. This downsampling operation makes the model more robust to spatial variations and is needed to pass from the input image consisting of millions of pixels to the desired output (object location and class).

- **Fully connected layers:** In the final stages of the network, these layers are used to combine the extracted features for the final predictions about object locations and classes.

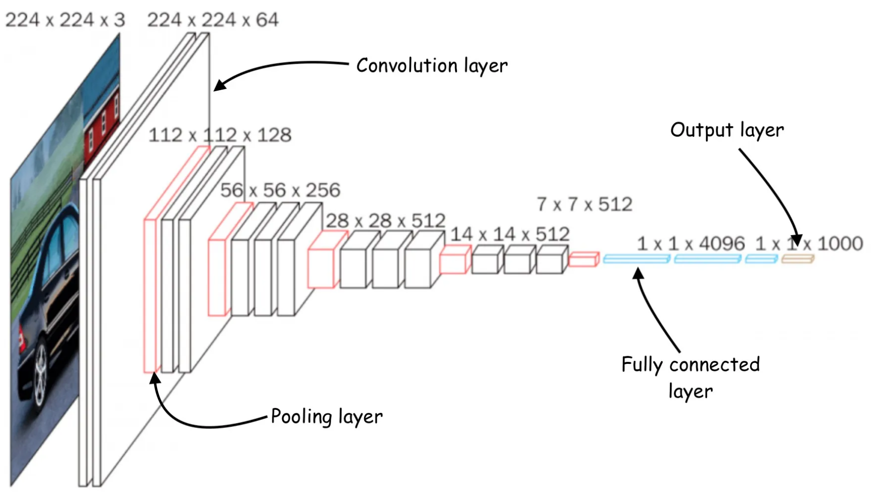


Figure 2.3: Example of the architecture of a Convolutional Neural Network (CNN) [8].

Object detectors based on deep learning can be divided into two classes:

- **Two-stage detectors** (like the R-CNN family), which first generate region proposals (regions of interest) and then classify them.

- **One-stage detectors** (like YOLO, SSD, RetinaNet), which predict bounding boxes and class scores directly in a single forward pass.

2.1.2 YOLOv8

The first YOLO model (You Only Look Once) was introduced by Redmon et al. in 2015 [9], and revolutionized object detection by reframing it as a single regression problem, predicting bounding boxes and class probabilities directly from image pixels. Compared to state-of-the-art detection systems at the time, YOLO was extremely faster while achieving comparable mAP.

The original YOLO architecture (Figure 2.4) divided the input image into a grid of SxS cells, and for each cell the model tried to predict B bounding boxes and class probabilities for objects centered in that cell. Therefore, each prediction included six numbers: the four coordinates of the bounding box, the class of the object and the confidence score about the model's certainty that the box contains the object. This prediction is performed directly on the last feature map, generated by the convolutional backbone [10].

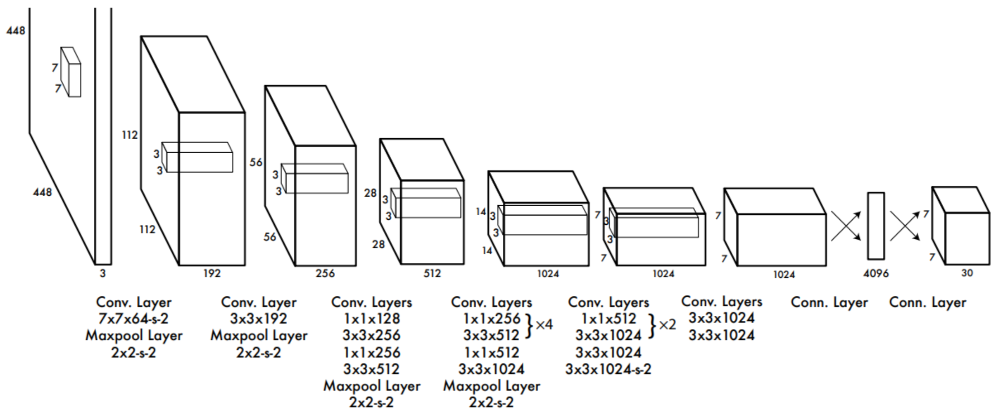

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 2.4: Architecture of the first YOLO network [9].

Despite its speed, the original YOLO model had limitations, particularly in detecting small objects or objects that were close together because of the constraint of predicting a single class per grid cell.

In the following years, many improved versions of YOLO have been proposed in literature, and YOLOv8 [11], developed by Ultralytics and released in 2023, represents one of the latest evolutions of the YOLO models family. It introduced several improvements in the architecture:

- A more efficient backbone based on CSPDarknet.
- A new detection head with better handling of different scales.
- Path Aggregation Network (PAN) for improved feature fusion.

- Native support for additional tasks including instance segmentation and pose estimation thanks to the dynamic architecture and task-specific heads.
- Advanced training strategies such as AutoAugment for data augmentation and knowledge distillation for model compression.

With these improvements, YOLOv8 achieved state-of-the-art performance while maintaining fast inference speed.

YOLOv8 comes in five different sizes (nano, small, medium, large and extra-large), to cover different trade-offs between accuracy and computational cost. Table 2.1 shows a comparison of the different pretrained models in terms of mAP50-95 computed on the validation set of COCO2017 dataset [12], number of parameters of the network and FLOPs (Floating Point Operations, and index of computational complexity).

Table 2.1: Performance comparison of YOLOv8 detection models on COCO2017 dataset.

Model	Size	$mAP_{50\text{-}95}$	Parameters	FLOPs
	(pixels)	(%)	(M)	(B)
YOLOv8n	640	37.3	3.2	8.7
YOLOv8s	640	44.9	11.2	28.6
YOLOv8m	640	50.2	25.9	78.9
YOLOv8l	640	52.9	43.7	165.2
YOLOv8x	640	53.9	68.2	257.8

2.2 Object Segmentation

Object (instance) segmentation extends object detection by adding, for every detected object, a pixel-wise mask that shows the shape of the object. While object detectors predict where objects are through bounding boxes, models for instance segmentation are also able to determine the exact pixels that belong to each object (Figure 2.5).

To evaluate instance segmentation models, the metrics used in object detection (see Section 2.1) are extended to measure mask quality. In particular, the primary metric is mask mAP, defined as for object detection but using the IOU between predicted and ground truth segmentation masks instead of between bounding boxes.

YOLOv8, thanks to its dynamic architecture, includes also a segmentation head that predicts pixel-wise masks. Like the base detection models, YOLOv8-seg is available in different sizes. Table 2.2 shows the performance of the different pretrained YOLOv8 segmentation models on the COCO2017 dataset.

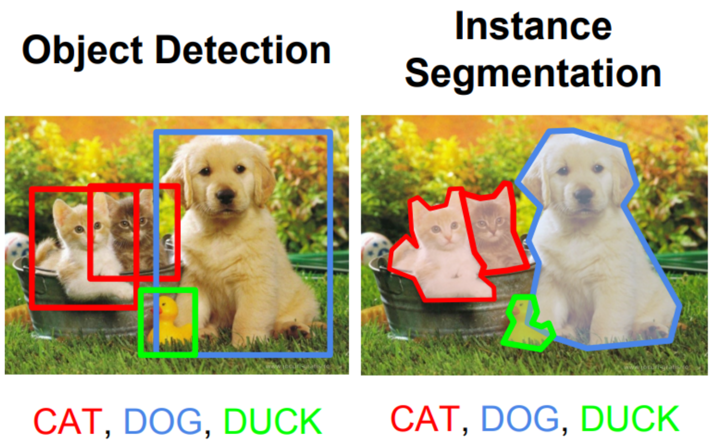


Figure 2.5: Difference between Object Detection and Instance Segmentation [4].

Table 2.2: Performance comparison of YOLOv8 Segmentation models on COCO2017 dataset.

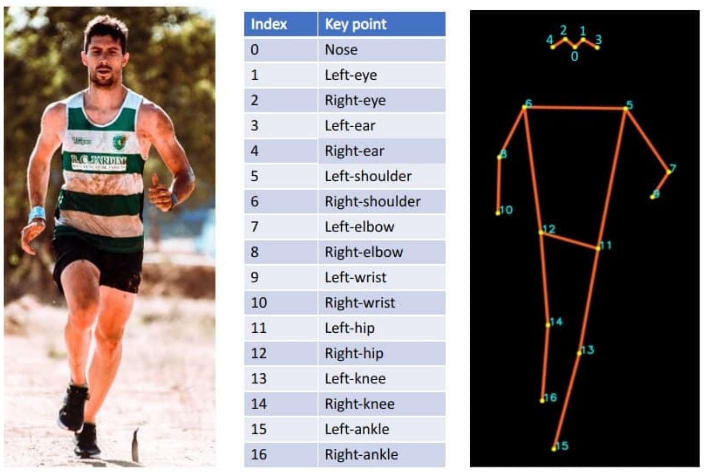
Model	Size (pixels)	$\mathrm{mAP}^{\mathrm{mask}}_{50\text{-}95}$ (%)	Parameters (M)	FLOPs (B)
YOLOv8n-seg	640	30.5	3.4	12.6
YOLOv8s-seg	640	36.8	11.8	42.6
YOLOv8m-seg	640	40.8	27.3	110.2
YOLOv8l-seg	640	42.6	46.0	220.5
YOLOv8x-seg	640	43.4	71.8	344.1

For this thesis work, YOLOv8s-seg model was selected to perform people segmentation, as it provided the best compromise in terms of accuracy and inference speed for the used hardware.

2.3 Pose Estimation

Human pose estimation is the task in computer vision that consists of identifying and localizing key body points (keypoints) of people in an image or video. Typically, these keypoints correspond to body joints like shoulders, elbows, knees, hips, and facial points like nose, ears and eyes. These keypoints are then connected to form a skeleton shape, which can be used to obtain information about people's poses and orientation [13].

As with object detection, deep learning-based methods have taken over from traditional methods for human pose estimation based on handcrafted features, due



Index	Key point
0	Nose
1	Left-eye
2	Right-eye
3	Left-ear
4	Right-ear
5	Left-shoulder
6	Right-shoulder
7	Left-elbow
8	Right-elbow
9	Left-wrist
10	Right-wrist
11	Left-hip
12	Right-hip
13	Left-knee
14	Right-knee
15	Left-ankle
16	Right-ankle

Figure 2.6: Example of human pose estimation, with keypoints in the COCO format [14].

to their ability to extract features implicitly from the data, thus being able to better handle complex scenes and poses.

For evaluation, the most popular metrics for 2D pose estimation models are the Object Keypoint Similarity (OKS), which measures how close predicted keypoints are to their ground truth locations, normalized by the person's size to make the metric scale invariant, and the mAP, usually measured at different OKS thresholds.

YOLOv8 includes also a pose estimation head that predicts keypoints 2D location and confidence score. YOLOv8-pose can detect up to 17 keypoints per person following the COCO keypoint format (Figure 2.6):

- 5 facial keypoints (nose, eyes, ears)
- 6 body keypoints (shoulders, elbows, wrists)
- 6 leg keypoints (hips, knees, ankles)

Like the detection and segmentation variants, YOLOv8-pose comes in different sizes to balance accuracy and computational requirements. Table 2.3 shows the performance comparison of different pretrained YOLOv8 pose models.

For this thesis work, the YOLOv8s-pose model was selected to perform keypoint detection, as it provided the best compromise in terms of accuracy and inference speed.

Table 2.3: Performance comparison of YOLOv8 Pose models on COCO2017 keypoints dataset.

Model	Size (pixels)	$\mathrm{mAP}^{\mathrm{pose}}_{50\text{-}95}$ (%)	Parameters (M)	FLOPs (B)
YOLOv8n-pose	640	50.4	3.3	9.2
YOLOv8s-pose	640	60.0	11.6	30.2
YOLOv8m-pose	640	65.0	26.4	81.0
YOLOv8l-pose	640	67.6	44.4	168.6
YOLOv8x-pose	640	69.2	69.4	263.2

2.4 Person Re-Identification (Re-ID)

Person Re-Identification is a task in computer vision whose goal is to determine, given a query person-of-interest, whether this person has appeared in another place at distinct time captured by a different camera, or even the same camera at a different time instant [15]. Unlike classic object detection or classification, Re-ID must learn discriminative features that are robust to variations in viewing angle, illumination, and pose, while being specific enough to distinguish between similar-looking objects.

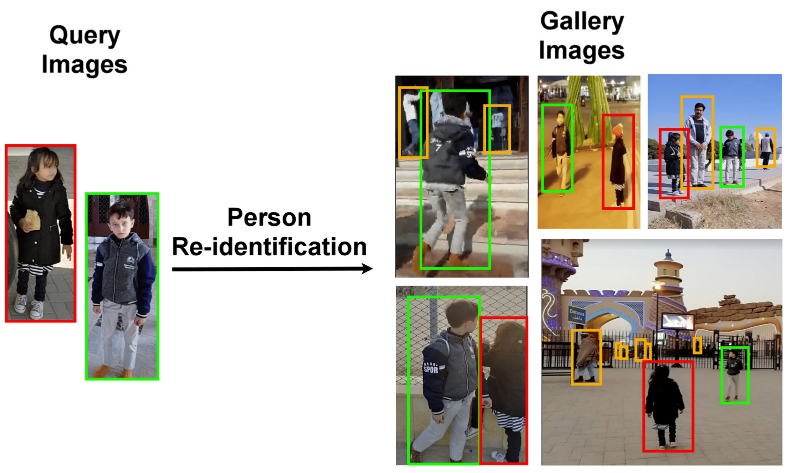


Figure 2.7: Example of person Re-Identification (Re-ID) task [16].

Traditional person Re-ID methods mainly used manual extraction of fixed discriminative features. Deep learning has led to a major improvement, allowing automatic learning of robust and discriminative features.

Modern Re-ID networks are designed to learn an embedding space where images of the same identity are mapped close together, while different identities are mapped far apart. This is achieved through convolutional neural networks (CNNs) that transform input images into compact feature vectors, typically 512-dimensional.

Image similarity in the embedding space is measured with the cosine distance:

$$d(\boldsymbol{a}, \boldsymbol{b}) = 1 - \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \|\boldsymbol{b}\|} \tag{2.3}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the feature vectors of two images. A smaller distance indicates higher similarity.

To evaluate Re-ID models, the test set is commonly split into a query set and a gallery set. The query set is made of multiple images of different people that the model is asked to identify. The gallery set contains a larger set of images with at least one image corresponding to each query image, along with distractor images of other identities. For each query image, the model generates a feature vector. This vector is compared to the embeddings from all images in the gallery set using the cosine distance, and gallery images are ranked based on their similarity to the query feature vector.

To quantitatively evaluate the model, the following metrics are used:

- **Cumulative Matching Characteristic (CMC):** for each query image, a correct match is counted if at least one image of the same person appears in the top-k ranked gallery images. CMC Rank-k represents the percentage of queries that have at least one correct match in their top-k results. A commonly reported value is Rank-1 accuracy, which indicates the percentage of queries for which the correct match is ranked first.

- **Mean Average Precision (mAP):** this metric evaluates the retrieval quality across the entire ranked list, measuring both precision and recall.

In this thesis work we used the OSNet Re-ID network [17]. In particular, the x0.25 model, with 2.2M parameters was chosen. This lightweight CNN is designed to extract omni-scale features, which combine both homogeneous and heterogeneous scales: homogeneous scales correspond to visual patterns of a single scale (like local details such as shoes and glasses, or global features like body shape), while heterogeneous scales represent patterns that span multiple scales simultaneously (like a white t-shirt with a specific logo in the front). This ability to capture and combine different scales allowed the model to achieve state-of-the-art performance despite its much smaller size if compared to other popular Re-ID networks like ResNet50-based models.

Chapter 3

Multi-Object Tracking

Multi-Object Tracking (MOT) is the task in computer vision that involves detecting multiple objects, maintaining their identities and yielding their individual trajectories across frames in a video sequence. It is a fundamental problem for numerous applications such as autonomous driving, robotics, video surveillance and human-robot interaction. Apart from determining the number of objects, which usually varies over time, and maintaining their identities, MOT has to deal with other issues like frequent occlusions, initialization and termination of tracks, changes in objects appearance and complex objects movements [18].

Thanks to the rapid advancement of object detectors, which are capable of providing object detections with increasingly accuracy, **tracking-by-detection** paradigm has emerged as one of the most widely used approaches for online, real-time MOT applications.

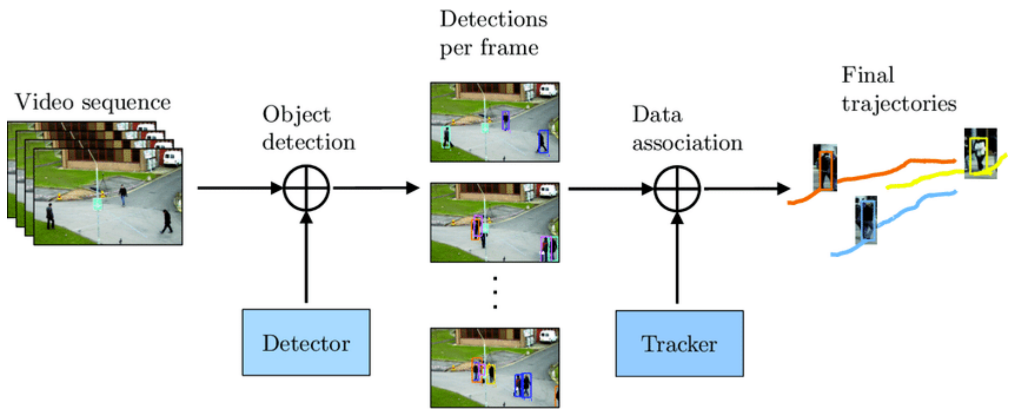


Figure 3.1: Tracking-by-detection paradigm. An object detector is used to extract object candidates from each frame of the video sequence. Then, a tracking algorithm is run on the set of detections to perform data association [19].

In this framework, object detection is performed independently at each frame using a dedicated detector, and the task of tracking is reduced to associating detected objects across consecutive frames. This modularity allows for flexibility, since advancements in object detection can directly improve the tracking pipeline without requiring significant modifications to the tracker itself. Thus, tracking-by-detection trackers mainly focus on improving data association, while exploiting deep leaning trends for the detection [20].

In recent years, trasformers have been proposed in the context of MOT to learn deep objects representations from both visual clues and object trajectories. However, they still lack performance with respect to SOTA tracking-by-detection methods, both in terms of accuracy and time efficiency [21].

3.1 Data Association in Tracking-by-Detection

The core challenge in tracking-by-detection trackers lies in the data association step, where the algorithm must determine which detections in the current frame correspond to which existing tracks from previous frames or if they represent new tracks. To do so, many modern MOT algorithms use motion models and/or appearance models and re-identification.

3.1.1 Motion Models

Motion models are used to predict where each tracked object is likely to appear in the current frame by maximizing a posterior probability.

One of the most classic and widely used motion models is the linear Kalman filter (KF, see Appendix A.1), which is a recursive Bayes filter based on the typical predict-and-update cycle. This filter is commonly used with a constant velocity assumption, which performs well at high frame rates, where the inter-frame object displacement between the current position and the next one is small, but lack robustness in non-linear motion scenarios and in the ability to recover the identity of an object after occlusions.

Because of these limitations, Extended Kalman filters (EKF) and Unscented Kalman filters (UKF) have also been proposed to handle non-linear motions, but they still require motion pattern assumptions and, because of the additional computational cost, they are rarely adopted [21]. Once predictions have been computed, the similarity scores between them and new detections are calculated using spatial proximity metrics like the Intersection-over-Union, 3D-Euclidean distance, or the Mahalanobis distance (see Appendix A.2) based on the Kalman filter's state prediction.

3.1.2 Appearance Models and Re-Identification (Re-ID)

Using deep visual features to discriminate and re-identify objects is another popular approach for the association step. Re-ID models are often implemented as deep convolutional neural networks (CNNs) that generate a feature embedding vector for each detected object. The visual appearance state of each tracked object is kept in memory and compared with the embedding vector of each detection using the cosine similarity (Equation (2.3)).

Appearance models can track objects through longer periods of occlusions and so reduce the number of identity switches, but fall short when scenes are crowded and objects are partially occluded.

For these reasons, many modern MOT algorithms (e.g. [20], [22], [23]) have tried to get the best of both worlds by combining motion models and appearance information into a unified distance metric.

3.1.3 Assignment Problem

The final stage of data association consists of solving the assignment problem, i.e., determining the optimal correspondence between detections and existing tracks.

This is done by constructing a cost matrix where each element represents the cost of assigning a particular detection to a track based on the motion and/or appearance similarity. The lower the cost, the more likely a detection corresponds to the track. The algorithm must also be able to determine when a detection should not be matched to existing tracks, as new objects can enter the scene, while other may leave it. To handle these cases, a cost threshold is normally applied to prevent associations with high cost. These unmatched detections then become candidates for new tracks, while unmatched tracks will be terminated after a given number of missed matches.

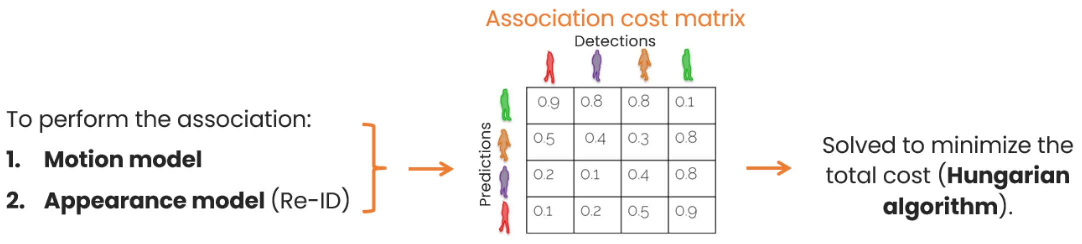


Figure 3.2: Data association in Tracking-by-Detection.

The optimization problem is then commonly solved with the Hungarian algorithm [24], which provides a solution by minimizing the total cost of matching tracks to detections while ensuring one-to-one assignments.

Once the associations have been made, the tracks' appearance vector and/or their Kalman filter state are updated. The Kalman filter integrates the new observation to refine its motion prediction for the next frame.

In the following sections, the two tracking algorithms that have been used, modified, and upon which the applications of this thesis work have been built will be discussed in their original formulation.

3.2 SORT Algorithm

Simple Online and Realtime Tracking (SORT, [25]) is a widely recognized lightweight algorithm designed for high-frame rates, online multi-object tracking. It follows the tracking-by-detection paradigm, and it uses a linear Kalman filter with constant velocity assumption to handle the motion prediction, coupled with the Hungarian algorithm for data association.

Despite its simple and pragmatic approach, SORT showed accuracy comparable to state-of-the-art online trackers at the time, while achieving a tracker update rates 20 times higher. Its influence has been considerable, and many "SORT-like" algorithms have been developed in the following years to improve the original formulation, especially in terms of association strategies and motion models, with the goal of reducing its relatively high number of identity switches and its deficiency to track through occlusions.

3.2.1 Motion Model

In SORT, as already mentioned, the inter-frame displacements of each object are approximated with a linear constant velocity model, independent of other objects and camera motion. The Kalman Filter's state vector $\boldsymbol{x}$, the output vector $\boldsymbol{y}$, the state-transition matrix $\boldsymbol{F}$, and the observation matrix $\boldsymbol{H}$ are the following:

$$\boldsymbol{x} = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^\top, \qquad \boldsymbol{y} = [u, v, s, r]^\top \tag{3.1}$$

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.2}$$

where u and v are the horizontal and vertical pixel location of the bounding box center, while s and r represent the area and aspect ratio of the bounding box, respectively. The aspect ratio is assumed to remain constant throughout tracking. Process and measurement noise covariance matrices, $\boldsymbol{Q}$ and $\boldsymbol{R}$ respectively, are defined as constant. The velocity components are solved optimally by the Kalman filter through its predict-update cycle.

If no detection is associated with a particular track at a given time step, the track's state is propagated solely using the prediction step, with no correction, based on the linear velocity model.

3.2.2 Data Association

To assign detections to tracks, SORT computes the intersection-over-union (IOU) distance between each detection and all the predicted bounding boxes. Then the assignment is solved with the Hungarian algorithm.

To prevent unlikely matches, a minimum IOU threshold IOU_{min} is imposed. Given that the IOU metric is bounded between 0 and 1 ($0 \leq IOU \leq 1$), this is equivalent to imposing a maximum cost $C_{max} = 1 - IOU_{min}$.

To handle scenarios where objects enter or leave the scene, any detection that does not overlap with any existing track by more than IOU_{min} is considered a candidate for a new object.

In such cases, a new track is initialized, with initial state derived from the detection, velocity set to zero and high covariance values for the velocity components, as the velocity is unobserved at this point.

To reduce false positives, a newly initialized track must "hit" a minimum number of matches *min_hits* before being considered a truly new object.

If a track is not matched with any detection for a number T_{lost} of consecutive frames, it is assumed that it has left the image, and the track is terminated.

The pseudo-code of this algorithm is shown in Algorithm 1.

3.3 StrongSORT Algorithm

StrongSORT [22] is a recent MOT tracking algorithms built upon SORT's efficient framework. It evolved from DeepSORT [26], the tracking algorithm that integrated appearance information and re-identification into SORT.

StrongSORT introduces multiple improvements, such as a more advanced association strategy, a Kalman filter with adaptive measurement noise covariance, and an exponential moving average (EMA) to update the feature vector of each tracked object.

Algorithm 1 Simple Online and Realtime Tracking (SORT)

```
Input: Detections $\mathcal{D} = \{d_j\}$, Previous tracks $\mathcal{T} = \{\tau_i\}$, Kalman filter KF, Cost
    threshold $C_{max}$, Probationary period $min\_hits$, Max age $T_{Lost}$
Output: Updated tracks $\mathcal{T}$
 1: $matched \leftarrow \emptyset$
 2: $unmatched\_tracks \leftarrow \mathcal{T}$
 3: $unmatched\_dets \leftarrow \mathcal{D}$
 4: for $track\ \tau_i \in \mathcal{T}$ do
 5:   $track.pred \leftarrow$ Predict track position in the current frame using KF
 6: $C \leftarrow$ empty cost matrix
 7: for $track\ \tau_i \in \mathcal{T}$ do
 8:   for $det\ d_j \in \mathcal{D}$ do
 9:     $c_{ij} \leftarrow 1 - \text{IOU}(track.pred, det.bbox)$
10:     Add $c_{ij}$ to $C$
11: $M \leftarrow Hungarian(C)$                                    {Optimal assignment}
12: for $(i, j) \in M$ do
13:   if $c_{ij} < C_{max}$ then
14:     $matched \leftarrow matched \cup \{(i, j)\}$
15:     $unmatched\_dets \leftarrow unmatched\_dets \setminus \{d_j\}$
16:     $unmatched\_tracks \leftarrow unmatched\_tracks \setminus \{\tau_i\}$
17:     Update Kalman Filter for track $\tau_i$ with detection $d_j$
18:     $track_i.hits \leftarrow track_i.hits + 1$ {Increment hit streak}
19: for $track\ \tau_i \in unmatched\_tracks$ do
20:   $track.time\_since\_update \leftarrow track.time\_since\_update + 1$
21:   if $track.time\_since\_update == T_{Lost}$ then
22:     $\mathcal{T} \leftarrow \mathcal{T} \setminus \{\tau_i\}$
23: for $det\ d_j \in unmatched\_dets$ do
24:   $track_{new} \leftarrow$ Initialize new Kalman filter track from $d_j$
25:   $track_{new}.hits \leftarrow 1$                              {Initialize hit streak}
26:   $track_{new}.state \leftarrow tentative$
27:   $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_{new}\}$
28: for $track\ \tau_i \in \mathcal{T}$ do
29:   if $track.hits \geq min\_hits$ then
30:     $track.state \leftarrow confirmed$
31: return $\mathcal{T}$
```

These enhancements aim to make StrongSORT more robust to low-confidence detections and occlusions while reducing identity switches. The main differences with respect to SORT are described in the following sections.

3.3.1 Motion Model and Appearance Model

Like SORT, StrongSORT uses a linear Kalman filter with constant velocity assumption. The state vector $\boldsymbol{x}$ is now eight dimensional:

$$\boldsymbol{x} = [u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h}]^\top, \qquad \boldsymbol{y} = [u, v, \gamma, h]^\top \tag{3.3}$$

where u and v are the horizontal and vertical pixel location of the bounding box center, while γ and h represent the aspect ratio and height of the bounding box, respectively.

The state-transition matrix $\boldsymbol{F}$ and the observation matrix $\boldsymbol{H}$ are defined similarly to those in SORT, with an additional dimension. However, StrongSORT modifies the noise covariance matrices $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ to dynamically adjust based on the estimated and measured bounding box height, respectively.

In particular:

- The process noise covariance $\boldsymbol{Q}_k$ is updated during the Kalman filter's prediction phase, as a function of the current state vector, which includes the estimated height h_{k-1} from the previous update step.
- The measurement noise covariance $\boldsymbol{R}_k$ is defined at each update phase based on the height of the newly associated bounding box.

This approach allows the filter to better account for scale-dependent uncertainties in both the motion model and measurements. The specific functions used to compute these matrices can be found in the StrongSORT GitHub repository.

Furthermore, StrongSORT proposes a formula to adaptively calculate the noise covariance matrix $\tilde{\boldsymbol{R}}_k$:

$$\tilde{\boldsymbol{R}}_k = (1 - c_k)\boldsymbol{R}_k \tag{3.4}$$

where $\boldsymbol{R}_k$ is the measurement noise covariance matrix preset on the height of the measured bounding box, and c_k is the confidence score of the detection at time k. The idea behind this adaptive approach is to make the KF less vulnerable with respect to low-quality detections. When a detection has a high score c_k, it has less noise, and so by lowering $\tilde{\boldsymbol{R}}_k$ it will have higher weight in the state update step, and vice versa.

Regarding the appearance model, StrongSORT uses a deep appearance descriptor to extract features vectors for each detection. After the association phase, it uses an exponential moving average to update the appearance state $\boldsymbol{e}_i^k$ for the i-th track at time k, as follows:

$$\boldsymbol{e}_i^k = \alpha \boldsymbol{e}_i^{k-1} + (1-\alpha)\boldsymbol{f}_i^k \tag{3.5}$$

where $\boldsymbol{f}_i^k$ is the appearance embedding of the matched detection and α is a momentum term. This updating mechanism allows the appearance model to adapt gradually, depressing detection noise and enhancing the matching quality.

3.3.2 Data Association

To incorporate motion information in the association phase, StrongSORT replaces the IOU metric used in SORT with the squared Mahalanobis distance (see Appendix A.2) between predicted Kalman filter states and newly arrived detections:

$$d^{(1)}(i,j) = (\boldsymbol{d}_j - \boldsymbol{y}_i)^\top \boldsymbol{S}_i^{-1}(\boldsymbol{d}_j - \boldsymbol{y}_i), \tag{3.6}$$

where $(\boldsymbol{y}_i, \boldsymbol{S}_i)$ is the projection of the i-th track's predicted state and state covariance matrix into the measurement space, and $\boldsymbol{d}_j$ is the j-th bounding box. The Mahalanobis distance measures how many standard deviations a detection is far from the prediction, taking state estimation uncertainty into account.

As in SORT, a threshold is applied to reject unlikely associations. Specifically, the Mahalanobis distance is thresholded at a 95% confidence interval computed from the inverse χ^2 distribution (For the four-dimensional measurement space used here, this value corresponds to a maximum Mahalanobis distance of 9.4877).

To incorporate appearance information, StrongSORT computes the cosine distance between the i-th track appearance state $\boldsymbol{e}_i$ and the appearance embedding of the detections $\boldsymbol{f}_j$:

$$d^{(2)}(i,j) = 1 - \boldsymbol{f}_j^\top \boldsymbol{e}_i \tag{3.7}$$

This metric is also subject to a threshold to exclude matches between objects with significantly different appearances. Finally, an unified cost matrix is obtained by combining the two distances with a weighted sum:

$$c_{ij} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j), \tag{3.8}$$

where λ is the hyperparameter that controls the relative importance of motion and appearance information. The assignment is solved with the Hungarian algorithm.

An association is considered valid only if it is within the threshold of both metrics.

Unlike SORT, where measurement-to-track associations are solved in a single, global assignment problem, StrongSORT introduces a more sophisticated matching cascade.

Each track in StrongSORT exists in one of three track states: *tentative*, *confirmed*, or *deleted*. When a new track is initialized, it is initially set as *tentative*. Only if it achieves n_{init} consecutive matches it becomes *confirmed*; otherwise, it is deleted. Once a track is *confirmed*, it remains in this state until it fails to match with a detection for a consecutive number of frames equal to A_{max} (same as T_{Lost} in SORT).

The matching strategy follows a cascade approach: first, the assignment problem is solved considering all detections and tracks currently in the *confirmed* state. Confirmed tracks left unmatched at this phase are then separated into two groups: those that were recently updated ($track.time\_since\_update = 1$) and those that instead are on a longer streak of non-matching ($track.time\_since\_update > 1$).

The algorithm proceeds differently with each group: tracks with longer unmatched streaks are left unmatched, while recently updated tracks are re-evaluated together with tracks currently in the *tentative* state in a second matching round with the unmatched detections from the first phase.

In this secondary assignment phase, the IOU metric is considered as distance measure, without using appearance information. This should help to account for sudden appearance changes, e.g. because of partial occlusion, and to make the algorithm more robust against false positive tracks.

Finally, detections that were not matched after both rounds of assignment are used to initialize new tracks.

The pseudo-code of this algorithm is shown in Algorithm 2.

Algorithm 2 StrongSORT

Input: Detections $\mathcal{D} = \{d_j\}$, Previous tracks $\mathcal{T} = \{\tau_i\}$, Kalman filter KF, Mahalanobis cost threshold C_{max}^{Mah}, Cosine cost threshold C_{max}^{Cos}, Probationary period n_{init}, Max age A_{max}, Hyperparameter λ

Output: Updated tracks $\mathcal{T}$

```
1: $\mathcal{T}^{conf} \leftarrow \{\tau_i$ for $track\ \tau_i \in \mathcal{T}$ if $track.is\_confirmed\}$
2: $\mathcal{T}^{tent} \leftarrow \{\tau_i$ for $track\ \tau_i \in \mathcal{T}$ if $track.is\_tentative\}$
3: $matched \leftarrow \emptyset$
4: $unmatched\_tracks \leftarrow \emptyset$
5: $unmatched\_dets \leftarrow \mathcal{D}$
6: for $det\ d_i \in \mathcal{D}$ do
7:     $det.feature \leftarrow$ Extract feature embedding
8: for $track\ \tau_i \in \mathcal{T}$ do
9:     $track.pred \leftarrow$ Predict track position in the current frame using KF
10:    $track.time\_since\_update \leftarrow track.time\_since\_update + 1$
                                                    {first round of assignment}
11: $unmatched\_tracks\_a \leftarrow \mathcal{T}^{conf}$
12: $C \leftarrow$ empty cost matrix
13: for $track\ \tau_i \in \mathcal{T}^{conf}$ do
14:     for $det\ d_j \in \mathcal{D}$ do
15:         $d^{(1)}(i,j) \leftarrow$ Mahalanobis($track.pred, det.bbox$)
16:         $d^{(2)}(i,j) \leftarrow$ Cosine($track.appearance\_state, det.feature$)
17:         $c_{ij} \leftarrow \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j)$
18:         Add $c_{ij}$ to $C$
19: $M \leftarrow Hungarian(C)$
20: for $(i,j) \in M$ do
21:     if $d^{(1)}(i,j) < C_{max}^{Mah}$ and $d^{(2)}(i,j) < C_{max}^{Cos}$ then
22:         $matched \leftarrow matched \cup \{(i,j)\}$
23:         $unmatched\_dets \leftarrow unmatched\_dets \setminus \{d_j\}$
24:         $unmatched\_tracks\_a \leftarrow unmatched\_tracks\_a \setminus \{\tau_i\}$
                                                    {second round of assignment}
25: $unmatched\_tracks\_b \leftarrow \mathcal{T}^{tent}$
26: for $track\ \tau_i \in unmatched\_tracks\_a$ do
27:     if $track.time\_since\_update == 1$ then
28:         $unmatched\_tracks\_b \leftarrow unmatched\_tracks\_b \cup \{\tau_i\}$
29:     else
30:         $unmatched\_tracks \leftarrow unmatched\_tracks \cup \{\tau_i\}$
31: $C \leftarrow$ empty cost matrix
32: for $track\ \tau_i \in unmatched\_tracks\_b$ do
33:     for $det\ d_j \in unmatched\_dets$ do
```

```
34:         $c_{ij} \leftarrow 1 - \text{IOU}(track.pred, det.bbox)$
35:         Add $c_{ij}$ to $C$
36: $M \leftarrow Hungarian(C)$
37: for $(i, j) \in M$ do
38:     if $c_{ij} < (1 - IOU_{min})$ then
39:         $matched \leftarrow matched \cup \{(i, j)\}$
40:         $unmatched\_dets \leftarrow unmatched\_dets \setminus \{d_j\}$
41:         $unmatched\_tracks\_b \leftarrow unmatched\_tracks\_b \setminus \{\tau_i\}$
42:         if $track.is\_tentative$ then
43:             $track.hits \leftarrow track.hits + 1$
                                                    {Creation and deletion of tracks}
44: $unmatched\_tracks \leftarrow unmatched\_tracks \cup unmatched\_tracks\_b$
45: for $(i, j) \in matched$ do
46:     Update Kalman Filter for track $\tau_i$ with detection $d_j$
47:     Update appearance state for track $\tau_i$ with detection $d_j$
48:     if $track.is\_tentative$ and $track.hits == n_{init}$ then
49:         $track.state \leftarrow confirmed$
50: for $track\ \tau_i \in unmatched\_tracks$ do
51:     if $track.is\_tentative$ then
52:         $track.state \leftarrow deleted$
53:     if $track.is\_confirmed$ and $track.time\_since\_update == A_{max}$ then
54:         $track.state \leftarrow deleted$
55: for $det\ d_j \in unmatched\_dets$ do
56:     $track_{new} \leftarrow$ Initialize new Kalman filter track from $d_j$
57:     $track_{new}.hits \leftarrow 1$
58:     $track_{new}.state \leftarrow tentative$
59:     $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_{new}\}$
60: return $\mathcal{T}$
```

Chapter 4

Algorithms

In this thesis work, three different implementations that integrates RGBD data with multi-object tracking frameworks were developed and tested to obtain a model capable of providing efficient people 4D pose estimation and tracking.

The first method combines YOLOv8 Segmentation with the SORT algorithm, establishing a baseline approach that focuses on reliable detection and basic tracking capabilities.

Following a progressive approach, the other two methods build upon the previous one to address specific limitations and improve overall performance: the second method enhances the tracking component by replacing SORT with StrongSORT, aiming to improve detections-to-tracks association and reduce ID switches; the third method further refines the system by switching to YOLOv8 Pose detection, to directly estimate orientations through body keypoint analysis.

In this chapter, each of the three methods is described in detail.

4.1 Segmentation-based Detection with SORT

The main challenge in developing the first method was to effectively integrate RGBD data into an existing RGB-based multi-object tracking framework.

Specifically, given the RGBD image and people detections in the form of 2D bounding boxes, it was essential to associate each person with an accurate position in 3D space. This requires selecting a pixel for deprojection into 3D using the depth map (refer to Section 5.2.1).

An initial approach we considered was to use the center of the bounding box, but this method was found to be inaccurate, especially in cases of partial occlusions or movements, where this point can lie outside the person's actual region and correspond to a 3D point not belonging to the person.

To address this and to maximize the likelihood that the extracted 3D location corresponds to the detected individuals, we opted to use the centroid of the segmentation mask (which is a binary image of the same size as the RGB image, with 1s located in correspondence of pixels that belong to the person) as the candidate 2D point for deprojection, verifying that it lies within the mask itself (Figure 4.1).

That is why, for object detection, we decided to use YOLOv8 Segmentation model, which provides both bounding boxes and segmentation masks for each detected person.

4.1.1 Centroid Calculation

The centroid of a shape is the average position of all its points. For a segmentation mask, the centroid coordinates (u_c, v_c) are computed as follows:

$$u_c = \frac{1}{N}\sum_{i=1}^{N} u_i, \qquad v_c = \frac{1}{N}\sum_{i=1}^{N} v_i \tag{4.1}$$

where (u_i, v_i) are the coordinates of the pixels that belong to the mask (i.e., pixels with value 1), and N is their total number.

Centroids can be effectively computed using image moments [27]. For grayscale images with pixel intensities $I(u, v)$, raw image moment M_{ij} of order $(i + j)$ is defined as:

$$M_{ij} = \sum_u \sum_v u^i v^j I(u, v). \tag{4.2}$$

Segmentation masks are binary images, therefore the intensity $I(u, v)$ is either 0 or 1. For binary images the moment of order zero (M_{00}) represents the area of the mask (i.e., the total number of pixels with value 1, N in Equations (4.1)), while the first moments M_{10} and M_{01} are the sum of u and v coordinates of the pixels in the mask, respectively:

$$M_{00} = \sum_u \sum_v I(u, v), \quad M_{10} = \sum_u \sum_v uI(u, v), \quad M_{01} = \sum_u \sum_v vI(u, v) \tag{4.3}$$

The centroid coordinated are then computed as:

$$u_c = \frac{M_{10}}{M_{00}}, \qquad v_c = \frac{M_{01}}{M_{00}} \tag{4.4}$$

Practically, image moments can be computed using OpenCV moments function[1].

[1]Link: `https://docs.opencv.org/4.x/d0/d49/tutorial_moments.html`.

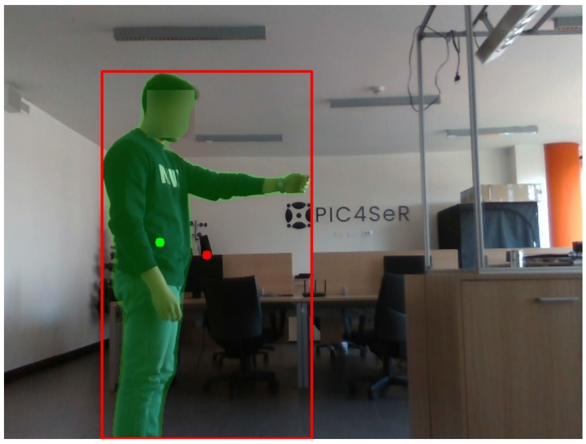


Figure 4.1: Comparison of segmentation mask centroid vs. bounding box center: the segmentation mask (green area) and its centroid (green dot) are shown alongside the center of the bounding box (red dot). The bounding box center lies outside the person's body, and would be deprojected to a 3D point not belonging to the person. The segmentation mask centroid, on the other hand, remains within the mask.

If the centroid lies within the segmentation mask, it is deprojected to obtain a 3D point (x, y, z), and these coordinates are used as the detected 3D position for the person.

In rare cases where the centroid falls outside the mask itself (which can happen as masks are in general non-convex), we explored several approaches, like randomly selecting another point from inside the mask, or scanning a circular region centered on the centroid until a valid point within the mask is found. However, these approaches introduced too much noise into the detections. Given the high frame rate at which this application can run, we found that the best solution was to just skip a detection for one frame if the centroid is outside the mask or if its deprojection does not yield a valid 3D point.

4.1.2 SORT Modifications

To incorporate 3D positional information in each detection, we extended SORT's Kalman filter. Specifically, state vector $\boldsymbol{x}$ and output vector $\boldsymbol{y}$ were augmented, adding the 3 spatial coordinates (x, y, z), and, in the case of state $\boldsymbol{x}$, the corresponding velocities as well. The system is now 13-dimensional:

$$\boldsymbol{x} = [u, v, s, r, x, y, z, \dot{u}, \dot{v}, \dot{s}, \dot{x}, \dot{y}, \dot{z}]^\top, \qquad \boldsymbol{y} = [u, v, s, r, x, y, z]^\top \tag{4.5}$$

This extension required modifications to the Kalman Filter matrices: the state transition matrix $\boldsymbol{F}$ was enlarged to model the motion in 3D space, under a constant velocity assumption; the measurement matrix $\boldsymbol{H}$ was adapted to accommodate the 3D measurements; the process noise covariance $\boldsymbol{Q}$ and measurement noise covariance $\boldsymbol{R}$ matrices were expanded accordingly. To determine the values of the new constant entries in the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices corresponding to the added 3D state variables, we used a trial-and-error approach. This helped achieve a good balance between filtering out noise and maintaining responsiveness in the estimated 3D quantities.

Considering that spatial location provides very meaningful information to help associate detections with tracks over time, we decided to integrate it with the IOU between 2D bounding boxes to improve the association strategy:

$$c = \alpha(1 - IOU) + (1 - \alpha)c_E \tag{4.6}$$

where IOU is the intersection-over-union between predicted and detected bouding boxes and c_E is the Euclidean distance between predicted and detected 3D positions, normalized to the $[0, 1]$ range using min-max normalization:

$$c_E = \frac{(d - d_{min})}{(d_{max} - d_{min})} \tag{4.7}$$

where d is the Euclidean distance between a detection and a track's prediction in 3D space, d_{min} and d_{max} are the minimum and maximum distances in the current set of detection-track pairs. The weighting factor α was set to 0.3 after empirical testing.

In line with the original SORT formulation, where unlikely associations are prevented by using a IOU_{min} threshold, we decided to implement a similar mechanism for the 3D Euclidean distance. In particular, we added an adaptive threshold so that a new detection-track pair is considered for association only if their 3D distance is below a value τ defined as:

$$\tau = \mu_d + 1.6 \cdot \sigma_d \tag{4.8}$$

where μ_d and σ_d are the mean and standard deviation of all detection-prediction distances in the previous frame. This threshold dynamically adapts to account for variations in distance distributions, especially in cases where people have different walking speeds, or when they enter/leave the scene.

The output of the algorithm consists of the updated state of each active track that was matched at the current frame. The orientation of each tracked individual is computed a posteriori as the angle described by the projection of the velocity vector $\boldsymbol{v} = [\dot{x}, \dot{y}, \dot{z}]^\top$ on the horizontal plane xy with respect to the x-axis:

$$\psi = \text{atan2}(v_y, v_x) \tag{4.9}$$

4.2 Segmentation-based Detection with StrongSORT

Although the first method showed good performance for simple scenes, where IOU and 3D Euclidean distance are sufficient to distinguish individuals, as the number of people in the scene and occlusions increased, the number of ID switches became considerable.

To try to address these limitations, in the second approach we replaced the SORT algorithm with StrongSORT, so as to also take advantage of visual features in addition to spatial and motion clues.

4.2.1 StrongSORT Modifications

As in the first method, we had to modify the Kalman filter to incorporate the 3D coordinates. This time, considering that spatial position is more discriminative than 2D IOU, and with the goal of keeping the application as efficient as possible (also considering the presence of the Re-ID network, which impacts the inference time), we decided to remove the variables related to the bounding box from the Kalman filter, replacing them with the (x, y, z) coordinates. As a result, the state vector of the filter becomes 6-dimensional and the measurement space 3-dimensional:

$$\boldsymbol{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^\top, \qquad \boldsymbol{y} = [x, y, z]^\top \tag{4.10}$$

Matrices $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ were redefined following the constant velocity motion model derivation shown in Appendix A.1.1. The process noise covariance matrix $\boldsymbol{Q}$ was constructed according to Equation (A.9), where σ_a represents the standard deviation of the acceleration noise that controls the relaxation of the constant velocity assumption. Similarly, the measurement noise covariance matrix $\boldsymbol{R}$ was defined starting from Equation (A.11), where σ_m represents the uncertainty in the position measurements. To account for the decrease in position accuracy as the depth increases, $\boldsymbol{R}$ was further modified as follows:

$$\tilde{\boldsymbol{R}}_k = (1 + \frac{x}{max\_depth})\boldsymbol{R}_k \tag{4.11}$$

where max_depth is set to 5.0 m (refer to to Section 5.2.1).

Through a trial-and-error approach, we determined that $\sigma_a = 0.5$ m/s^2 and $\sigma_m = 0.5$ m provided the best tracking performance. The value of σ_a is reasonable considering typical pedestrian acceleration values $(0.4 - 1.2$ m/s$^2)$ [28]. Similarly, the value $\sigma_m = 0.5$ m accounts not only for the camera's depth uncertainty but also for the fact that the measured 3D position, obtained by deprojecting a 2D point, is an approximation of the true person's centroid. This approximation can vary significantly ($\pm 0.2 - 0.3$ m) due to changes in pose and orientation.

We kept the Mahalanobis distance as metric for motion information, adjusting the max cost threshold to account for the smaller measurement space (for a 3-dimensional measurement space, the 95% confidence interval of the inverse χ^2 distribution is 7.8147).

Additionally, we replaced the IOU metric used in the second round of assignment (ref. Section 3.3.2) with the 3D Euclidean distance.

To compute the orientations we kept the same approach as in the first method, using the angle described by the velocity vector.

4.3 Pose-based Detection with StrongSORT

While the velocity-based orientation estimation used in the two previous methods provided reasonable results, it showed an intrinsic limitation: the velocity components used in Equation (4.9) are estimated by the Kalman filter and not directly measured, and they exhibit a natural delay in responding to changes in people's motion.

This delay, known as transient lag, is partly due to the constant velocity motion model used, as the filter expects the velocity to remain constant between consecutive frames, and it takes several consistent position measurements to steer the velocity prediction onto a new value.

The delay is further influenced by the fact that the Kalman filter, being a recursive state estimator, needs to balance between measurement noise filtering and state update speed. This balance is determined by the entries of matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. Higher uncertainty in measurements (larger $\boldsymbol{R}$ values) results in smoother trajectories but lower state update speed and lower filter responsiveness. In fact, when measurement uncertainty is high, the filter becomes more conservative in updating its state estimates, treating initial position changes as potential noise. This results in a delayed response in the velocity estimates and, consequently, in the derived orientation angles.

To address this limitation, we developed a direct orientation measurement approach using the body keypoints provided by YOLOv8 Pose model, replacing the segmentation model used in the previous method while keeping the modified version of StrongSORT.

In particular, two methods were implemented and combined to obtain orientation estimates.

4.3.1 Orientation Estimation Methods

Orientation based on 3D shoulder and hip keypoints. The first method uses the 3D position of shoulder and hip keypoints, if available. After deprojecting these keypoints using the depth map, the orientation is computed as the angle of the vector perpendicular to either the shoulder line or hip line:

$$\boldsymbol{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} x_r - x_l \\ y_r - y_l \\ z_r - z_l \end{bmatrix}, \qquad \psi = \operatorname{atan2}(d_x, -d_y) \tag{4.12}$$

where $\boldsymbol{d}$ is the 3D vector between the left (x_l, y_l, z_l) and right (x_r, y_r, z_r) shoulders (or hips).

YOLOv8 Pose can reliably distinguish between left and right shoulder/hip, regardless of the person orientation. This allows to solve the ambiguity between ψ and $\pi + \psi$ rad (Figure 4.2).

The main limitation of this method is that when a person is oriented at roughly $\pm\ \pi/2$ rad with respect to the camera, the 3D position of the farthest shoulder/hip is no longer obtainable from the depth map since it is covered by the other shouder/hip (Figure 4.3). To avoid wrong estimations when the orientation approaches these angles, we discard the estimate if the 3D positions of the two shoulders/hips are too close to each other (distance < 0.1 m).

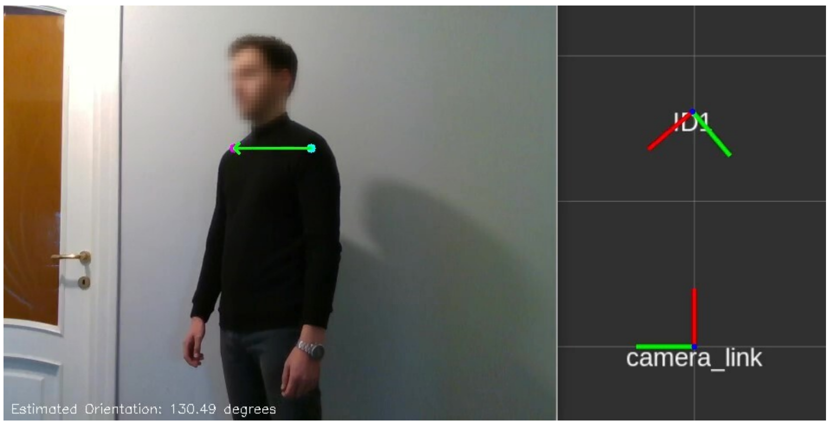
Orientation based on Anthropometric Ratio. The second method exploits the human body proportions. Considering the 2D image plane, we compute the ratio between the horizontal distance between shoulder keypoints (and hip keypoints) and the vertical distance between a shoulder and the corresponding hip keypoints. Given the pixel coordinates (u_{rs}, v_{rs}), (u_{ls}, v_{ls}), (u_{rh}, v_{rh}), (u_{lh}, v_{lh}) of right shoulder, left shoulder, right hip, left hip, respectively, we can define:

$$d_{ss} = u_{rs} - u_{ls}, \qquad d_{hh} = u_{rh} - u_{lh}, \qquad d_{sh} = v_{lh} - v_{ls} \tag{4.13}$$

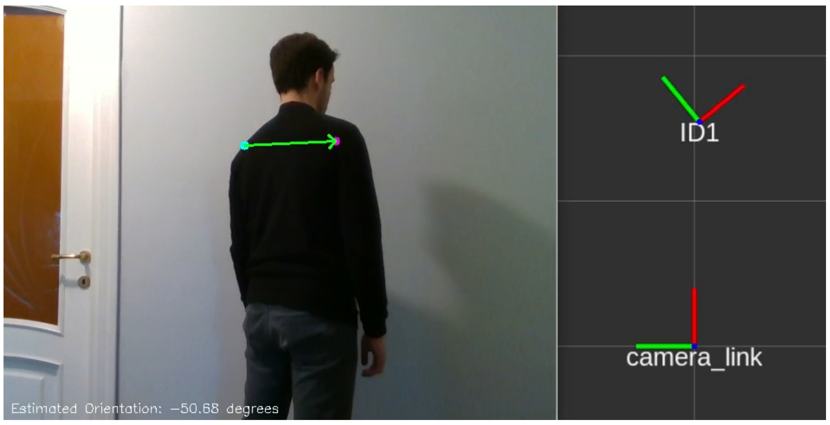
where d_{ss} is the shoulder-to-shoulder horizontal distance, d_{hh} is the hip-to-hip horizontal distance and d_{sh} is the shoulder-to-hip vertical distance. From these distances, we can compute the ratios:

$$r_s = \frac{d_{ss}}{d_{sh}}, \qquad r_h = \frac{d_{hh}}{d_{sh}} \tag{4.14}$$

These ratios reach their maximum when the person is facing directly away from or towards the camera (0 or π rad, respectively), and approach zero when the person



(a)



(b)

Figure 4.2: Example of orientation estimation based on the 3D locations of shoulder keypoints. Image (a) shows an orientation of $\psi = 130°$, while image (b) shows an orientation of $\psi = -50/310°$, equivalent to $130° + 180°$. YOLOv8 Pose distinguishes between left and right keypoints, allowing to differentiate ψ and $180° + \psi$ cases.

is viewed from the side ($\pm\pi/2$ rad). These ratios must be normalized to the $[-1, 1]$ range to be converted to angles using the arccos function:

$$\tilde{r}_s = \max\left(\min\left(\frac{r_s}{r_s^{max}}, 1.0\right), -1.0\right), \quad \tilde{r}_h = \max\left(\min\left(\frac{r_h}{r_h^{max}}, 1.0\right), -1.0\right) \tag{4.15}$$

$$\psi_s = \arccos(\tilde{r}_s), \qquad \psi_h = \arccos(\tilde{r}_h) \tag{4.16}$$

where the values for the maximum ratios were obtained from [29]. The average of

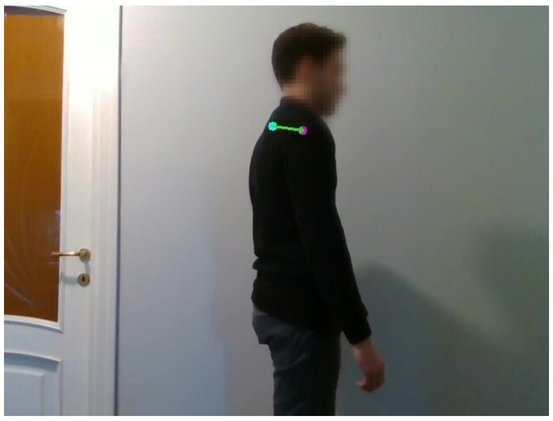


Figure 4.3: Example of a scenario where orientation estimation based on 3D keypoints is not possible. The person is oriented at roughly $-90°$ with respect to the camera, causing one shoulder to obstruct the other. Both keypoints would be deprojected to approximately the same 3D point on the right shoulder.

these two angles is then computed to obtain an unique orientation estimate from this method.

This method has the advantage of being invariant to the person's distance from the camera, but it presents an ambiguity between ψ and $-\psi$ because the two situations are equivalent in terms of 2D shoulder/hip keypoints location (Figures 4.4, 4.5). To try resolve this, we utilized facial keypoints (in particular ears and eyes) to determine the sign of the orientation:

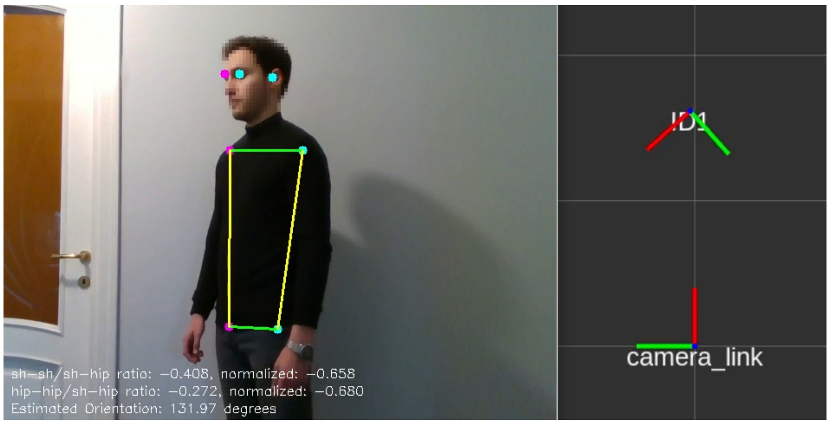
$$\text{sign} = \begin{cases} +1, & \text{if left eye/ear is visible and right ear is not} \\ -1, & \text{if right eye/ear is visible and left ear is not} \\ \text{None}, & \text{other cases} \end{cases} \tag{4.17}$$

4.3.2 Orientation Filtering and Fusion

When both methods provide valid estimates, their orientations are combined using circular averaging to handle the discontinuity at $\pm\pi$ rad:

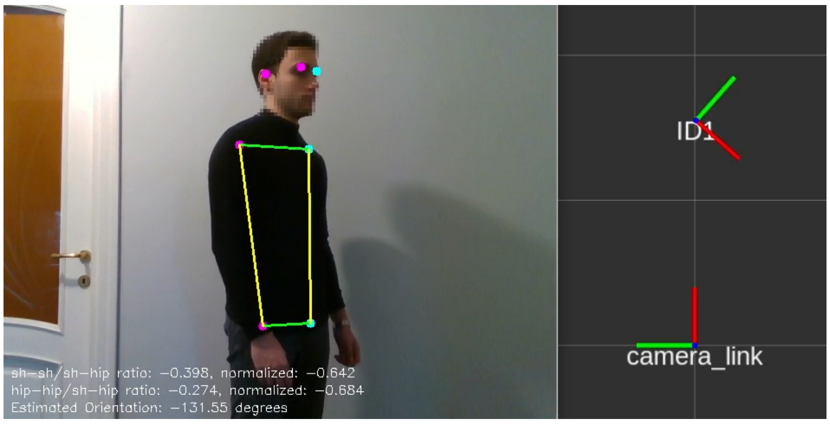
$$\psi = \text{atan2}(\sin\psi_1 + \sin\psi_2, \cos\psi_1 + \cos\psi_2) \tag{4.18}$$

where ψ_1 and ψ_2 are the orientations from the 3D keypoint and anthropometric ratio methods respectively. This approach ensures correct averaging even when the angles cross the $\pm\pi$ rad boundary (e.g., averaging -175° and 175° correctly yields 180° rather than 0°).



sh–sh/sh–hip ratio: −0.408, normalized: −0.658
hip–hip/sh–hip ratio: −0.272, normalized: −0.680
Estimated Orientation: 131.97 degrees

(a)

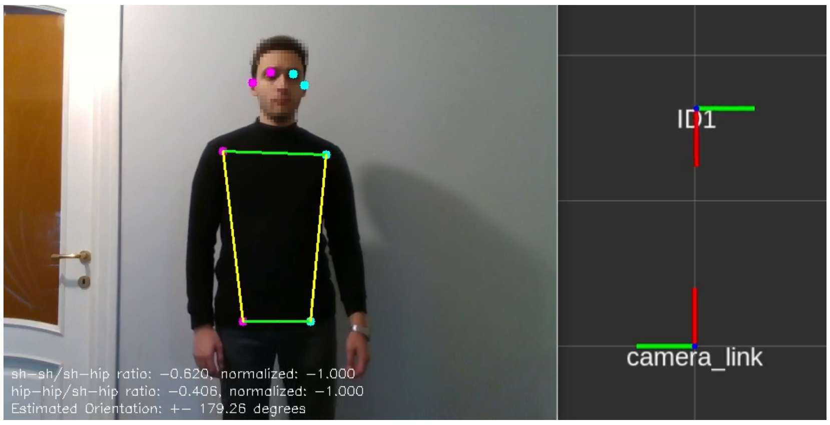


sh–sh/sh–hip ratio: −0.398, normalized: −0.642
hip–hip/sh–hip ratio: −0.274, normalized: −0.684
Estimated Orientation: −131.55 degrees

(b)

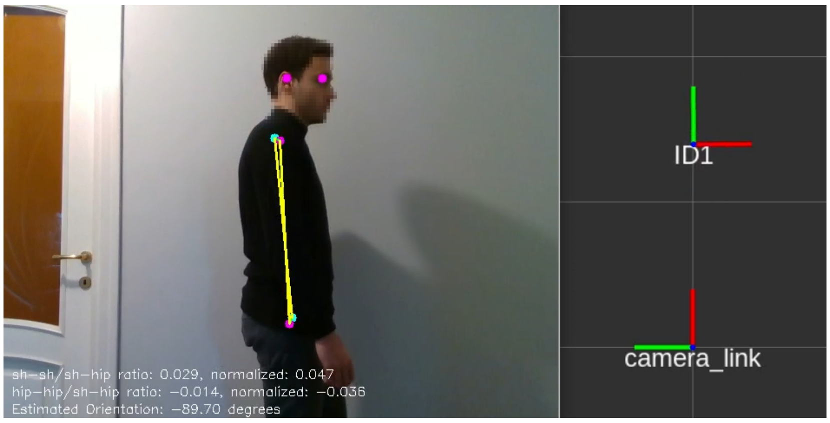
Figure 4.4: Example of orientation estimation based on the anthropometric ratios of shoulder and hip keypoints. Image (a) shows an orientation of $\psi = 130°$, while image (b) shows an orientation of $\psi = -130°$. Thanks to the facial keypoints, the method can differentiate ψ and $-\psi$ cases.

In cases where anthropometric ratio-based orientation method cannot determine a sign for the orientation using the facial keypoints, we adopt the sign from the 3D keypoint method, as it provides unambiguous direction information when valid. To filter the quite high noise that both methods showed, and to account for those scenarios where both cannot produce an orientation estimate (for example when the keypoints are occluded/not visible), we implemented a dedicated Kalman filter for orientation filtering. Each track maintains its own orientation filter with state vector $\boldsymbol{x} = [\psi, \dot{\psi}]^\top$, where ψ is the orientation angle and $\dot{\psi}$ its derivative. The filter uses a constant velocity model.



sh–sh/sh–hip ratio: −0.620, normalized: −1.000
hip–hip/sh–hip ratio: −0.406, normalized: −1.000
Estimated Orientation: +− 179.26 degrees

(a)



sh–sh/sh–hip ratio: 0.029, normalized: 0.047
hip–hip/sh–hip ratio: −0.014, normalized: −0.036
Estimated Orientation: −89.70 degrees

(b)

Figure 4.5: Example of two extreme cases for the orientation estimation based on the anthropometric ratios. Image (a) shows an orientation of $\psi = 180°$. This is one of the two extreme cases (0°,180°) where the ratios are close to the maximum value. In this limit case, the facial keypoints are not sufficient to determine the sign of the angle. Image (b) shows an orientation of $\psi = -90°$. This, instead, is one of the two cases (±90°) where the ratios are close to zero.

To handle angle wrapping and ensure proper tracking across the $\pm\pi$ rad boundary, at each step of the Kalman filter where new angles are computed, we apply the following normalization to map any angle back to the $[-\pi, \pi]$ range:

$$\psi = (\psi + \pi) \bmod (2\pi) - \pi \tag{4.19}$$

Finally, since the YOLOv8 Pose model does not provide a segmentation mask, the 3D position of each person is computed by averaging the 3D locations of all the body keypoints that have, in that frame, a confidence score higher than 60%.

Chapter 5

Experimental Setup

This chapter describes the complete experimental setup used in this thesis work. Starting with the optimization and deployment of the deep learning models, it then details the sensors and hardware components employed and concludes with the software framework used to integrate these elements.

5.1 Models Optimization and Deployment

Three deep network models were used in this thesis work: YOLOv8 Segmentation (YOLOv8s-seg) for people detection and segmentation, YOLOv8 Pose (YOLOv8s-pose) for people keypoints detection, and OSNet x0.25 network as Re-ID model for appearance features extraction. This section describes the optimization process applied to these models to enable efficient real-time inference on the Intel® hardware used to develop and test the algorithms (see Section 5.2.3).

YOLOv8 Segmentation models come pretrained on the COCO2017 dataset [12], which consists of 118,000 images in the training set and 5,000 images in the validation set, covering 80 different object categories including person as well as common objects like cars, bicycles, and animals.

We initially explored fine-tuning it to specialize it for people segmentation. For this purpose, we constructed a combined dataset using images randomly extracted from the COCO2017 training set and containing at least one person, together with the MHP v1.0 dataset [30], which is specifically designed for multi-human parsing and contains precise person segmentation masks. The resulting dataset consisted of 14,780 training images and 3,693 validation images. Despite several attempts to optimize the training hyperparameters, the fine-tuning process showed only marginal improvements in performance (2.5% increase in mask mean average precision mAP50-95, from 51.4% to 53.8%). These results indicated that the original model, pretrained on COCO2017's full dataset, was already well-optimized for people segmentation.

The other two models were used with their original pretrained weights: YOLOv8 Pose come pretrained on COCO Keypoint 2017, which has the same structure as COCO2017, but with images annotated with people keypoints, making it specifically suited for human pose estimation. Similarly, the OSNet x0.25 model comes pretrained on Market1501 dataset [31], a person re-identification dataset containing over 32,000 annotated bounding boxes of 1,501 identities captured from 6 different cameras.

Given these considerations, we focused our efforts on optimizing the inference speed of the models through OpenVINO deployment.

5.1.1 OpenVINO Export

OpenVINO™[1] is an open-source toolkit developed by Intel® for optimizing and deploying deep learning models. It is primarily intended for achieving high performance inference on Intel® hardware, but it also supports ARM/ARM64 processors.

Starting with a model from one of the multiple supported frameworks (e.g., PyTorch, TensorFlow, ONNX), the optimization process involves two main steps:

1. Model conversion. Through the Model Optimizer (MO) tool, the original model is converted into OpenVINO Intermediate Representation (IR) format, which consists of two files: *.xml* file, which describes the model structure in terms of layers and connections, and a binary file (*.bin*), which contains the model weights. This conversion alone drastically improves performance and allows for further optimizations.

2. Post-training optimization. Through the Neural Network Compression Framework (NNCF), which is a set of compression algorithms, the converted model can be further optimized. Specifically, it allows for 8-bit Post-training Quantization, which lowers model weight and activation precisions from FP32 (32-bit floating point precision) to INT8 (8-bit integer precision). This leads to a significant improvement in inference speed, as the footprint is just a quarter of the original full-precision model.

The (basic) quantization process (Figure 5.1) requires a calibration dataset, i.e., a small, representative sample of the input data the model is intended to process. This dataset is used to determine the scaling factor for each layer's weights and activations for the conversion to lower precision, minimizing the accuracy loss.

[1]OpenVINO™ Toolkit: `https://github.com/openvinotoolkit/openvino`.

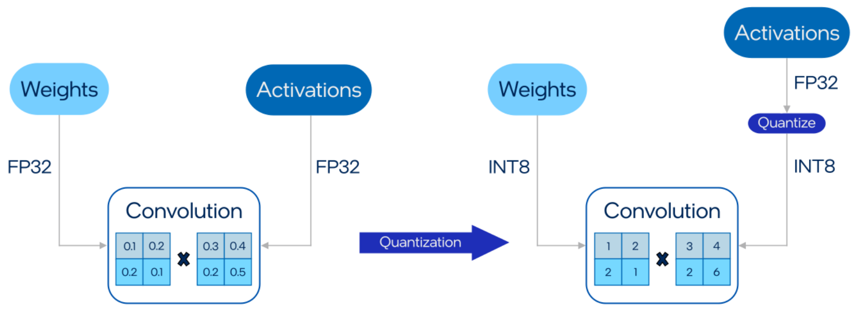


Figure 5.1: OpenVINO™ INT8 post-training quantization. The quantization process maps the continuous range of floating-point values to a discrete set of integer values.

NNCF offers another quantization method, called Accuracy-aware Quantization. This method builds on the basic one to ensure that the accuracy of the resulting model does not drop below a user-defined threshold.

In addition to the calibration dataset, it requires a validation dataset and a validation function to compute the accuracy metric, and it works as follows: the original model is first quantized with the basic quantization algorithm; its accuracy is compared to the one of the original model on the validation set; if the accuracy level is not satisfactory, an iterative approach is used to rank network's layers in order of impact on the accuracy, and top scored layers are reverted back to FP32 until the accuracy meets the specified threshold (Figure 5.2). This adaptive approach ensures that the performance benefits of INT8 quantization are achieved while maintaining the model's accuracy within acceptable bounds.

In our implementation, all deep learning models were optimized using INT8 quantization with accuracy control.

Regarding YOLOv8 models, we used a random subset of 1000 images from COCO2017 and COCO Keypoints 2017 as calibration dataset for YOLOv8s-seg and YOLOv8s-pose models, respectively. As validation datasets for the accuracy control, we used the full validation datasets from the above datasets. The validation functions were obtained from the SegmentationValidator and PoseValidator classes provided by Ultralytics API, and the accuracy drop threshold was set to 1%.

For the OSNet x0.25 Re-ID model, as calibration dataset we used 1000 randomly selected images from the Market-1501 training set. The validation was performed on the complete Market-1501 query and gallery sets using the *evaluate_rank* function from Torchreid [33], which evaluates the CMC rank metric (see Section 2.4), with the accuracy drop threshold also set to 1%.

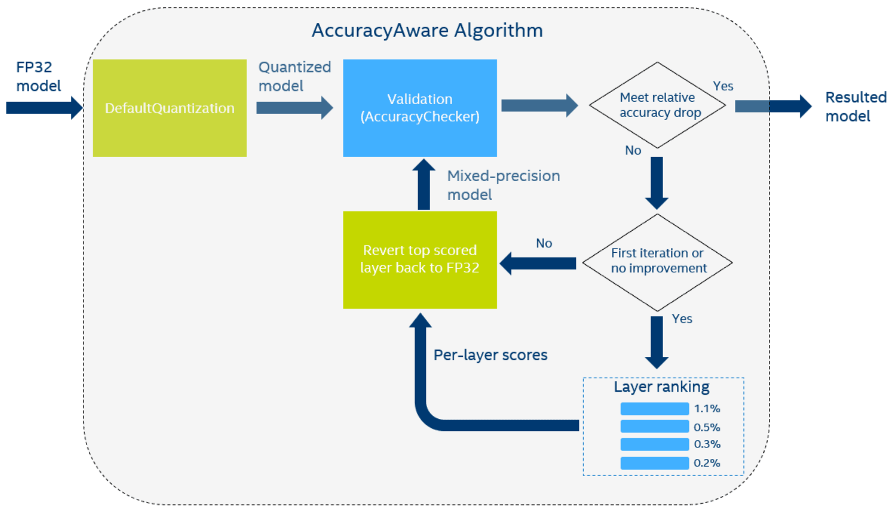


Figure 5.2: Flow diagram of NNCF's quantization with accuracy control, showing the iterative approach to maintain model accuracy while quantizing to INT8 precision [32].

All models were successfully quantized to INT8 precision while maintaining accuracy within the specified threshold, and their inference was performed using OpenVINO Runtime. The performance evaluation of the optimized models in terms of inference speed is presented in Section 6.3.3.

5.2 Hardware and Sensors

This section describes the hardware and sensor components that were used in the thesis work. Specifically, it details the RGBD camera used to obtain the image stream processed by the algorithms, the motion capture system needed to collect ground truth data for the dataset, and the computing hardware where algorithms were executed.

5.2.1 RGBD Camera

RGBD cameras are cameras that capture the depth information of a scene along with the traditional color (RGB) image, providing both a visual image and a spatial map of the environment. They represent a key sensing technology in modern robotics and computer vision applications.

Depth information, meaning the distance between the camera lens and the objects in the environment, is obtained through a depth map created by a 3D depth sensor such as a stereo sensor or time of flight sensor.

Depth maps are 2D greyscale images of the same size of the RGB images they are associated with. The grey value of each pixel of this map indicates the distance to the camera of the point in space represented by the associated RGB pixel.

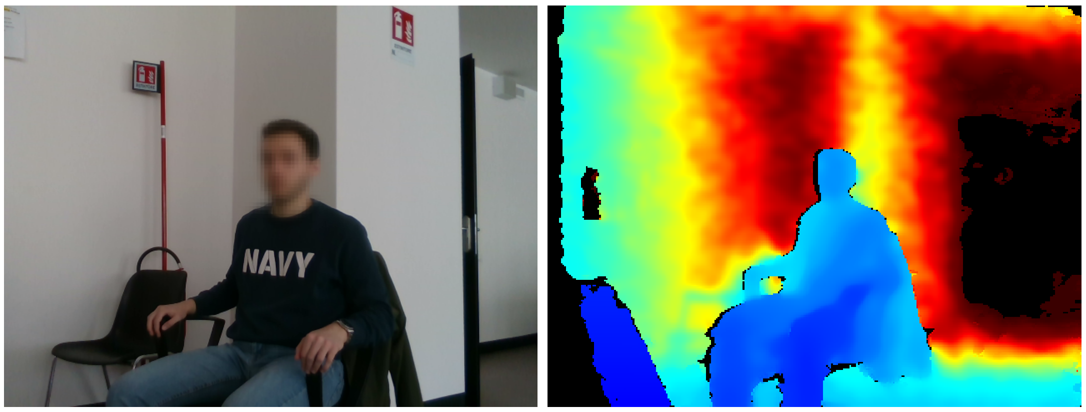


Figure 5.3: RGB image (left) and its corresponding depth map (right).

Considering that in this thesis work we used an Intel® RealSense™ D435i, which is a stereo depth camera, this depth sensor technology is analyzed.

Stereo depth. Stereo depth cameras are based on the same principle as human binocular vision: two camera sensors, the left and right imagers, spaced a small distance apart, capture the scene and send an image to the depth imaging processor. This one compares the two images and calculates depth by estimating disparities between corresponding points in the left and right images.

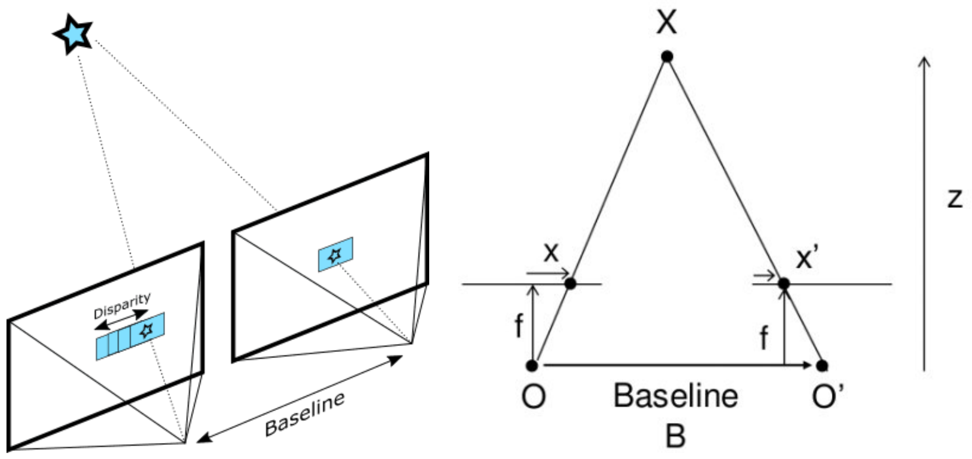


Figure 5.4: Working principle of stereo depth cameras [34],[35].

The disparity refers to the difference in position of a feature when observed from the two different viewpoints. Points closer to the camera will appear to move significantly from left to right image (big disparity), where point in the far distance would appear to move very little (small disparity).

Since the distance between the two sensors, called baseline b, is known, as well as the focal length of the cameras f, the image processor can extract the depth Z from point disparity d with the following equation:

$$disparity\ d = x - x', \qquad\qquad Z = \frac{f}{d}b \tag{5.1}$$

This method is called triangulation.

The quality of the depth map obtained through stereo vision depends primarily on the density of visually distinguishable points/features for the algorithm to match. Indeed, one of the main challenges in stereo vision is finding corresponding points in the two images.

This can become difficult in areas with uniform texture or color, as well as in low lighting conditions. Any source of texture — natural or artificial — can significantly improve the accuracy [34],[36].

That is why stereo cameras like the Intel® RealSense™ D435i integrate an infrared projector that casts a pseudo-random pattern of dots onto the scene to add details outside the visible spectrum. This is known as active stereo vision. The projected pattern creates artificial texture that helps the stereo matching algorithm (Figure 5.5).

Mathematical model. A fundamental operation that is at the base of all algorithms developed and tested in this thesis work is the deprojection of pixels to 3D points from the RGBD images captured by the camera.

To understand how this works, we first need to understand how 3D points in the world coordinate are mapped into pixels in the image plane. If we assume no distortions from the camera lens, we can consider the ideal pinhole camera model, where the camera aperture is described as a point. According to this model, the projection of a 3D point $P = [X, Y, Z]$ results in a 2D point $p = [u, v]$ can be described using the intrinsic camera parameters [37]:

- $F = (f_x, f_y)$: focal length in multiple of pixel size.
- $P = (c_x, c_y)$: Principal point (the image center), as a pixel offset from the left-top corner of the image.

The projection equations are:

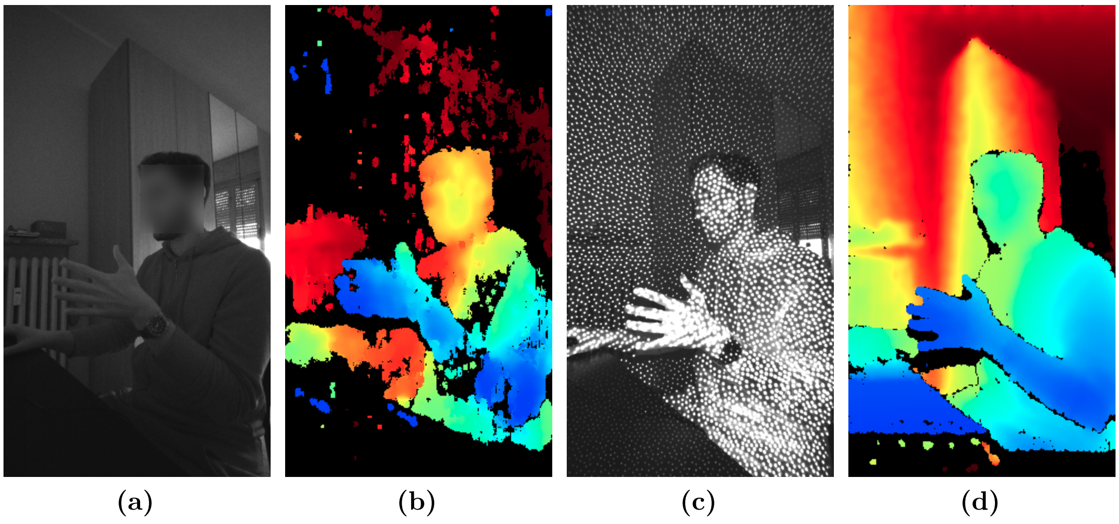


Figure 5.5: Comparison between passive stereo vision (a),(c) and active stereo vision (b),(d) in low lighting conditions. Images (a) and (c) show the infrared images captured by one infrared imager with and without the infrared projector active, respectively. Images (b) and (d) are the resulting depth maps.

$$u = f_x \frac{X}{Z} + c_x, \qquad\qquad v = f_y \frac{Y}{Z} + c_y \tag{5.2}$$

In practice, real cameras introduce both radial and slight tangential distortions that need to be accounted for. In the case of the Intel® RealSense™ D435i, the lens distortion model and its coefficients can be obtained from the RealSense API (RealSense SDK 2.0).

The process of projecting 3D point to pixel is not reversible (as in standard RGB images), unless the pixel's depth Z is known, which is provided by the depth map in RGBD cameras. In this case, given a pixel coordinate $p = [u, v]$ and its depth Z, we can reconstruct the corresponding 3D coordinates by using the deprojection equations:

$$X = (u - c_x)\frac{Z}{f_x}, \qquad\qquad Y = (v - c_y)\frac{Z}{f_y} \tag{5.3}$$

These equations are still under the assumption of pinhole camera model with no distortions.

Intel® RealSense™ D435i. To test in real-time all the algorithms and to record the dataset we used an Intel® RealSense™ Depth Camera D435i. This RGBD

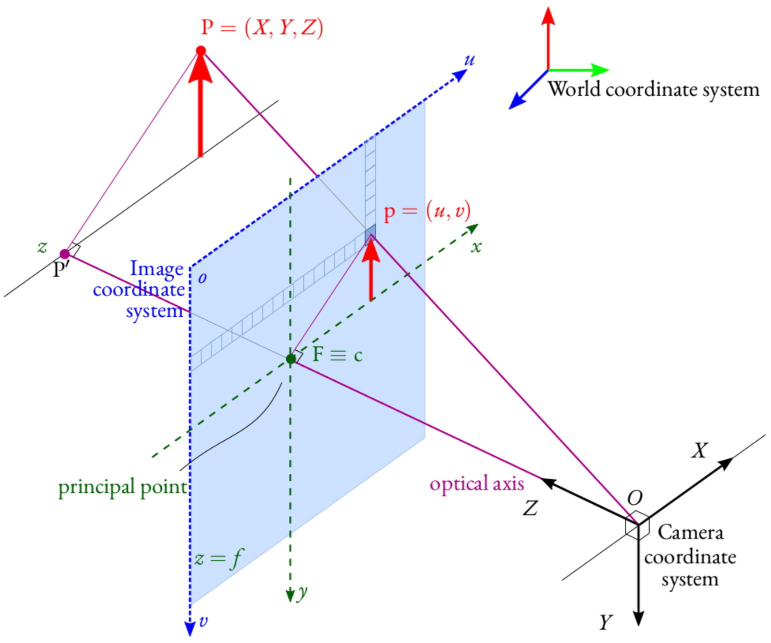


Figure 5.6: Pinhole camera model [38].

camera combines stereo depth vision with regular RGB camera in a compact form factor, making it suitable for applications such as robotic navigation and object recognition.

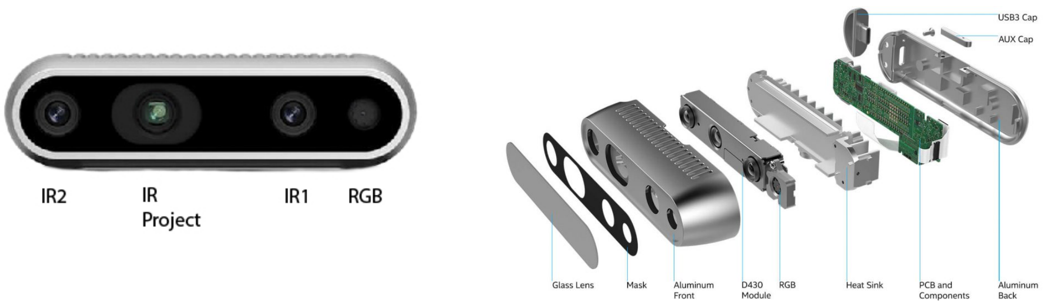


Figure 5.7: Intel® RealSense™ D435i on the left, and its internal hardware components on the right. **IR1**, **IR2**: left and right infrared imagers, **IR projector**: infrared projector, **RGB**: RGB camera.

The D435i comes with a dedicated vision processor, the Intel® RealSense™ Vision D4 processor, whose primary function is to perform depth stereo vision processing. The camera system consists of (see Figure 5.7 for reference):

- Two infrared cameras (right and left imager) with a global shutter;
- An infrared projector;
- A color camera sensor;
- An inertial measurement unit (IMU).

The two infrared cameras, spaced 50 mm apart (baseline b), are used by the stereo depth system to create the depth map. The infrared projector, as described in the previous section, is used to create artificial textures to improve stereo matching. The images from the stereo depth system are already rectified and need no use of algorithms for the rectification.

The specifications of the D435i are the following [39]:

Specification	**Details**
Depth Field of View (FOV)	87° × 58° × 95° (Horizontal × Vertical × Diagonal)
Color Camera FOV	69° × 42° × 77° (H × V × D)
Depth Output Resolution	Up to 1280 × 720 pixels
RGB Output Resolution	Up to 1920 × 1080 pixels
Frame Rate	Up to 90 fps for depth, up to 60 fps for RGB
Ideal Range	0.2 m to over 3 m (varies with lighting conditions)
IMU	6-axis sensor (3-axis accelerometer and 3-axis gyroscope)

Table 5.1: Specifications of the Intel® RealSense™ D435i.

For our application, we used the camera with a resolution of 640×480 at 30 fps for both depth and RGB streaming and we set the maximum depth distance to 5.0 m. This configuration provided a good balance between processing requirements and the spatial resolution needed for accurate tracking.

A ROS2 Wrapper[2] is provided within the Intel® RealSense™ SDK (Software Development Kit) 2.0 cross-platform library, which we used to integrate the camera into the ROS2-based framework where the applications were developed and tested (more on this in section 5.3).

[2]GitHub link: `https://github.com/IntelRealSense/realsense-ros`.

This `realsense2_camera` package allows launching a ROS2 node that reads data streams and calibration parameters from the camera and publishes them on various topics. In particular, the main topics published are:

- `/camera/color/image_raw`: RGB images;
- `camera/depth/image_rect_raw`: Depth images;
- `/camera/aligned_depth_to_color/image_raw`: Depth images aligned to the RGB frame;
- `/camera/color/camera_info`: RGB camera intrinsic parameters;
- `/camera/depth/camera_info`: Depth camera intrinsic parameters.

The camera parameters can be configured through ROS2 node parameters. Key parameters to be set to true are `enable_color` and `enable_depth`, to enable the respective sensors, `align_depth.enable`, to ensure that RGB and depth data are aligned, which is crucial for accurate 3D position estimation, and `enable_sync`, to ensure that messages published on different topics are synchronized in time.

Furthermore, if also `enable_rgbd` is set to true, the node publishes on a unique topic, `/camera/rgbd`, where each message contains all the synchronized messages of the aforementioned topics.

One important aspect to consider when working with the data provided by the camera is the different coordinate frames involved. There are two main families of frames (Figure 5.8): optical frames, which follow the conventional computer vision coordinate system of (X: right, Y: down, Z: forward) if seen from behind the camera, and frames that instead follow the ROS2 coordinate system convention of (X: forward, Y: left, Z: up).

All data published by the ROS2 node in the different topics is optical data taken directly from the camera sensors. The `realsense2_camera` node also publishes the correct transformations between these frames on the `/tf` and `/tf_static` topics.

The following coordinate frames were considered in this work:

- ROS2 and optical frames of the RGB camera: `camera_color_frame` and `camera_color_optical_frame`, with origin on the RGB sensor;
- ROS2 and optical frames of the depth camera: `camera_depth_frame` and `camera_depth_optical_frame`, with origin on the left IR imager;
- Camera link frame: `camera_link`, which represents the camera base, and is placed on the left IR imager (coincides with `camera_depth_frame`).

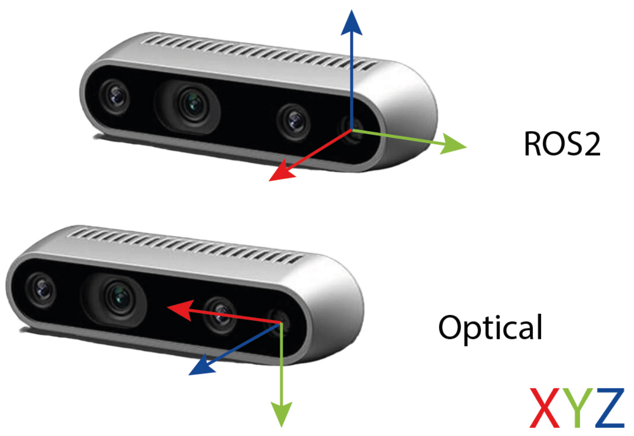


Figure 5.8: Optical and ROS2 coordinate frames.

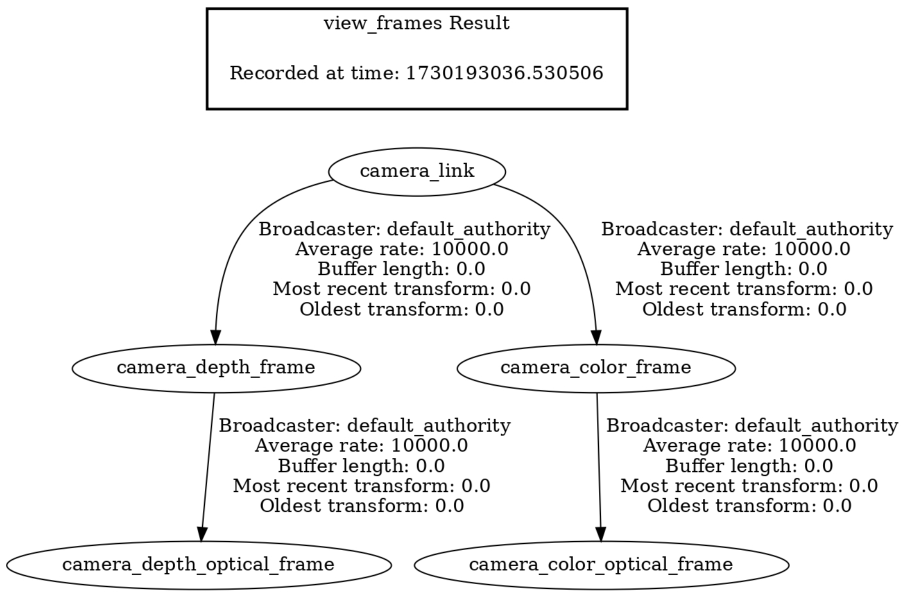

view_frames Result
Recorded at time: 1730193036.530506
camera_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_depth_frame
camera_color_frame
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_depth_optical_frame
camera_color_optical_frame

Figure 5.9: Complete transformation chain between camera frames, generated using ROS2's `tf2_tools`.

The transformation chain between these frames is shown in Figure 5.9.

When working with aligned depth images (`align_depth.enable` set to true), the depth points are already transformed into the RGB camera's optical frame, simplifying the computation of 3D positions of pixels in the RGB image. This means that for any pixel (u,v) in the RGB image, its corresponding depth value from the aligned depth image can be directly used with the RGB camera's intrinsic parameters for 3D position computation, without requiring additional frame transformations.

5.2.2 Motion Capture System

To record ground truth data to evaluate and compare the different algorithms, we used the Vicon® Motion Capture system installed in the PIC4SeR center laboratory.

This system consists of 10 high-speed infrared cameras mounted around the perimeter of the room. These cameras emit infrared light that reflects off passive markers coated with reflective material, making them recognizable in the captured images.

Before collecting data, a calibration process is needed. An object with markers attached to it in a known configuration is used to make the system determine the relative position and orientation of the cameras. Once calibrated, the system software can triangulate the 3D location of new markers by combining the 2D views from the different cameras. In addition, the software can also automatically label and link markers to form a representation of a solid object. This allows new objects to be registered in the system, and they will be recognized as such as soon as they enter the tracking area.

To be able to make the system identify a person as an object and track its 4D pose, we designed four custom U-shaped cardboard frames, each with six reflective markers, as shown in Figure 5.10. These frames were designed to be worn as collars, and to ensure unique identification of each person, the markers on each frame were placed in asymmetric positions, with each frame having a distinct marker configuration.

This design was intended to meet several needs: first, the need to have an object that could be worn by people while still allowing them to move naturally. Second, we needed an object that would allow at least 3 markers to be seen in any possible orientation so that the system would not lose tracking. Third, we were looking for a solution that is not too bulky so as not to affect the recognition of people in YOLOv8 and the Re-ID network. Additionally, the object must be rigid enough to keep the relative position of the markers constant, otherwise the system would no longer be able to recognize it.

To collect the ground truth data within a ROS2 framework, we used a customization made by the PIC4SeR research group of the project mocap4ros2_vicon[3],

[3]GitHub link: `https://github.com/MOCAP4ROS2-Project/mocap4ros2_vicon`.

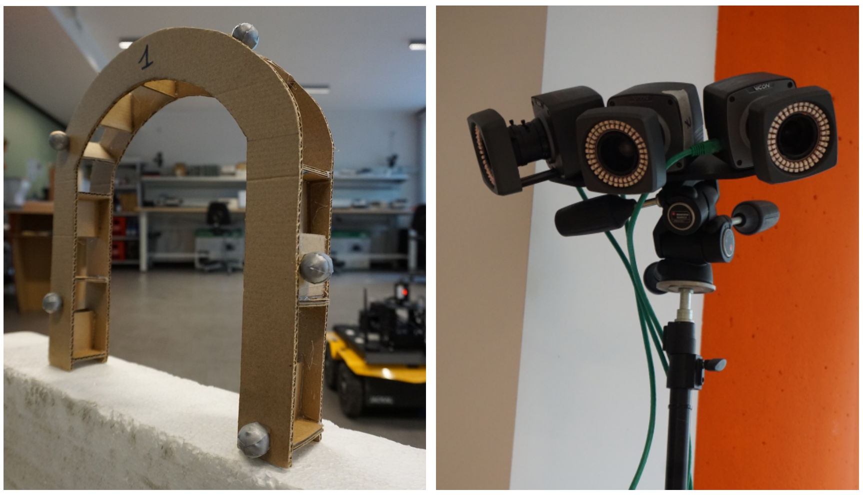


Figure 5.10: One of the U-shaped cardboard frames on the left; three of the ten high-speed infrared cameras of the Vicon® system on the right.

which provides the ROS2 integration with Vicon cameras, and is part of the MOCAP4ROS2 project [40].

The motion capture data was published on the `/vicon/people` topic using messages of type `People` from the `people_msgs` ROS2 package. Each People message contains a list of `Person` messages, where each `Person` represents one of the U-shaped frames present in the tracking area. The frames are univocally identified with names ranging from "U_1" to "U_4". The `Person` message includes the position, orientation, and velocity data of the corresponding frame, allowing for complete 4D tracking of each person wearing it.

5.2.3 Hardware

The computer hardware used in this thesis work is the following:

- **Training workstation**. The YOLOv8s-seg model was trained on a workstation equipped with an Intel Core i7-9700K CPU (8 cores, clock speed 3.6 GHz, up to 4.9 GHz) and two NVIDIA GeForce RTX 2080 SUPER, each with 8 GiB GDDR6 memory. The system has 62 GiB of RAM.

- **Development and Testing Platform**. Algorithms were developed and tested on an Intel NUC 11 Mini PC with 11th Gen Intel Core i5-1135G7 CPU (4 cores/8 threads, base clock 2.40 GHz, up to 4.20 GHz). The system has 8 GiB of RAM (7.4 GiB usable). This system was also used to acquire data from the Intel® RealSense™ D435i depth camera and the Vicon® Motion Capture system to record the dataset.

5.3 ROS2 Framework

Robot Operating System 2 (ROS2, [41]) is a popular open-source middleware framework for building robot applications. Building on the original ROS [42], provides higher efficiency, reliability, security and real-time processing support. It runs on Unix-based platforms, and provides services commonly associated with an operating system, including hardware abstraction, low-level device control, message passing between processes and package management. The core of ROS2 is its runtime "graph", a peer-to-peer network of processes, called nodes, that are coupled using the ROS2 communication infrastructure.

One of the main differences with respect to ROS is the usage of DDS (Data Distribution Service) as the default middleware for communication between nodes. This middleware is suitable for real-time and reliable communication and follows the publish/subscribe paradigm [41].

The decision to develop this thesis work in a ROS2 framework came naturally considering our purpose of developing efficient perception algorithms for social robot navigation, and also considering the already supported integration of the sensors used (RGBD camera and motion capture system) with ROS2.

In this project, we chose ROS2 Humble as distribution, running on Ubuntu 20.04. We ran the entire ROS2 system within a Dev container to simplify dependencies management and making the system portable.

5.3.1 Nodes Structure

Following the modular approach typical of the ROS2 environment, we structured the system on multiple nodes.

One “base” node, `realsense_subscriber_node`, was first developed to receive and process the data stream from the Intel® RealSense™ RGBD camera.

In particular, it synchronizes, through ROS2 message filters, the messages from the different camera topics (Figure 5.9) to guarantee time consistency between RGB image, depth image, and also camera parameters. This is crucial to ensure that the overlaying detection and tracking algorithm works at each iteration with data produced at the same time instant. This node also processes the messages containing the RGB and depth images to convert them in the appropriate format, and also implements the deproject function (Equation (5.3)) to map a 2D pixel to a 3D position.

The detection and tracking algorithms (`rs_yolo_sort`, `rs_yolo_strongsort` and `rs_yolopose_strongsort`) are built as child nodes of the previous. They inherit all its methods and attributes, and they add the detection and tracking phase. They publish the results of the tracking on a topic called `/tracked_people` as messages of type `People`, the same used in the `/vicon/people` topic. Each

`People` message contains a list of `Person` messages, where each `Person` message represents one person tracked by the algorithm. Another node, called `to_csv_node`, is used to export the ground truth data from the `/vicon/people` and the `/tracked_people` topics to a csv file for results analysis.

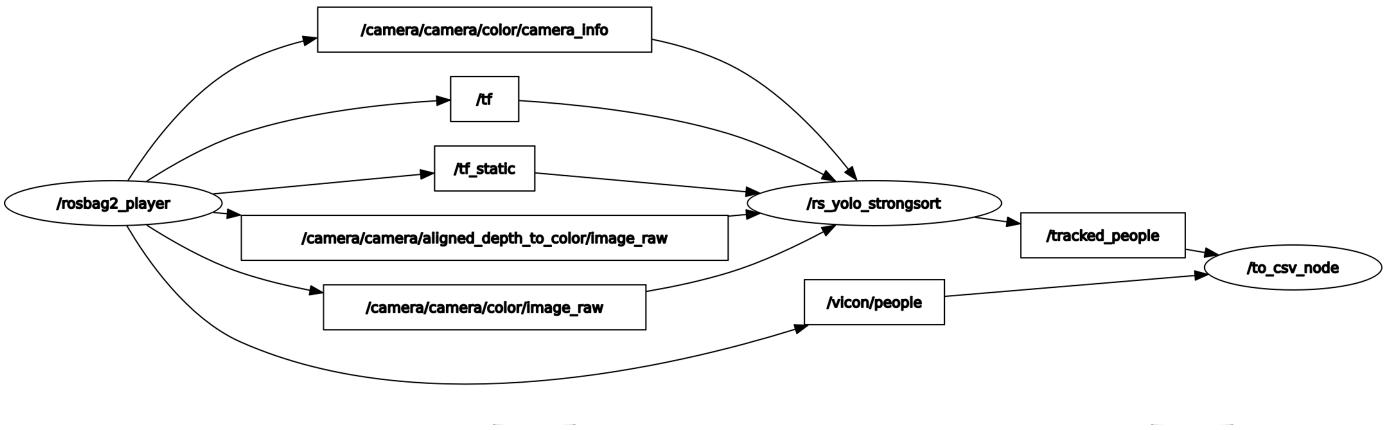

/camera/camera/color/camera_info
/tf
/tf_static
/rosbag2_player
/rs_yolo_strongsort
/camera/camera/aligned_depth_to_color/image_raw
/tracked_people
/to_csv_node
/camera/camera/color/image_raw
/vicon/people

Figure 5.11: ROS2 Computation graph of one detection and tracking algorithm when run on a recorded bag. Topics are shown in boxes, while nodes are shown in ovals.

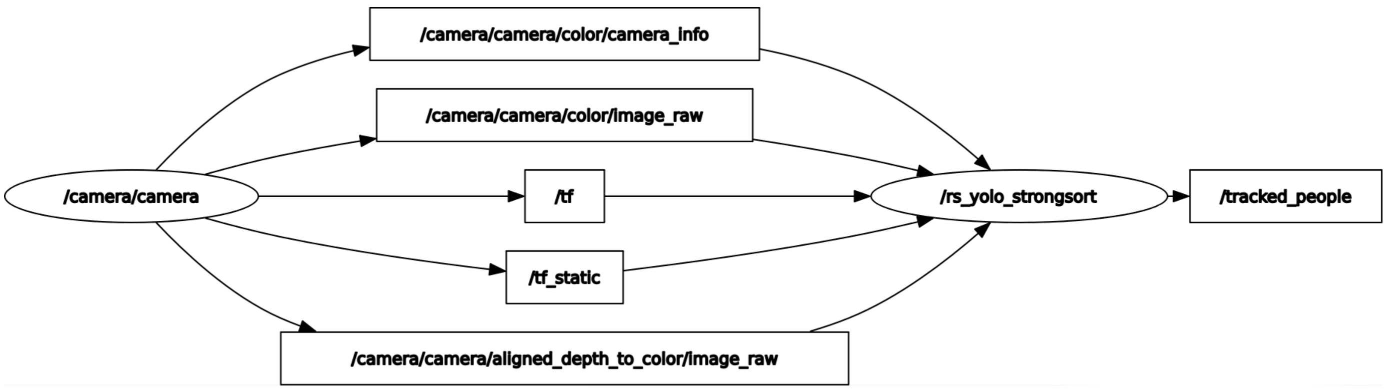

/camera/camera/color/camera_info
/camera/camera/color/image_raw
/camera/camera
/tf
/rs_yolo_strongsort
/tracked_people
/tf_static
/camera/camera/aligned_depth_to_color/image_raw

Figure 5.12: ROS2 Computation graph of one detection and tracking algorithm when running in real-time using data from the camera. Topics are shown in boxes, while nodes are shown in ovals.

Figure 5.11 shows the computation graph (rqt_graph) of one of the algorithms (YOLOv8 Segmentation with StrongSORT) when run on one of the bags from the recorded dataset (hence the presence of the `rosbag2_player` node and the `/vicon/people` topic). Figure 5.12 shows instead the graph of the same algorithm when running in real time with the data provided by the camera.

Chapter 6

Experimental Testing

This chapter presents the experimental testing performed to evaluate the three developed methods and determine if they are suitable for efficient people 4D pose estimation and tracking. The following sections describe the evaluation metrics used, the recorded dataset on which the algorithms were tested and compared, and finally present the comparative results.

6.1 Evaluation Metrics

To quantitatively evaluate the performance of the algorithms, a set of metrics was chosen to measure both the tracking quality and the pose estimation accuracy. We evaluate tracking consistency through normalized ID switches, while pose estimation accuracy is assessed using two primary metrics: root mean square error (RMSE) and tracking success rate, with an additional circular correlation coefficient specifically for orientation estimation to analyze temporal alignment. Finally, to verify real-time capabilities of each method, the computational performance is measured in terms of frames per second (FPS) across the recorded dataset.

6.1.1 Tracking Consistency

To evaluate tracking consistency, we measured the number of ID switches. An ID switch occurs when the tracking algorithm incorrectly changes the identity assigned to a tracked person. This can happen for example when two people cross paths and their identities are swapped, or when tracking is lost and then reinitialized with a new ID.

Within our testing framework, an ID switch can be detected by comparing the IDs assigned by the algorithm with the IDs that univocally identify each custom-made U-shaped cardboard frame (see Section 5.3) in the ground truth data. Since each frame in the motion capture system maintains a consistent ID throughout the recording, any change in the association between tracking IDs and ground truth IDs represents an ID switch.

The number of ID switches was normalized by the duration of each bag to obtain a comparable metric:

$$\text{Normalized ID Switches} = \frac{\text{Total ID Switches}}{\text{Bag Duration (minutes)}} \tag{6.1}$$

6.1.2 Pose Estimation Accuracy

To evaluate the estimated people 4D pose, we used two metrics: root mean square error (RMSE) and the tracking success rate. For orientation estimation specifically, we also analyze the circular correlation coefficient to better evaluate temporal alignment between estimated and ground truth orientations.

Root Mean Square Error (RMSE). RMSE measures the absolute accuracy of the estimates. Considering that in social robot navigation, large deviations in position, velocity and orientation estimates can be more critical than smaller, constant errors, the RMSE was preferred over the mean absolute error (MAE) as it penalizes larger errors more heavily thanks to the square operation.

For position and velocity estimation, RMSE is computed for the x and y components as follows:

$$\text{RMSE}_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2}, \qquad \text{RMSE}_y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{6.2}$$

$$\text{RMSE}_{v_x} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{v}_{x,i} - v_{x,i})^2}, \qquad \text{RMSE}_{v_y} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{v}_{y,i} - v_{y,i})^2} \tag{6.3}$$

where N is the number of frames, $(\hat{x}_i, \hat{y}_i)$ and $(\hat{v}_{x,i}, \hat{v}_{y,i})$ are, respectively, the estimated x and y coordinates and velocities at frame i, and (x_i, y_i) and $(v_{x,i}, v_{y,i})$ the corresponding ground truth variables. The vertical components z and v_z were not considered in the evaluation as people tracking for social robotics applications primarily operates on the ground plane.

Regarding orientation estimation, to compute the RMSE we must consider the cyclic nature of angles:

$$\text{RMSE}_\psi = \sqrt{\frac{1}{N}\sum(\min(|\hat{\psi} - \psi|,\ 2\pi - |\hat{\psi} - \psi|))^2} \tag{6.4}$$

Circular Correlation Coefficient. To evaluate orientation estimation, we also computed the correlation coefficient. Considering once again the circular nature of angles, we used Mardia's circular correlation coefficient defined as [43]:

$$r_\psi = \frac{\sum_{i=1}^{N}(\sin(\psi_i)\sin(\hat{\psi}_i) + \cos(\psi_i)\cos(\hat{\psi}_i))}{\sqrt{\sum_{i=1}^{N}(\sin^2(\psi_i) + \cos^2(\psi_i))\sum_{i=1}^{N}(\sin^2(\hat{\psi}_i) + \cos^2(\hat{\psi}_i))}} \tag{6.5}$$

where ψ_i and $\hat{\psi}_i$ represent the ground truth and estimated orientations respectively.

Tracking Success Rates. The success rate measures the percentage of frames where the absolute error between the estimated quantity and its corresponding ground truth is within a predefined threshold. It provides an intuitive measure for how often the system performs within acceptable bounds for social navigation applications.

For each component (position, velocity, and orientation), we define separate success rates to better evaluate the performance of different aspects of the tracking system. The position success rate measures the percentage of frames where the Euclidean distance between estimated and ground truth positions falls within a threshold τ_{pos}. Similarly, the velocity success rate quantifies how often the magnitude of velocity estimation error is below a threshold τ_{vel}. For orientation, considering the $\pm\pi$ wrapping, the success rate represents the percentage of frames where the absolute angular difference between estimated and ground truth orientations is less than a threshold τ_ψ. Results are presented for different threshold values (τ_{pos}, τ_{vel}, and τ_ψ) to provide a comprehensive view of system performance at various accuracy requirements.

6.1.3 Computational Performance

The computational performance was evaluated by measuring the average processing time per frame across the recorded bags in terms of frame per second (FPS):

$$\text{FPS} = \frac{1000}{\text{Average processing time (ms/frame)}} \tag{6.6}$$

The average processing time per frame is measured on the entire pipeline, starting from the detection performed by YOLOv8 models, the tracking phase performed by the tracking algorithm and finally the post-processing phase to publish the estimated people pose on the ROS2 topic (see Section 5.3). The measurements were performed on the hardware setup described in Section 5.2.3.

6.2 Dataset

To test the algorithms, we recorded a dataset consisting of seven ROS2 bag files. Each bag includes the following synchronized data (see Sections 5.2.1 and 5.3):

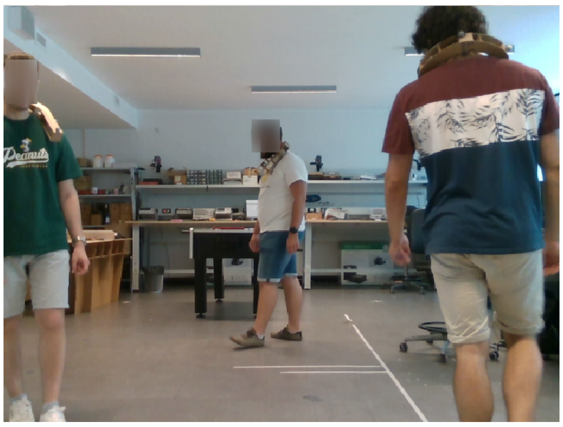
- Ground truth data $(\hat{x}, \hat{y}, \hat{v}_x, \hat{v}_y, \hat{\psi})$ from the motion capture system.
- RGBD data from the Intel® RealSense™ D435i.
- Coordinate frame transformations.

The recorded bags capture different levels of tracking complexity in terms of number of people in the scene and occlusions. Specifically:

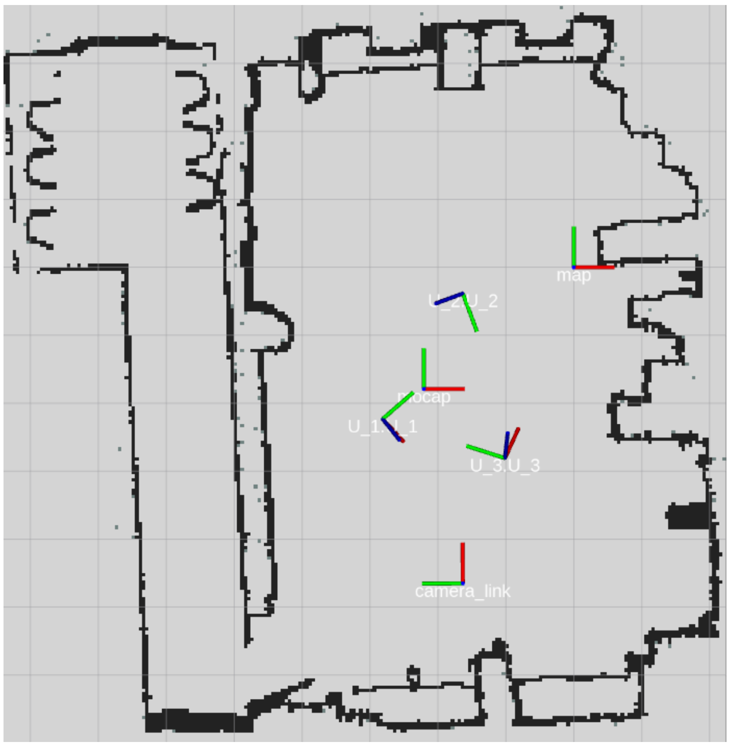
- 2 bags feature scenes with 2 people.
- 3 bags feature scenes with 3 people.
- 2 bags feature scenes with 4 people.

The total dataset size is 22.2 GB, with an average sequence duration of 1 minute per bag. Each person in the scene wore one of the four U-shaped cardboard frames to obtain ground truth data, which is expressed in the `map` base frame (Figure 6.1).

The transformations (`map` -> `mocap`, `mocap` -> `camera_link`) needed to align the estimated 4D poses expressed in the `camera_link` coordinates frame with the ground truth data in the `map` base frame are included in the dataset.



(a)



(b)

Figure 6.1: Visual example of the recorded dataset. (a) An RGB image frame from one of the bag, captured by the camera and showing the participants wearing the U-shaped frames. (b) RViz2 visualization of the corresponding PIC4SeR laboratory map, showing the `map` base frame, the `mocap` frame (where the ground truth is initially expressed), the `camera_link` frame, and the frames of the three people (`U_1`, `U_2`, and `U_3`) tracked by the motion capture system. In the dataset, ground truth data is already transformed from the `mocap` to the `map` frame.

6.3 Results

This section presents the experimental results of the three developed methods, evaluating their tracking consistency, pose estimation accuracy, and computational performance. The analysis includes both quantitative metrics and qualitative examples from representative sequences of the recorded dataset.

6.3.1 Tracking Consistency

The number of ID switches was measured for all three methods on each bag and normalized to account for different bag durations, as defined in Equation (6.1). Table 6.1 presents both the overall average values across the whole dataset and the averages for each scenario type (2 people, 3 people, 4 people).

Table 6.1: Normalized ID switches (switches/minute) for each method.

Method	**Overall**	**2 People**	**3 People**	**4 People**
YOLOv8-Seg + SORT	8.6	3.1	6.5	17.2
YOLOv8-Seg + StrongSORT	2.5	1.5	0.7	6.5
YOLOv8-Pose + StrongSORT	2.1	1.0	0.3	6.0

Results show that StrongSORT's more sophisticated tracking algorithm, and particularly its integration of appearance information in the association phase through the Re-ID model, significantly improves tracking consistency compared to the first method with SORT, reducing ID switches by more than 70% on average.

Regarding the relatively high number of ID switches in the four-people scenarios, this is likely due to experimental constraints rather than the number of people itself. The limited tracking area of the motion capture system (approximately 3.5 m $\times$ 5.5 m), combined with the maximum depth range of the camera (5.0 m), narrowed the area where participants could move, leading to scenes with unrealistic people movements, frequent occlusions, and/or participants moving outside the camera's field of view for long periods. However, even under these challenging conditions, the Re-ID model demonstrates its effectiveness, with both StrongSORT-based methods maintaining significantly lower switch rates compared to the SORT implementation.

The impact of the different methods on tracking consistency is visualized in Figures 6.2 and 6.3, which show trajectory plots from two representative sequences extracted from different bags from the dataset to demonstrate the performances in scenarios with different complexity.

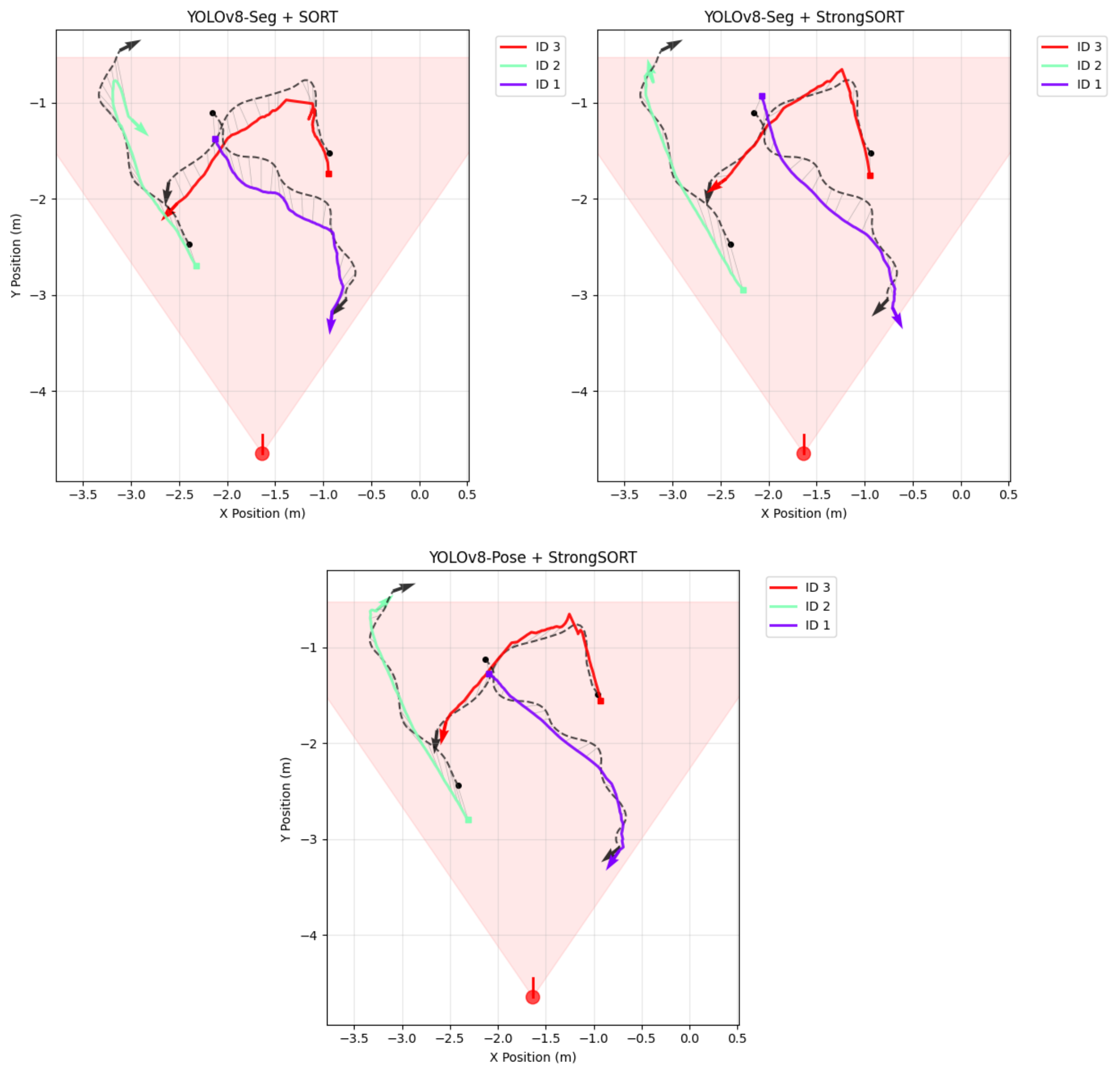


Figure 6.2: Estimated trajectories comparison for a simple three people scene (5 s duration). Each person's estimated trajectory is shown in a different color, while the ground truth trajectories are shown as black dashed lines. The orientation of each person at the end of the sequence is shown by an arrow. The red circle shows the camera position, and the shaded area represents the camera's horizontal field of view and depth range. In this simple scenario with no occlusions and crossing paths, all three methods maintain consistent tracking. The YOLOv8-Pose with StrongSORT method showed a more precise trajectory and orientation estimation.

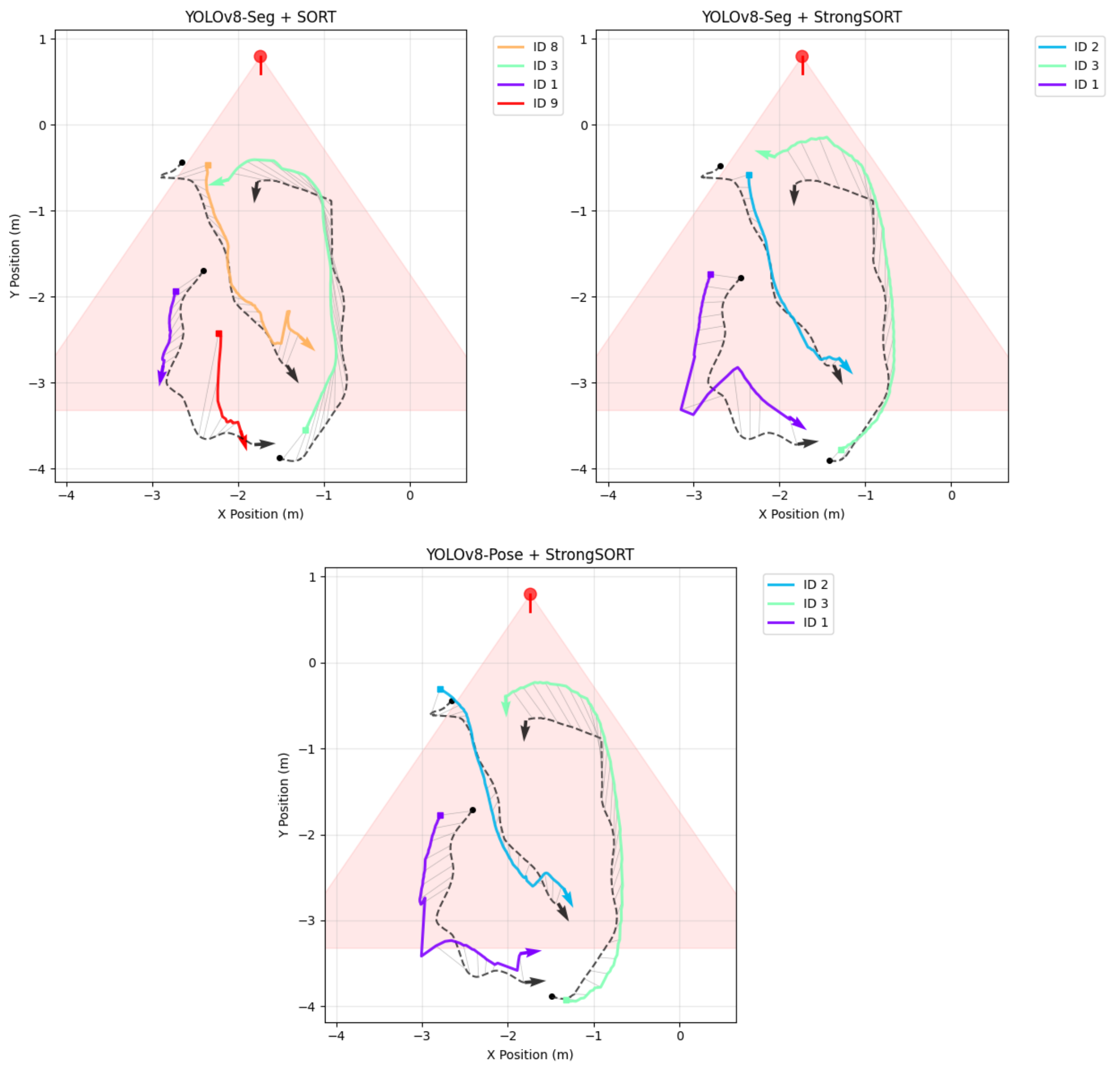


Figure 6.3: Estimated trajectories comparison for a complex three people scene (6 s duration). Each person's estimated trajectory is shown in a different color, while the ground truth trajectories are shown as black dashed lines. The orientation of each person at the end of the sequence is shown by an arrow. The red circle shows the camera position, and the shaded area represents the camera's horizontal field of view and depth range. In this more complex scenario with occlusions and crossing paths, only the two StrongSORT models maintain consistent tracking, showing the impact of the Re-ID network. The SORT-based method has an ID switch during the scene, assigning a new ID (ID 9) to the previously tracked ID 1. The high ID numbers (8, 9) assigned by SORT indicate that multiple prior ID switches had occurred before this scene.

6.3.2 Pose Estimation Accuracy

The accuracy of the 4D pose estimation performed by each method was assessed in terms of RMSE, success rates at different thresholds and, for the orientation, also in terms of circular correlation, as described in section 6.1.2.

Table 6.2 and 6.3 present RMSE and success rates for position and velocity, respectively. Both position and velocity estimation show similar accuracy performance across all methods, with RMSE values ranging from approximately 15 to 30 cm for position and 0.18 to 0.29 m/s for velocity.

Table 6.2: Position (x, y) RMSE and success rates for each method.

Method	**RMSE (m)**		**Success Rate (%)**		
	x	**y**	**@0.15m**	**@0.3m**	**@0.5m**
YOLOv8-Seg + SORT	0.150	0.308	8.4	63.4	84.0
YOLOv8-Seg + StrongSORT	0.160	0.300	18.1	52.6	88.2
YOLOv8-Pose + StrongSORT	0.165	0.283	20.1	58.4	90.1

Table 6.3: Velocity (v_x, v_y) RMSE and success rates for each method.

Method	**RMSE (m/s)**		**Success Rate (%)**		
	vx	**vy**	**@0.15m/s**	**@0.3m/s**	**@0.5m/s**
YOLOv8-Seg + SORT	0.178	0.258	47.3	83.8	93.5
YOLOv8-Seg + StrongSORT	0.213	0.272	21.2	60.4	90.7
YOLOv8-Pose + StrongSORT	0.222	0.285	21.2	60.6	90.2

All methods show higher RMSE values for the y component compared to the x component, both for position and velocity. This difference can be explained with the depth estimation uncertainty from the RGBD camera, which affects the depth direction (y axis in map frame given how the camera was placed to record the dataset) more significantly than the horizontal direction (x axis). Furthermore, when performing the pixel deprojection, each pixel is assigned the depth of the closest 3D point. This means that when a person is oriented perpendicular to the camera, their estimated position tends to be biased towards the camera compared to their actual centroid location, as the depth is measured on their side surface rather than their center.

SORT-based method's better results in terms of RMSE and/or success rates, in particular for velocity, needs further consideration: as shown in section 6.3.1 and in Figure 6.3, this method suffers from numerous ID switches, meaning that it is not able to track people during occlusions or complex scenes, and it is more likely to initiate new tracks once the person is again clearly visible in the scene. The other two methods instead are much more robust in this sense, tracking through occlusions thanks to the Re-ID network. However, during occlusions, when the Kalman filter is not updated with new associations, errors accumulate. This explains the apparent better results of SORT-based method's estimations.

To support this observation, SORT-based method has shown an average segment duration (i.e. the duration of one track with a given ID) 43% lower than YOLOv8-Seg + StrongSORT method, and 49% lower than YOLOv8-Pose + StrongSORT method.

Table 6.4 presents RMSE, correlation and success rates for the orientation estimation. Yolov8-Pose with StrongSORT method demonstrates clear superiority: its direct orientation estimation approach achieves both lower RMSE (32.3°) and higher temporal correlation ($r_\psi = 0.801$) compared to the other two methods that derive orientation from the direction of the velocity vector. This improved performance is particularly evident in the success rates, where it scores 70.6% at the challenging 30° threshold. This means that across all bags, the third method has been able to estimate orientation with an error smaller than 30° for more than 70% of the estimated poses. The higher r_ψ also confirms that thanks to the direct orientation estimation, it suffers much less from the inherent delay introduced by the Kalman Filter when estimating a quantity not directly measured, as is the case for velocity-derived orientation.

Table 6.4: Orientation ψ RMSE, correlation coefficient and success rates for each method.

Method	**RMSE (°)**	**Correlation**	**Success Rate (%)**		
	Ψ	r_ψ	**@30°**	**@45°**	**@90°**
YOLOv8-Seg + SORT	44.4	0.739	52.6	70.6	94.8
YOLOv8-Seg + StrongSORT	61.9	0.535	33.9	48.5	86.3
YOLOv8-Pose + StrongSORT	32.3	0.801	70.6	85.2	97.8

To visualize the estimation differences between the three methods, Figures 6.4, 6.5 and 6.6 show, respectively, the estimated position, velocity and orientation as function of time from a representative 40-second sequence of one person extracted from one of the recorded bags of the dataset.

Overall, these results demonstrate that the YOLOv8-Pose with StrongSORT method provides the most well-rounded performance: it maintains robust tracking through occlusions thanks to the Re-ID network, achieves accuracy levels suitable for social robotics applications both for position and velocity, and significantly outperforms the other methods in orientation estimation thanks to its direct measurement approach.

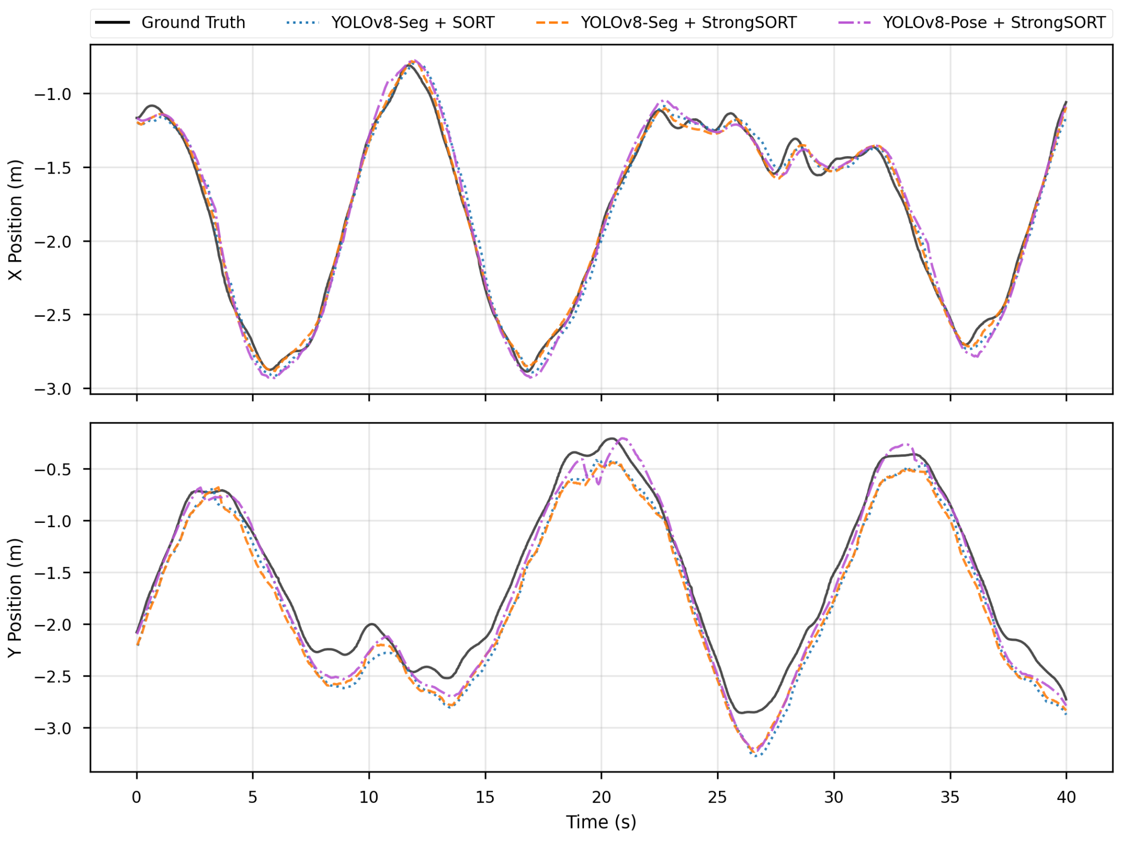


Figure 6.4: Position estimation comparison in x and y coordinates for a 40-second sequence of one person extracted from one of the recorded bags. All methods show good tracking of position, with only minor deviations from the ground truth trajectory. All methods are able to track without ID switches.

6.3.3 Computational Performance

To evaluate real-time capabilities, we measured the computational performance of each method across all recorded bags. Results are shown in Table 6.5, which presents the average frame rates achieved by each method, both overall and categorized by the number of people in the scene.

As expected, the frame rate decreases moving from SORT to StrongSORT, because of the more complex tracking algorithm and because of the Re-ID model, which introduces additional computational overhead. This can be seen by comparing the overall values of the first two methods.

The higher average FPS across all bags achieved by the third method with respect to the second one, while both using StrongSORT as tracking algorithm, is due to the consistently faster inference time of YOLOv8 Pose over YOLOv8 Segmentation.

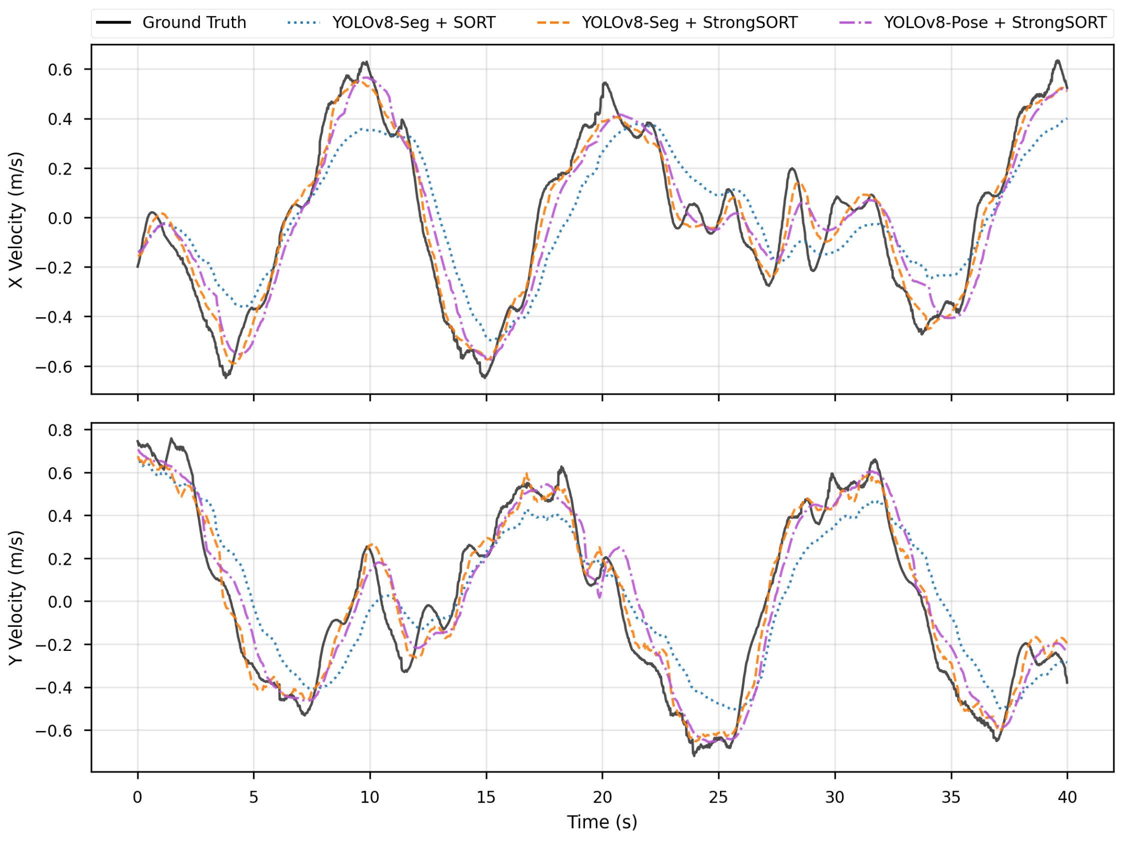


Figure 6.5: Velocity estimation comparison in x and y components for a 40-second sequence of one person extracted from one of the recorded bags. All methods follow the velocity ground truth, though with some smoothing of rapid changes due to the Kalman Filter. The SORT-based method shows slightly more deviation, particularly visible in underestimating velocity peaks.

Table 6.5: Average frame rates (FPS) achieved by each method.

Method	**Overall**	**2 People**	**3 People**	**4 People**
YOLOv8-Seg + SORT	22.9 ± 0.5	23.1	23.2	22.4
YOLOv8-Seg + StrongSORT	18.6 ± 1.0	19.8	18.7	17.3
YOLOv8-Pose + StrongSORT	22.7 ± 1.5	24.1	23.0	21.0

Furthermore, we can notice that methods using the Re-ID model (second and third rows) show a more pronounced decrease in frame rate as the number of people increases. This is expected, as the Re-ID network must process each detection

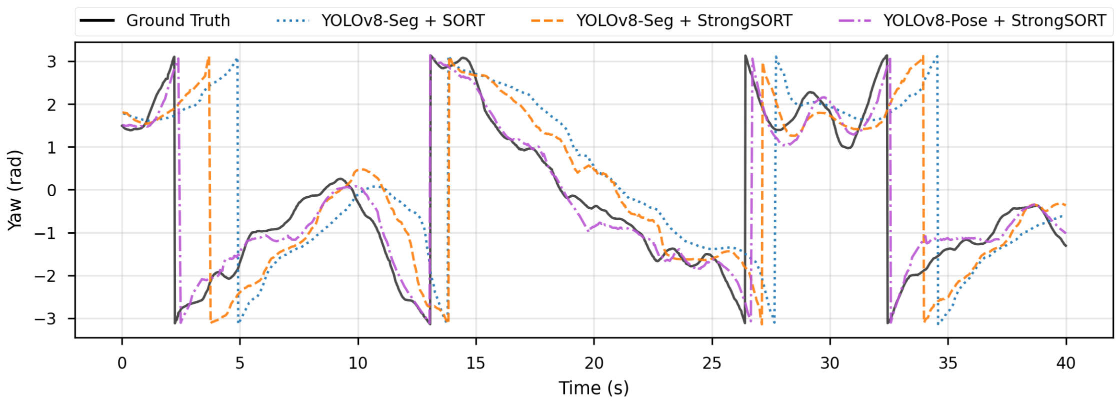


Figure 6.6: Orientation estimation comparison for the same sequence of Figures 6.4 and 6.5. The YOLOv8-Pose with StrongSORT method (purple) shows notably better tracking of orientation changes, with faster response times and closer alignment to ground truth compared to the velocity-based orientation estimates of the other methods. The inherent delay in velocity-derived orientation of the first two methods is visible throughout the whole sequence.

independently to extract appearance features. However, all methods showed real-time performance, maintaining frame rates above 17 FPS (17.3-22.4 FPS) even in the most challenging scenarios with four people. This performance level is well-suited for integration with typical social navigation algorithms, which commonly operate at 10-20 Hz.

Performance improvements from OpenVINO export. To quantify the benefits of models optimization using OpenVINO export, we measured the inference performance of each method using also the original PyTorch models. Table 6.6 presents the average frame rates achieved across all recorded bags.

Table 6.6: Comparison of average frame rates (FPS) of the three algorithms using original PyTorch models vs. OpenVINO models.

	PyTorch models	**OpenVINO models**	**Speed-up Factor**
YOLOv8-Seg + SORT	6.0	22.9	×3.8
YOLOv8-Seg + StrongSORT	2.3	18.6	×8.1
YOLOv8-Pose + StrongSORT	2.5	22.7	×9.1

The OpenVINO optimization led to a significant performance improvements across all methods, in particular for the StrongSORT-based methods, where two neural networks (YOLOv8 and Re-ID) need to operate sequentially.

Chapter 7

Conclusions

The goal of this thesis work has been to study and develop efficient methods for people 4D pose estimation and tracking that can guarantee real-time performance and good accuracy for social robot navigation. Starting from well-known multi-object tracking algorithms (SORT, StrongSORT) originally designed to track objects in the 2D image plane, three different methods were developed to extend them to 3D tracking by integrating RGBD camera data. This allowed estimation and tracking of each person's position (x, y), velocity (v_x, v_y), and orientation ψ in real-world space rather than just image coordinates.

To evaluate and compare the methods, a dataset consisting of multiple RGBD videos was recorded using an Intel® RealSense™ D435i camera, capturing different levels of complexity in terms of number of people (from two to four) and occlusions. Ground truth data was obtained using a Vicon motion capture system after building four U-shaped cardboard frames with reflective markers attached to them, that could be worn by the participant in the tracking area.

The first method, combining YOLOv8 Segmentation with a modified SORT algorithm, established a baseline approach that demonstrated the feasibility of integrating RGBD data with multi-object tracking frameworks. The segmentation masks were used to extract people's centroids in the image plane, which were then deprojected to 3D points using the depth map provided by the RGBD camera. The SORT algorithm was extended to incorporate these 3D positions in its Kalman filter state and in the association phase. While sufficient for simple scenarios, its limitations in handling occlusions and identity switches became evident in more complex scenes.

Building on these observations, the second method was developed by replacing SORT with StrongSORT, a more advanced tracking algorithm. The integration of visual re-identification features improved track consistency and reduced ID switches by more than 70% compared to the baseline approach.

In both methods, orientation was estimated as the heading direction, meaning the angle between the velocity vector estimated by the Kalman filter and the x-axis. To address the intrinsic delay of this velocity-based orientation estimation, in the third method YOLOv8 Segmentation was replaced with YOLOv8 Pose to use the detected body keypoints to directly estimate the orientation from shoulders and hips 3D position and from anthropometric ratio considerations. This approach provided significant improvement in orientation estimation accuracy, with an RMSE of 32.3° and maintaining orientation estimates within 30° of ground truth for more than 70% of frames. It also showed good accuracy in position estimation (RMSE of 16.5 cm and 28.3 cm for x and y coordinates, success score of 90.1% at 50 cm) and velocity estimation (RMSE below 0.3 m/s for both components, success score of 90.2% at 0.5 m/s), while also marginally improving the already good tracking consistency of the second approach, with an even lower average number of ID switches.

Importantly, all three methods maintained real-time performance on the used hardware, with frame rates ranging from 17.3 to 24.1 FPS even in challenging scenarios with four people.

Overall, the results showed that the final method achieved the best well-rounded performance in terms of accuracy and robustness, particularly in high-complexity scenarios, and demonstrated its suitability as an efficient solution for real-time 4D pose estimation and tracking in social robot navigation.

7.1 Future Work

While the developed methods show promising results, several limitations and opportunities for improvement remain.

- **Robust orientation estimation:** The orientation estimation, although significantly improved in the third method, still struggles in scenarios where key body keypoints are occluded. A more robust approach could be to use a dedicated lightweight neural network specifically trained for this task, as already demonstrated in recent literature. While this would add computational overhead, careful optimization could help maintain real-time performance.

- **Leverage depth during detection:** Another possible direction for further development would be to leverage depth information during the detection phase, with a neural network designed and trained to process 4 channels images rather than the common RGB ones. This could improve detection accuracy, especially in challenging conditions or when people are partially occluded, by providing additional spatial context to the detector, and could also solve the systematic bias in the depth direction that our models showed, by generating accurate 3D bounding boxes.

- **Tracking enhancements:** The tracking component could also be enhanced by exploring other recent tracking-by-detection algorithms and integrating camera motion compensation techniques in the tracking pipeline. The latter would be particularly beneficial when deploying the system on mobile robots, where camera movements or vibrations can introduce noise and affect tracking stability.

Appendix A

Mathematical tools

A.1 Kalman Filter

The Kalman filter [44] is a recursive algorithm that provides optimal state estimation for linear systems under the assumption of Gaussian noise. It is widely used in tracking and navigation systems, and in the context of multi-object tracking, it is used to estimate the state $\boldsymbol{x} \in \mathbb{R}^n$ of the tracked objects given the noisy measurements $\boldsymbol{y} \in \mathbb{R}^m$.

The discrete-time Kalman filter, when there is no active control in the system like in object tracking applications, is governed by the following linear difference equations, expressed in state-space form:

$$\begin{cases} \boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{w}_k \\ \boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \end{cases} \tag{A.1}$$

where $\boldsymbol{F}_k$ is the transition matrix, $\boldsymbol{H}_k$ is the observation matrix, and the random variables $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are the process and measurement noise, respectively. These two noise random variables are assumed to be independent and identically distributed (i.i.d), with Gaussian distribution:

$$\boldsymbol{w_k} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k), \qquad \boldsymbol{v_k} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k) \tag{A.2}$$

where $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ are the process and measurement noise covariance matrices, respectively, and in general they are functions of time.

The filter operates in two steps:

1. Prediction step, to estimate the current state based on the previous state:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= \boldsymbol{F}_k \hat{\boldsymbol{x}}_{k-1|k-1} \\ \boldsymbol{P}_{k|k-1} &= \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^\top + \boldsymbol{Q}_k \end{aligned} \tag{A.3}$$

2. Update step, to correct the prediction based on the new measurements:

$$\begin{aligned}
\boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^\top(\boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^\top + \boldsymbol{R}_k)^{-1} \\
\hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_{k|k-1}) \\
\boldsymbol{P}_{k|k} &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1}
\end{aligned} \tag{A.4}$$

Matrix $\boldsymbol{P}$ is the state error covariance matrix, while $\boldsymbol{K}$ is the Kalman gain. Initial values for state estimate, $\hat{\boldsymbol{x}}_{0|0}$, and matrix $\boldsymbol{P}_{0|0}$ are needed to initialize the filter. For quicker convergence, it is common to choose a large $\boldsymbol{P}_{0|0}$. This reflect the initial ignorance about the process.

Regarding the measurement error covariance matrix $\boldsymbol{R}_k$ and the process noise $\boldsymbol{Q}_k$, although they are supposed to reflect the statistics of the noises, the true statistics of the noises is not known or not Gaussian in many practical applications. Therefore, even when an initial guess for their values can be obtained with rational considerations or by measurements, usually a tuning phase for $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ is needed to get the desired performance [45].

A.1.1 Constant Velocity Motion Model

Constant velocity motion model is the most widely used motion model for multi-object tracking applications [46]. If we assume that the state vector contains position and velocity in 3D space:

$$\boldsymbol{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^\top \tag{A.5}$$

The process model for the target as it moves from time $k-1$ to time k (considering only the x component for brevity) is:

$$\begin{aligned}
x_k &= x_{k-1} + \Delta t \dot{x}_{k-1} \\
\dot{x}_k &= \dot{x}_{k-1} \\
\ddot{x}_k &= 0
\end{aligned} \tag{A.6}$$

where $\ddot{x}_k$ is the acceleration and Δt the time between frames. However, perfect constant velocity is an unrealistic assumption in real tracking applications. To model the unknown dynamics we can introduce a piecewise constant white acceleration noise [46]:

$$\begin{aligned}
x_k &= x_{k-1} + \Delta t \dot{x}_{k-1} + \frac{\Delta t^2}{2}\ddot{x}_{k-1} \\
\dot{x}_k &= \dot{x}_{k-1} + \Delta t \ddot{x}_{k-1} \\
\ddot{x}_k &= w_{x,k} \sim \mathcal{N}(0, \sigma_{x,k}^2)
\end{aligned} \tag{A.7}$$

where $\sigma_{x,k}^2$ is a variance that controls the amount of relaxation of the constant velocity assumption. From this equation, considering also the ones for y and z components, we can obtain the state transition matrix $\boldsymbol{F}_k$ and the process noise $\boldsymbol{w}_k$:

$$\boldsymbol{F}_k = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{w}_k = \begin{bmatrix} \frac{\Delta t^2}{2} w_{x,k} \\ \frac{\Delta t^2}{2} w_{y,k} \\ \frac{\Delta t^2}{2} w_{z,k} \\ \Delta t w_{x,k} \\ \Delta t w_{y,k} \\ \Delta t w_{z,k} \end{bmatrix} \tag{A.8}$$

Given that $\boldsymbol{w_k} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k)$ (Equation (A.2)) and assuming for simplicity $\sigma_{x,k}^2 = \sigma_{y,k}^2 = \sigma_{z,k}^2 = \sigma_a^2$ and that the acceleration noise components are uncorrelated, we can determine $\boldsymbol{Q}_k$ by computing the covariance of $\boldsymbol{w}_k$:

$$\boldsymbol{Q}_k = \mathbb{E}\left[\boldsymbol{w}_k \boldsymbol{w}_k^\top\right] = \sigma_a^2 \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 & 0 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 & 0 \\ 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 \end{bmatrix} \tag{A.9}$$

Finally, using the same approach, we can derive matrices $\boldsymbol{H}_k$ and $\boldsymbol{R}_k$:

$$\boldsymbol{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{v}_k = \begin{bmatrix} v_{x,k} \\ v_{y,k} \\ v_{z,k} \end{bmatrix} \tag{A.10}$$

$$\boldsymbol{R}_k = \mathbb{E}\left[\boldsymbol{v}_k \boldsymbol{v}_k^\top\right] = \sigma_m^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.11}$$

under the assumption of independent noise in each dimension with same variance σ_m^2 for simplicity.

A.2 Mahalanobis Distance

The Mahalanobis distance is a multivariate distance metric between a point $\boldsymbol{P}$ and a distribution $\boldsymbol{D}$, introduced by P. C. Mahalanobis in 1936 [47]. Unlike the Euclidean distance, it accounts for the correlation between variables and it is also scale-invariant, making it a more meaningful metric for measuring dissimilarity between data points in multi-dimensional space.

Given a point $\boldsymbol{x} = (x_1, x_2, ..., x_n)^\top$, representing the observation, and a distribution with mean $\boldsymbol{\mu} = (\mu_1, \mu_2, ..., \mu_n)^\top$ and positive semi-definite covariance matrix $\boldsymbol{S}$, the Mahalanobis distance is defined as:

$$d_M(\boldsymbol{x}) = \sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{S}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})} \tag{A.12}$$

Intuitively, it can be seen as a multivariate equivalent of the z-score, $z = (x - \mu)/\sigma$. Just like the z-score measures how many standard deviations a data point x is from the mean value μ in a univariate distribution with standard deviation σ, the Mahalanobis distance measures the distance of a point from the centroid of the multivariate distribution, taking in consideration the shape of the latter through its covariance matrix $\boldsymbol{S}$. It is commonly used to determine whether a sample is an outlier [48].

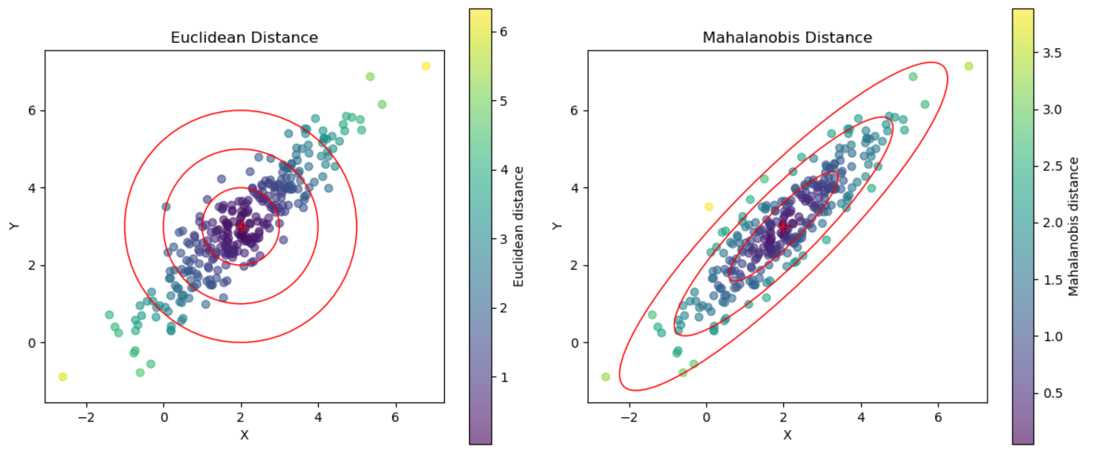


Figure A.1: Comparison of Euclidean and Mahalanobis distances for a bivariate normal distribution with correlated variables. Isolines of equal distance from the mean (red cross) are shown in red.

Bibliography

[1] Thais Campos, Adam Pacheck, Guy Hoffman, and Hadas Kress-Gazit. «SMT-based control and feedback for social navigation». In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 5005–5011 (cit. on p. 1).

[2] Tim Salzmann, Hao-Tien Lewis Chiang, Markus Ryll, Dorsa Sadigh, Carolina Parada, and Alex Bewley. «Robots that can see: Leveraging human pose for trajectory prediction». In: *IEEE Robotics and Automation Letters* (2023) (cit. on p. 1).

[3] Prajakta Saraf, Sanika Watve, and Anagha Kulkarni. «Object Detection: Literature Review». In: *Proceedings of the 14th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2022)*. Cham: Springer Nature Switzerland, 2023, pp. 485–494 (cit. on p. 3).

[4] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. *cs231n, Lecture 8 — Spatial Localization and Detection, Slide 8*. 2016. URL: `http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf` (cit. on pp. 4, 9).

[5] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022 (cit. on p. 4).

[6] Nikhil Tomar. *What is Intersection Over Union (IoU) in Object Detection?* 2021. URL: `https://nikhilroxtomar.medium.com/what-is-intersection-over-union-iou-in-object-detection-idiot-developer-39448ac56431` (cit. on p. 5).

[7] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. «Object detection with deep learning: A review». In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232 (cit. on p. 5).

[8] Ben Dickson. *Object Detection and Deep Learning: Everything You Need to Know*. 2021. URL: `https://bdtechtalks.com/2021/06/21/object-detection-deep-learning/` (cit. on p. 6).

[9] J Redmon. «You only look once: Unified, real-time object detection». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016 (cit. on p. 7).

[10] Valliappa Lakshmanan, Martin Görner, and Ryan Gillard. *Practical machine learning for computer vision*. " O'Reilly Media, Inc.", 2021 (cit. on p. 7).

[11] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: `https://github.com/ultralytics/ultralytics` (cit. on p. 7).

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. «Microsoft coco: Common objects in context». In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755 (cit. on pp. 8, 35).

[13] Esraa Samkari, Muhammad Arif, Manal Alghamdi, and Mohammed A Al Ghamdi. «Human pose estimation using deep learning: a systematic literature review». In: *Machine Learning and Knowledge Extraction* 5.4 (2023), pp. 1612–1659 (cit. on p. 9).

[14] Satya Mallick. *Human Pose Estimation using Keypoint RCNN in PyTorch*. 2021. URL: `https://learnopencv.com/human-pose-estimation-using-keypoint-rcnn-in-pytorch/` (cit. on p. 10).

[15] Mang Ye, Jianbing Shen, Gaojie Lin, Tao Xiang, Ling Shao, and Steven CH Hoi. «Deep learning for person re-identification: A survey and outlook». In: *IEEE transactions on pattern analysis and machine intelligence* 44.6 (2021), pp. 2872–2893 (cit. on p. 11).

[16] Nazia Perwaiz, Muhammad Moazam Fraz, and Muhammad Shahzad. «Stochastic attentions and context learning for person re-identification». In: *PeerJ Computer Science* 7 (2021), e447 (cit. on p. 11).

[17] Kaiyang Zhou, Yongxin Yang, Andrea Cavallaro, and Tao Xiang. «Omni-scale feature learning for person re-identification». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 3702–3712 (cit. on p. 12).

[18] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. «Multiple object tracking: A literature review». In: *Artificial Intelligence* 293 (2021), p. 103448. ISSN: 0004-3702. DOI: `https://doi.org/10.1016/j.artint.2020.103448` (cit. on p. 13).

[19] Laura Leal-Taixé. «Multiple object tracking with context awareness». In: *arXiv preprint arXiv:1411.7935* (2014) (cit. on p. 13).

[20] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. «BoT-SORT: Robust associations multi-pedestrian tracking». In: *arXiv preprint arXiv:2206.14651* (2022) (cit. on pp. 14, 15).

[21] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris Kitani. «Observation-centric sort: Rethinking sort for robust multi-object tracking». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 9686–9696 (cit. on p. 14).

[22] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. «Strongsort: Make deepsort great again». In: *IEEE Transactions on Multimedia* 25 (2023), pp. 8725–8737 (cit. on pp. 15, 17).

[23] Gerard Maggiolino, Adnan Ahmad, Jinkun Cao, and Kris Kitani. «Deep oc-sort: Multi-pedestrian tracking by adaptive re-identification». In: *2023 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2023, pp. 3025–3029 (cit. on p. 15).

[24] Harold W Kuhn. «The Hungarian method for the assignment problem». In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97 (cit. on p. 15).

[25] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. «Simple online and realtime tracking». In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3464–3468 (cit. on p. 16).

[26] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. «Simple online and realtime tracking with a deep association metric». In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 3645–3649 (cit. on p. 17).

[27] R Mukundan and K R Ramakrishnan. *Moment Functions in Image Analysis — Theory and Applications*. WORLD SCIENTIFIC, 1998. DOI: 10.1142/3838 (cit. on p. 25).

[28] Kardi Teknomo. «Microscopic pedestrian flow characteristics: Development of an image processing data collection and simulation model». In: *arXiv preprint arXiv:1610.00029* (2016) (cit. on p. 28).

[29] Irving P Herman. *Physics of the human body*. Springer, 2016 (cit. on p. 31).

[30] Jianshu Li, Jian Zhao, Yunchao Wei, Congyan Lang, Yidong Li, Terence Sim, Shuicheng Yan, and Jiashi Feng. «Multiple-human parsing in the wild». In: *arXiv preprint arXiv:1705.07206* (2017) (cit. on p. 35).

[31] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. «Scalable person re-identification: A benchmark». In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1116–1124 (cit. on p. 36).

[32] LearnOpenCV. *Post-Training Quantization with OpenVINO Toolkit.* URL: https://learnopencv.com/post-training-quantization-with-openvino-toolkit/ (cit. on p. 38).

[33] Kaiyang Zhou and Tao Xiang. «Torchreid: A library for deep learning person re-identification in pytorch». In: *arXiv preprint arXiv:1910.10093* (2019) (cit. on p. 37).

[34] Intel® RealSense™ SDK Manager Sergey Dorodnicov. *The basics of stereo depth vision.* 2018. URL: https://www.intelrealsense.com/stereo-depth-vision-basics/ (cit. on pp. 39, 40).

[35] OpenCV Documentation. *Depth Map from Stereo Images.* 2024. URL: https://docs.opencv.org/4.x/dd/d53/tutorial_py_depthmap.html (cit. on p. 39).

[36] Intel RealSense blog. *Beginner's Guide to Depth.* 2019. URL: https://www.intelrealsense.com/beginners-guide-to-depth/ (cit. on p. 40).

[37] Sergey Dorodnicov, Anders Grunnet-Jepsen, and Guoping Wen. *Projection, Texture-Mapping and Occlusion with Intel® RealSense™ Depth Cameras.* Rev 0.3. 2024. URL: https://dev.intelrealsense.com/docs/projection-texture-mapping-and-occlusion-with-intel-realsense-depth-cameras (cit. on p. 40).

[38] OpenCV Documentation. *Camera Calibration and 3D Reconstruction.* OpenCV 2.4.13.7. 2019. URL: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (cit. on p. 42).

[39] Intel Corporation. *Intel® RealSense™ Product Family D400 Series Datasheet.* Revision 019, October 2024. 2024. URL: https://www.intelrealsense.com/download/21345/?tmstv=1697035582 (cit. on p. 43).

[40] Francisco Martín, José Guerrero, A. Garcia, Francisco Rodríguez Lera, and Vicente Matellán. «MOCAP4ROS2: An Open Source Framework for Motion Capture Systems in Robotics». In: *Proceedings of the 18th International Symposium on Open Collaboration.* Sept. 2022, pp. 1–3. DOI: 10.1145/3555051.3555076 (cit. on p. 47).

[41] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. «Robot Operating System 2: Design, architecture, and uses in the wild». In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074 (cit. on p. 48).

[42] Morgan Quigley. «ROS: an open-source Robot Operating System». In: *IEEE International Conference on Robotics and Automation.* 2009. URL: https://api.semanticscholar.org/CorpusID:6324125 (cit. on p. 48).

[43] KV Mardia. «Linear-circular correlation coefficients and rhythmometry». In: *Biometrika* (1976), pp. 403–405 (cit. on p. 52).

[44] Youngjoo Kim and Hyochoong Bang. «Introduction to Kalman Filter and Its Applications». In: *Introduction and Implementations of the Kalman Filter*. Ed. by Felix Govaers. Rijeka: IntechOpen, 2018. Chap. 2. DOI: `10.5772/intechopen.80600`. URL: `https://doi.org/10.5772/intechopen.80600` (cit. on p. 66).

[45] Greg Welch and Gary Bishop. «An Introduction to the Kalman Filter». In: *Proc. Siggraph Course* 8 (Jan. 2006) (cit. on p. 67).

[46] Nathanael Lemessa Baisa. «Derivation of a Constant Velocity Motion Model for Visual Tracking». In: *ArXiv* abs/2005.00844 (2020). URL: `https://api.semanticscholar.org/CorpusID:218486932` (cit. on p. 67).

[47] «Reprint of: Mahalanobis, P.C. (1936) "On the Generalised Distance in Statistics."» In: *Sankhya A* 80 (2018), pp. 1–7. URL: `https://api.semanticscholar.org/CorpusID:239595337` (cit. on p. 68).

[48] Richard Brereton. «The Mahalanobis distance and its relationship to principal component scores». In: *Journal of Chemometrics* 29 (Mar. 2015). DOI: `10.1002/cem.2692` (cit. on p. 69).