

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF NUCLEAR SCIENCES AND PHYSICAL
ENGINEERING

Department of Software Engineering

Study program: Applications of Informatics in Natural Sciences
Specialization: –



Odhad a sledování polohy člověka pomocí jediné RGB kamery

Estimation and Tracking of the Human Pose with a Single RGB camera

MASTER'S THESIS

Author: Bc. Antonín Čech
Supervisor: Ing. Adam Novozámský, Ph.D.
Year: 2024

České Vysoké Učení Technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Akademický rok 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Antonín Čech
Studijní program: Aplikace informatiky v přírodních vědách
Název práce česky: Odhad a sledování polohy člověka pomocí jediné RGB kamery
Název práce anglicky: Estimation and Tracking of the Human Pose with a Single RGB camera
Jazyk práce: Anglicky

Pokyny pro vypracování:

1. Seznamte se s problematikou odhadu polohy člověka pomocí moderních technik rozpoznávání obrazu. Na základě této rešerše vyberte několik metod, které budete v práci dále používat.
2. Stáhněte několik volně dostupných anotovaných datasetů, na které jste během rešerše v literatuře narazili. Dále vytvořte soubor videí bez anotací pro testovací účely.
3. U vybraných metod proveďte porovnání úspěšnosti a rychlosti na získaných datech.
4. Navrhněte metodu pro sledování osob ve videu.
5. Implementujte GUI pro vizualizaci jednotlivých metod detekce a následného sledování osob.

Doporučená literatura:

- [1] Cao, Z., Hidalgo, G., Simon, T., Wei, S., & Sheikh, Y. (2021). OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1), 172–186.
- [2] Liu, W., & Mei, T. (2022). Recent Advances of Monocular 2D and 3D Human Pose Estimation: A Deep Learning Perspective. *ACM Comput. Surv.*
- [3] Song, L., Yu, G., Yuan, J., & Liu, Z. (2021). Human pose estimation and its application to action recognition: A survey. *Journal of Visual Communication and Image Representation*, 76, 103055.
- [4] Dubey, S., & Dixit, M. (2022). A comprehensive survey on human pose estimation approaches. *Multimedia Systems*.
- [5] Surówka, A. (2021). Real-time Multi Pose Trajectory Tracking based on OpenPose Keypoints. *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*.
- [6] Ning, G., Pei, J., & Huang, H. (2020). LightTrack: A Generic Framework for Online Top-Down Human Pose Tracking. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

Jméno a pracoviště vedoucího práce:


Ing. Adam Novozámský, Ph.D.

Ústav teorie informace a automatizace AV ČR, v.v.i.
Pod Vodárenskou věží 4, 182 00, Praha 8

Jméno a pracoviště konzultanta:

Ing. Tomáš Kerepecký


Ústav teorie informace a automatizace AV ČR, v.v.i.
Pod Vodárenskou věží 4, 182 00, Praha 8


.....
vedoucí práce

Datum zadání diplomové práce: 14. 10. 2022

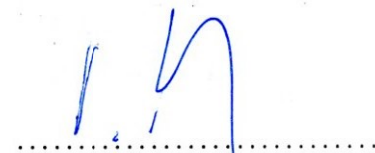
Termín odevzdání diplomové práce: 3. 5. 2023

Doba platnosti zadání je dva roky od data zadání.


.....
garant oboru


.....
vedoucí katedry




.....
děkan

V Praze dne 14. 10. 2022

Statement

I declare that I have developed my master's thesis independently and have used only materials (literature, projects, software, etc.) listed in the attached list.

In Prague on 8.1.2024

Antonín Čech

Bc. Antonín Čech

Acknowledgment

Thank you Ing. Adam Novozámský, Ph.D. for guiding my master's thesis and for stimulating suggestions that enriched it.

Bc. Antonín Čech

Název práce:

Odhad a sledování polohy člověka pomocí jediné RGB kamery

Autor: Bc. Antonín Čech

Studijní program: Applications of Informatics in Natural Sciences

Specializace: –

Druh práce: Master's Thesis

Vedoucí práce: Ing. Adam Novozámský, Ph.D.

Institute of Information Theory and Automation of the CAS
Pod Vodárenskou věží 4, 182 00, Prague 8

Konzultant: –

Abstrakt: Odhad a sledování lidské pozice jsou základní úkoly v počítačovém vidění. Cílem této práce je poskytnout srovnání některých dostupných metod pro odhad a sledování pozic a vytvořit aplikaci pro jejich vizualizaci. Nejprve je popsán problém odhadu pozice spolu s vybranými metodami, vybranými datovými soubory a jejich metrikami přesnosti. Dále je obdobně popsáno sledování pozice s metrikami přesnosti, vybranými metodami a datovými soubory. Výsledky pro všechny metody na všech souborech dat jsou poté prezentovány, diskutovány a porovnávány. Poté je popsána implementace a použití aplikace. V závěru práce jsou konfrontovány cíle a je uveden závěr, které metody jsou nejlepší z hlediska přesnosti a rychlosti. Aplikace je volně dostupná na <https://github.com/cechantonin/PoseEstimationApp>.

Klíčová slova: odhad lidské pozice, sledování lidské pozice, počítačové vidění, neuronové sítě

Title:

Estimation and Tracking of the Human Pose with a Single RGB camera

Author: Bc. Antonín Čech

Abstract: Human pose estimation and tracking are fundamental tasks in computer vision. The goal of this thesis is to provide a comparison between some available methods for pose estimation and tracking and create an application for their visualization. First, the problem of pose estimation is described along with the selected methods, selected datasets and their accuracy metrics. Next, pose tracking is similarly described with the accuracy metrics and the selected methods and datasets. The results for all methods on all datasets are then presented, discussed and compared. After that, the implementation and the usage of the application are described. At the end of the thesis, the goals are confronted and a conclusion is given which methods are the best in terms of accuracy and speed. The application was made publicly accessible at <https://github.com/cechantonin/PoseEstimationApp>.

Key words: human pose estimation, human pose tracking, computer vision, neural networks

Contents

Introduction	9
1 Pose estimation	11
1.1 Datasets	13
1.2 Methods	13
1.2.1 Top-down approach	15
1.2.2 Bottom-up approach	16
1.3 Evaluation metrics	18
2 Pose tracking	20
2.1 Datasets	22
2.1.1 MOT17	22
2.1.2 MOT20	23
2.1.3 Created dataset	23
2.2 Methods	24
2.2.1 ByteTrack	24
2.2.2 QDTrack	25
2.2.3 Greedy approach	25
2.2.4 OC-SORT	26
2.3 Evaluation metrics	27
2.3.1 MOTA	27
2.3.2 HOTA	29
3 Experiments	31
3.1 Pose estimation	32
3.1.1 MS COCO	32
3.1.2 CrowdPose	33
3.2 Pose tracking	34
3.2.1 MOT17 and MOT20	34
3.2.2 Created dataset	36
4 Application	42
4.1 Implementation	42
4.2 Usage	44
Conclusion	46
References	48

Appendix	53
A Pose tracking dataset	53
B Application source code	54
B.1 Main event loop and user interface	54
B.2 Pose estimation and tracking functions	68

Introduction

Human pose estimation and tracking are fundamental tasks in computer vision. Pose estimation provides spatial information about a person's body joints from images or videos, while pose tracking aims to provide spatial and temporal information about a person's body. The estimation of human pose provides useful information about human gestures and posture. This information can be used in many applications like motion capture, action recognition, human-computer interactions, etc. On the other hand, human pose tracking builds upon pose estimation and, in addition to spatial information, provides temporal data about the person. Pose tracking has similar use cases as pose estimation, but the temporal information is useful for example in surveillance, where it can be used to predict or prevent accidents from a security camera feed.

This thesis was written up as a part of the AISEE project at the Institute of Information Theory and Automation of the Czech Academy of Sciences. The AISEE or Artificial Intelligence based Search Environment for video/photo project [7] aims to provide a specialized object detection software platform for the use in AI forensic data mining. This will enable the AI to be trained directly in the environment of the Police of the Czech Republic using semi-automatic and weakly supervised learning. The project will also address the ethics, legal framework and human rights issues of AI usage in a research report.

The aim of pose estimation is to locate the positions of human keypoints, which are key body joints or features such as the shoulders, wrists, knees, eyes, head etc. The number of keypoints sought differs between methods and datasets. After the keypoints are found, they are grouped together by the person they represent as well as interconnected in correct pairs to form a human skeleton. Despite the impressive progress made in human pose estimation, several challenges remain. One major challenge is the accurate localization of keypoints in realistic scenarios such as occlusion, partial visibility, and self-occlusion caused by body parts overlapping. Additionally, all current state-of-the-art methods struggle with efficient real-time pose estimation without the need for a powerful system, which would allow the use of pose estimation in embedded systems or autonomous robots for example.

Human pose tracking aims to keep track of which skeleton or bounding box belongs to each person across multiple frames in a video. This is done by correctly associating a unique identification number to each person in each frame. Similar to pose estimation, several challenges remain to be solved such as occlusion between multiple people or with the environment, rapid movement and high compute demand to

name a few.

In this thesis, an overview of pose estimation and pose tracking methods is presented along with their associated evaluation metrics and popular datasets. The first chapter is focused on the pose estimation task and presents the two paradigms according to which all pose estimation methods are developed. Subsequently, two of the most popular datasets for pose estimation are described along with their keypoint annotations. Then, several methods from each paradigm are described and lastly, the evaluation metrics for single image pose estimation are presented.

Second chapter contains an overview of the pose tracking task. The main objective of pose tracking is described along with its challenges and usage, followed by the explanation of the main approaches of pose tracking methods. Next, two commonly used datasets and the dataset created in this thesis are described. The main methods which were tested are then introduced with the specifics of how they operate. At the end of the chapter, the most widely adopted metric MOTA is described along with its problems, followed by the introduction of HOTA which aims to mitigate problems with MOTA.

Third chapter is dedicated to the experiments with the methods, their results and the discussion about the results. Firstly, all the preliminary information about the tests is described. Then, the results for pose estimation methods on the chosen datasets are presented along with the discussion about the results for each of the dataset. Results of pose tracking are then presented and discussed for the standard datasets followed by a discussion on the accuracy of the methods on the created dataset.

The last chapter describes the implementation of the visualization application which was the last objective of this thesis. Firstly, details of the implementation are presented, such as packages used, threading implementation, etc. Lastly, the usage of the application is described.

In the conclusion of the thesis, the goal from the assignment are confronted.

Chapter 1

Pose estimation

The goal of pose estimation is to locate the positions of human keypoints and determine the correct interconnections between them to create a skeleton which represents the given person. Definition and number of keypoints differ between dataset annotations and most frequently represent key body joints or features such as the shoulders, wrists, knees, eyes, head etc. After the keypoints are found, they are grouped together by the person they represent as well as interconnected in correct pairs to form a human skeleton.

Despite the impressive progress made in human pose estimation, several challenges remain. One major challenge is the accurate localization of keypoints in realistic scenarios such as occlusion, partial visibility, and self-occlusion caused by body parts overlapping. Additionally, environment lighting and person clothing can significantly impact the accuracy of the estimated pose as shadows, reflections and different clothing can alter the body features, thus generating false-positives. Another major challenge is the high compute demand. Although many methods achieved fast and accurate results on high-end hardware, all current state-of-the-art methods are not viable for deployment on low power or embedded systems which could improve many pose estimation use cases.

Pose estimation has uses in many different fields like human-computer interactions, motion-capture, healthcare, security, virtual and augmented reality, etc. An example of human-computer interaction is eye gaze tracking [26] where pose estimation, specifically face estimation, is used to correctly detect the spot a person is looking at. Another example is the usage of pose estimation in robots [31] to correctly detect a human and react accordingly. Motion-capture is another use case where it can be used to create animations [39] for video games and movies without the need for an expensive motion tracking equipment. A use case can also be found in healthcare where it can be used to monitor a patients activity. [23] Security is another field where pose estimation, more precisely action recognition [9], can be utilized to recognize or predict illegal activity. This is especially useful when paired with pose tracking which described in Chapter 2.

Multi-person pose estimation methods generally follow one of two paradigms: top-down or bottom-up. However, there are some exceptions which deviate from these

approaches. For instance, Stacked hourglass network [5] repeatedly uses top-down and bottom-up processing or Deeply Learned Compositional Model [44] that works in two stages, one is top-down, second one is bottom-up.

Methods designed by the top-down approach work in two steps. First, an object detector is used to detect all regions of the image where a person may be located with a certain confidence value. When the confidence value is above a certain threshold, a bounding box of the person is then extracted from the region. The second step is to perform a single-person pose estimation on each of the bounding boxes to localize all keypoints of the person. There are many single-person pose estimation methods, but most of them fall into one of two categories: keypoint regression-based approaches and heatmap-based approaches [37]. Keypoint regression-based approaches work by regressing the position of keypoints directly from the image (e.g., AlphaPose [13], DeepPose [6], Iterative Error Feedback [16]). On the other hand, heatmap-based approaches generate a confidence heatmap of all keypoints first, see Figure 1.1, and then according to the heatmap estimates the location of all keypoints (e.g., HRNet [24]).



Figure 1.1: Keypoint location estimation using heatmaps. Figure from [36].

There are several differences between keypoint regression-based and heatmap-based pose estimation approaches. Keypoint-based approach is simpler to implement, less computationally demanding but more challenging to train, whereas heatmap-based approach is generally more accurate and robust [37]. Arguably the biggest difference is the ability to use the heatmap-based approach in multi-person scenarios, which is why heatmap keypoint estimation is used by almost all of the bottom-up methods.

Similarly to top-down methods, bottom-up methods have two steps, the order is reversed compared to top-down methods. First, keypoints of all persons on the image are estimated via a heatmap-based algorithm. Afterwards, all detected keypoints are grouped together by the person they belong to and the correct keypoint interconnections are determined to create the skeleton. Many research papers, that propose novel approaches on keypoint grouping, have been published. For instance, DEKR [50] and DeeperCut [10] propose grouping by direct regression of keypoints, OpenPose [49] introduces the concept of a vector field named Part Affinity Field used to solve a bipartite graph, Associative embedding [4] and HigherHRNet [8]

predict tags for each of the keypoints and groups the keypoint so that keypoint with the same tag belong to the same person.

Altogether, it is impossible to determine which of these two paradigms is better since both of them have their strengths and weaknesses. For example, top-down methods are heavily dependent on accurate human detection algorithms and are able to provide better estimations for persons far away from the camera thanks to detection cropping. In contrast, bottom-up methods are more accurate in crowded scenarios with high occlusion but requires solving the keypoint grouping problem which is an NP-hard problem. [43]

1.1 Datasets

Microsoft Common Objects in Context [40] dataset was first released in 2014 and originally was intended for use in object detection, classification and segmentation problems. Annotations for human pose estimation, which were added in 2016, contain the positions of 17 keypoints in total. These keypoints include 12 body joints, specifically ankles, knees, hips, shoulders, elbows and wrists. The last 5 keypoints are facial features which include ears, eyes and nose. An example of a human pose skeleton with the COCO keypoints annotation is shown in Figure 1.2. The validation set consists of a total of 5,000 annotated images, though not all of these images contain humans.

Many datasets designed for pose estimation are dominated by uncrowded scenes with only few persons in each image and little to no occlusion. The CrowdPose [17] dataset was developed to provide a set of crowded images. A metric named *Crowd index* was proposed [17] and the CrowdPose dataset achieves near uniform distribution of *Crowd index*, in other words each level of occlusion is included in the dataset by the same part. This can be also seen when comparing the average bounding box intersection over union between CrowdPose and COCO which have 0.27 and 0.06 respectively [17]. The annotations provide the locations of 14 keypoints on more than 80,000 people. These keypoints include the same 12 body joints as COCO, but the head is only annotated by the neck and the top of the head. An example of the CrowdPose keypoints annotation is shown in Figure 1.3. The testing subset of CrowdPose consists of 8,000 images.

1.2 Methods

Due to the shear number of state-of-the-art methods that have been published in the recent years, only a few were chosen for testing in this thesis. Methods were chosen with high emphasis on implementation in Python and the usage of PyTorch with CUDA support for the purpose of easier integration in the GUI application. All top-down methods were tested with the YoloV3 object detector trained on MS COCO for the initial human detection.

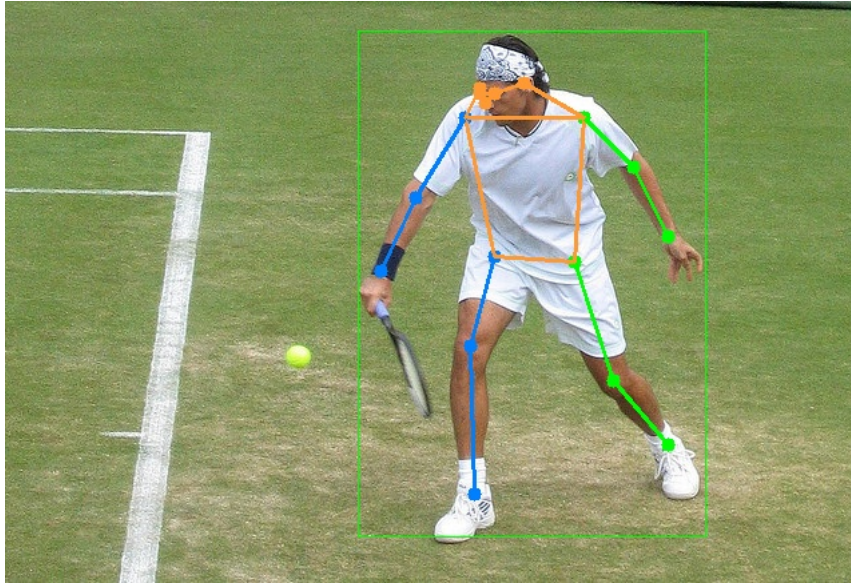


Figure 1.2: Example of the COCO human pose keypoints.

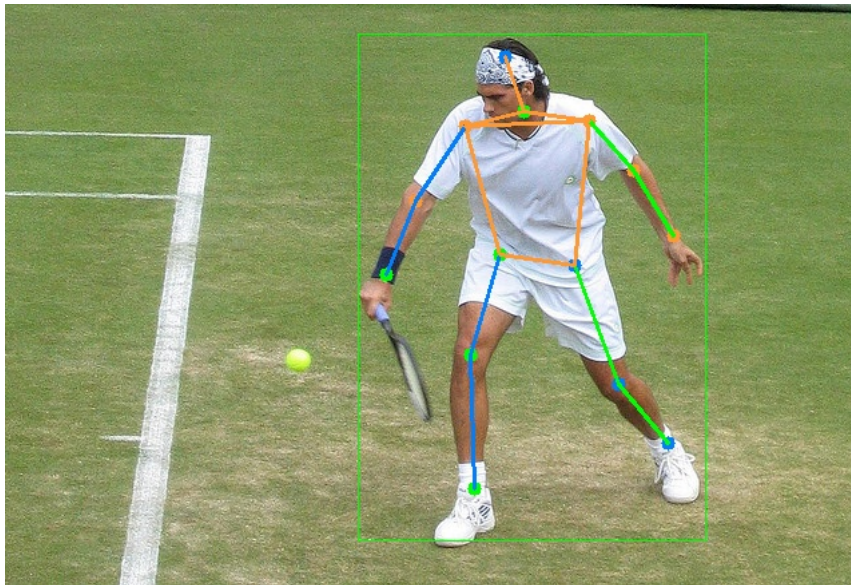


Figure 1.3: Example of the CrowdPose human pose keypoints.

1.2.1 Top-down approach

One of the most notable methods for pose estimation is HRNet [24]. It is a heatmap-based deep-learning single-person pose estimation method which, given a bounding box of a detected person, generates a heatmap of a probable keypoint locations. Actual keypoint locations are calculated from the heatmap by finding the point with the highest heat value and slightly shifting it towards the second highest value. The network itself consists of several high-to-low resolution subnetworks which operate in parallel and the image is repeatedly exchanged between them while also being upscaled or downscaled depending on the receiving subnetwork [24]. The architecture of HRNet is in Figure 1.4. Thanks to the heatmap-based approach, this method was modified to estimate keypoints in multi-person scenarios and many state-of-the-art bottom-up methods use the modified HRNet as the backbone for estimating keypoints.

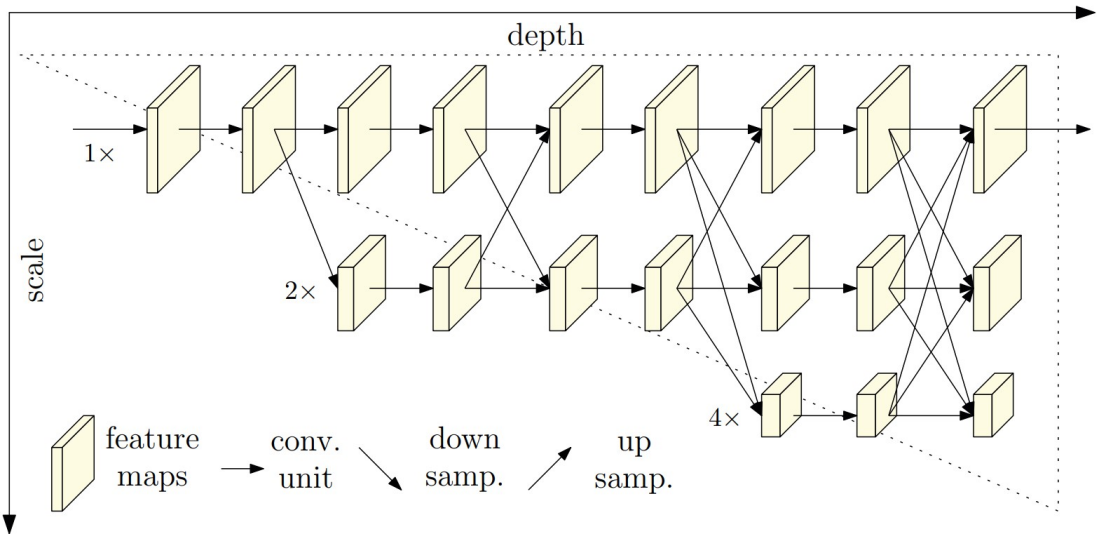


Figure 1.4: Illustration of the HRNet network architecture. Figure from [24].

The Stacked hourglass network [5] was proposed in 2016 and was amongst the first methods for human pose estimation that used deep neural networks. Moreover, it is one of a few methods that cannot be classified under the top-down or the bottom-up approach since it repeatedly uses top-down and bottom-up inference. The network consists of multiple "hourglass" modules which are stacked in series, see Figure 1.5. These modules are used to process the image down to a low resolution (bottom-up processing) and after reaching the lowest resolution, the network begins the upsampling and combination of features across the resolutions of the image (top-down processing) [5].

Many top-down methods suffer from the early commitment problem, as stated in [13]. This refers to the inability to rectify a falsely detected person which happens when a person is detected with a lower confidence score than the method's threshold. To combat this, a top-down method named AlphaPose [13] was published. AlphaPose sets the threshold for correct person detections very low so that only the worst detections are discarded at this stage. Then, pose estimation is done

on these bounding boxes and the redundant poses are discarded using parametric pose non-maximum-suppression [13].

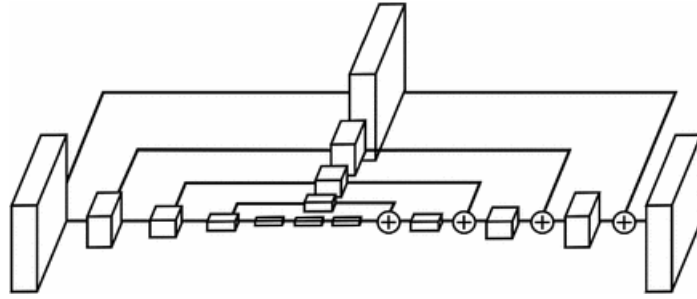


Figure 1.5: Illustration of a single "hourglass" module. Figure from [5].

MediaPipe [29] is a Google project which provides solutions for common problems in computer vision like object detection, object tracking, face detection, hand detection and pose estimation. The pose estimator is designed around the model BlazePose [42] which is a lightweight model optimized to run on mobile devices. BlazePose estimates a total of 33 keypoints which can be seen in Figure 1.6. MediaPipe uses Google's own person detector and both the detector and estimator are trained on Google proprietary data.

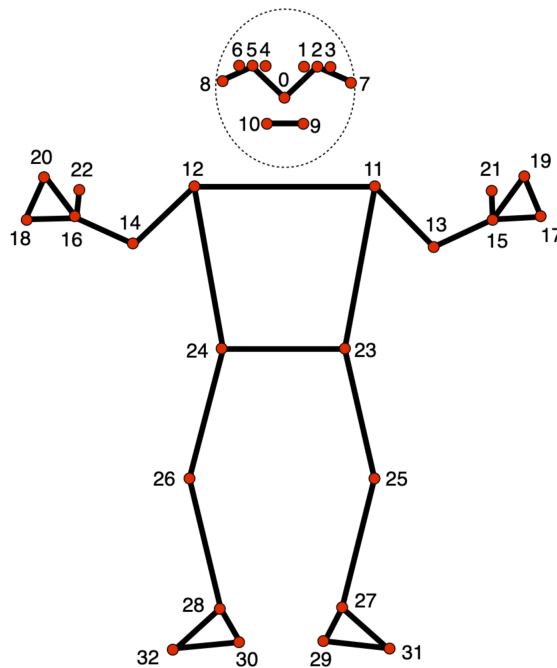


Figure 1.6: Illustration all 33 keypoints estimated by BlazePose. Figure from [29].

1.2.2 Bottom-up approach

One of the most recent advances in the field of bottom-up methods is the Pose Estimation Via Disentangled Keypoint Regression (DEKR) [50] published in 2021.

It uses HRNet as its backbone to estimate all the keypoints in the image. Then, the keypoints are grouped using disentangled representation learning. This essentially means that every keypoint type, e.g. left knee, right knee, left hip, etc., is regressed independently on the others which improves the quality of the regression. [50]

A major problem for bottom-up methods is correctly estimating smaller person, i.e. person far from the camera. HigherHRNet [8], which is based on the HRNet architecture, aims to mitigate this problem by pairing HRNet with several deconvolution modules to generate multi-resolution heatmaps. Generally, HRNet heatmaps are generated at $\frac{1}{4}$ of the original image resolution but the modifications proposed by HigherHRNet the generation of up to two times higher resolution heatmaps than regular HRNet. An illustration of the HigherHRNet architecture can be seen in Figure 1.7. After the keypoints are estimated, the Associative embedding [4] method is used for grouping the keypoints.

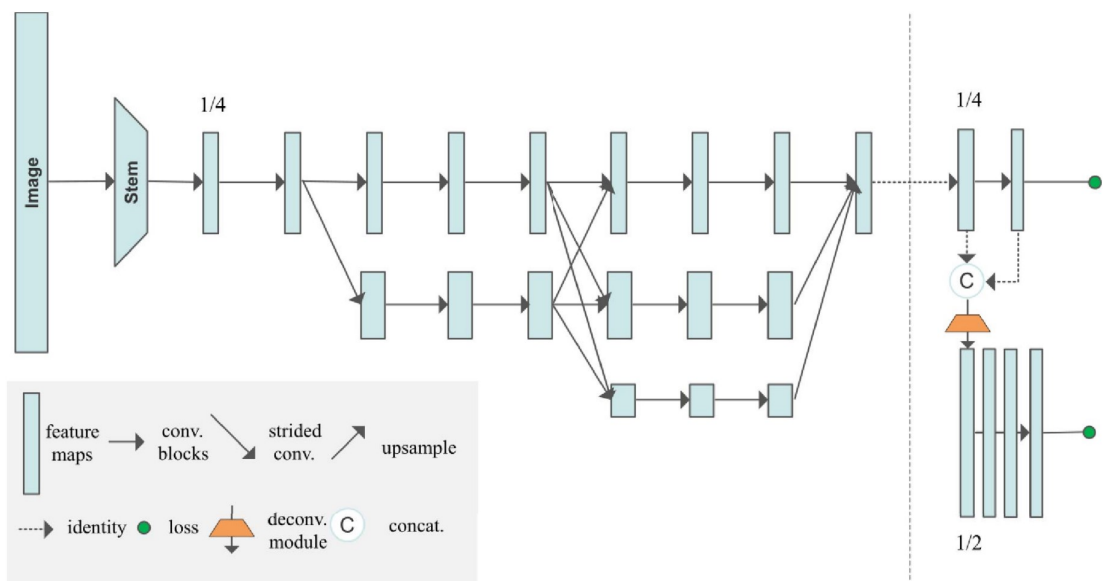


Figure 1.7: Illustration of the HigherHRNet network architecture. Figure from [8].

Another method from the HRNet-based family is LitePose [45] which proposes a more efficient neural network model with better performance on low power devices. The model is derived from HigherHRNet multi-branch architecture by gradual shrinking and removing redundant branches, thus resulting in a single-branch architecture. The shrinking process can be seen in Figure 1.8. This architecture shows better performance while retaining high precision and scale-awareness [45]. Same as HigherHRNet, LitePose uses the Associative embedding [4] for keypoint grouping.

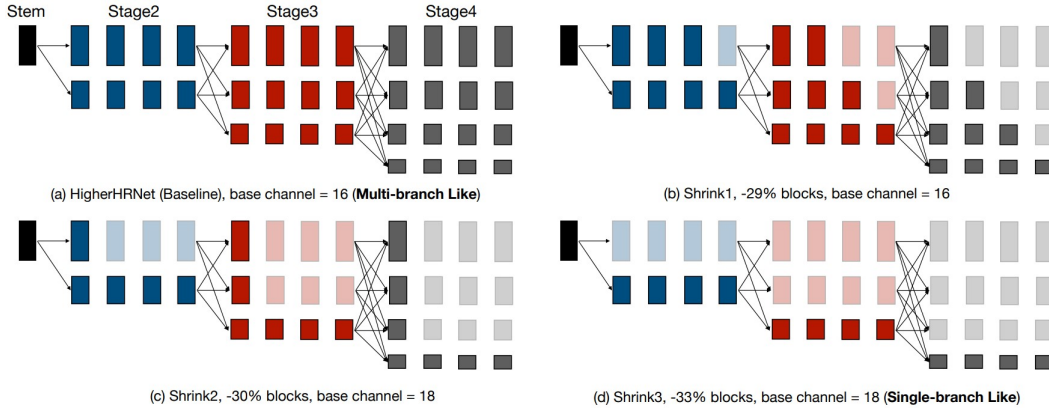


Figure 1.8: Illustration of the shrinking process from HigherHRNet architecture to LitePose architecture, transparent blocks indicate removed blocks. Figure from [45].

1.3 Evaluation metrics

Many metrics for evaluating the precision of pose estimation have been proposed over the years, but AP, short for average precision, is the most widely adopted mainly thanks to MS COCO which uses AP as its default metric. Average precision is presented in 5 standard variations: AP, AP^{.50}, AP^{.75}, AP^{medium} and AP^{large}.

The average precision metric is based on Object Keypoint Similarity (OKS) which is a value between 0 and 1 describing the accuracy of the prediction. OKS = 1 means a perfectly estimated pose and OKS = 0 marks a very poor estimation. OKS is defined as:

$$\text{OKS} = \frac{\sum_i \exp\left(\frac{-d_i}{2s^2k_i^2}\right) \cdot \delta(v_i > 0)}{\sum_i \delta(v_i > 0)}. \quad (1.1)$$

Here, the sums are over all annotated keypoints of the person, d_i denotes the Euclidean distance between a keypoint and its corresponding ground truth location, s is the object’s scale defined as the square root of the object’s area, k_i is a constant that controls falloff [24] with a separate value for each keypoint type (shoulders, knees, wrists, etc.) and lastly, v_i denotes a visibility flag of the ground truth keypoint. The visibility flag has a value $v_i = 0$ for keypoints that are not annotated, $v_i = 1$ for keypoints that are annotated but not visible and $v_i = 2$ for keypoints which are annotated as well as visible.

Afterwards, the pose is correctly estimated if OKS is above a certain threshold. The value of AP^{OKS_{thr}} can be computed as:

$$\text{AP}^{\text{OKS}_{\text{thr}}} = \text{mean}(X_i \geq \text{OKS}_{\text{thr}}) \quad (1.2)$$

where the mean is over all images and X_i denotes OKS for the image i . This means that AP^{.50} is AP^{.50} = mean($X_i \geq 0.50$) which is considered as a loose metric and analogously, AP^{.75} is AP^{.75} = mean($X_i \geq 0.75$) which in turn is a strict metric. The general AP metric is then calculated as a mean of AP^{OKS_{thr}} at OKS_{thr} = 0.50, 0.55, ..., 0.90, 0.95. The last two metrics are AP^{medium} and AP^{large}

which are defined as the AP metric but only for objects with $32^2 < s < 96^2$ and $s > 96^2$, respectively.

Besides AP, other pose estimation metrics are used by different datasets. For example, metric PCKh considers a keypoints estimated correctly, if its distance from the ground truth is no more than 50% of the head segment. This metric is used in the MPII dataset. [28]

Chapter 2

Pose tracking

Human pose tracking has gained significant attention in the recent years due to the advancements in deep neural networks. The goal of pose tracking is to correctly estimate all persons on the frame and assign a unique tag which marks the same person across multiple frames. Given a frame from the input video, a pose estimation method is used to detect all persons in the frame and estimate their poses. There are some tracking methods which use only the bounding box of a person for tracking, thus eliminating the need for pose estimation. The newly detected bounding boxes or poses are then compared against the ones on the previous frame or multiple frames and all persons are assigned an identification tracking number so that every ID is consistently assigned to the same person in all frames. An example of the pose tracking task is shown in Figure 2.1.

Similar to pose estimation, several challenges and problems in pose tracking remain to be solved. For instance, one of the biggest challenges is handling occlusion between multiple people or with the environment. This usually happens in very crowded scenes where people are overlapped most of the time and can frequently disappear completely from the camera's vision. Another problem is motion blur or any sort of distortion (e.g., from camera perspective, change of viewpoint or focal point, etc.) where a person can appear correctly on one frame, blurred or distorted on the second and correctly again on the third one. Lastly, since pose tracking relies on person detection and or pose estimation, the hardware needed for achieving real-time performance is really high.

The utilization of pose tracking is very similar to pose estimation, but allows the use in multi-person scenes and crowded scenes. For instance, pose tracking in motion-capture could be used to film scenes with multiple actors whereas without it, the results would need a lot of manual post-processing to ensure the poses correspond with the actors. In security, pose tracking could enhance the ability to recognize and predict an illegal activity by providing temporal data about all the individuals on the camera feed.

Multi-person human pose tracking methods can be generalized into two categories: bottom-up and top-down. Top-down methods works in sequence with person detection and pose estimation. First, all persons are detected using an object detector,

then a pose is estimated for all detected persons and lastly, the tracking algorithm is used to correctly assign tracking IDs.



Figure 2.1: Example of the pose tracking task. Each color indicates a unique tracking ID.

On the other hand, bottom-up methods first detect all the keypoints and then a pair of graphs is constructed using these keypoints. One of the graphs is a spatial one to ensure the keypoints get correctly grouped to each person and the other one is temporal which handles the correct assignment of tracking IDs. The final grouping is then acquired by solving an optimization problem on these graphs.

The current state-of-the-art methods use are designed by the top-down approach [35]. An example of a top-down method is PoseFlow [46] which uses a short sequence of frames to associate poses that belong to the same person. Redundant associations are then reduced by a non-maximum suppression technique. LightTrack [12] is another top-down method which proposes a way to re-ID a person when they disappear from the frame and then reappear. If a person is lost from a frame, the method calls the detection module which can associate the lost target to the poses from the previous frame via pose matching. Along with the pose estimation in AlphaPose [13], which was described in Chapter 1, was proposed a method for pose tracking that also adopts the re-ID feature and additionally implements a Pose-Guided Attention Mechanism for more accurate person identity features. A novel approach named Pose-Guided Pose Tracking [35] was introduced in 2021 uses pose information to enhance both the human detection and the human association. One of the most

recent additions to pose tracking is ByteTrack [47] which was tested in this thesis and is described in Chapter 2.2.

The bottom-up approach is less frequent due to its time complexity and memory inefficiency [13], but some methods follow this approach. PoseTrack [41] is one of the first methods which was designed for multi-person tracking. It uses integer linear programming to find the solution of the optimization problem on the temporal and spatial graphs. Another example is JointFlow [2] which uses some pose features from pose estimation in frame $i - 1$ to predict the location of a keypoint in frame i . Then, the tracking is done by creating a bipartite graph from the estimated poses and solving the bipartite matching problem with the predicted keypoints as a similarity measure.

2.1 Datasets

2.1.1 MOT17

MOT17 is an updated version of the MOT16 challenge dataset [3] which designed as a benchmark for tracking various classes of objects, for example pedestrians, cars, motorcycles, etc. The dataset is composed of 7 videos in the training set and 7 in the testing set but because the ground truth is available only for the testing set, the results in the following chapter are achieved on the training set. Video sequences in the MOT17 are all captured in different environments, from different viewpoints, with different lighting and with different amounts of crowding. Examples of 4 frames from the videos in the MOT17 training set are on Figure 2.2.



Figure 2.2: Examples of 4 frames from the different videos from the MOT17 dataset.

2.1.2 MOT20

MOT20 [33] is a dataset designed as a benchmark for multi-object tracking in crowded scenes. The dataset includes 8 videos in total, 4 in the training set and 4 in the testing set. Similarly to MOT17, all objects are annotated as one of several classes, such as pedestrians, cars, motorcycles, etc., and the ground truth is only available for the testing set. MOT20 video sequences are all captured from a high viewpoint with a very crowded scenario to evaluate state-of-the-art tracking methods on extremely crowded scenes. Examples of the frames from the videos in MOT20 training set are on Figure 2.3.



Figure 2.3: Examples of the 4 frames from the different videos from the MOT20 dataset.

2.1.3 Created dataset

As a part of this thesis, a custom dataset was created for the purpose of pose estimation and pose tracking visualization. The dataset is composed of 18 unannotated videos from YouTube [48] whose list can be found in Appendix A. These videos range from a few seconds in length up to 5 or 6 minutes in some cases and contain a few different environments, specifically an airport, mall, street and a playing field. Videos were also chosen so that they were filmed from different viewpoints, e.g. stationary and moving camera, top-down and eye-level height, stationary and moving

drone shot, and in different lighting conditions, e.g. outdoors, indoors, night, sunset, low light. Moreover, the dataset was constructed to include videos with different levels of crowding and occlusion. The levels of crowding in each video cannot be exactly determined thanks the absence of annotations, thus the crowding levels shown in table A.1 are estimated by the eye. Examples of the videos in the created dataset are on Figure 2.4.



Figure 2.4: Examples of 6 frames from the videos in the created dataset.

2.2 Methods

2.2.1 ByteTrack

ByteTrack [47] was proposed in 2022 and introduced a simple yet effective method for associating detections. First, the person detector is used to predict the bounding boxes with confidence score for every person in the frame. ByteTrack uses the

YOLOX [51] person detector which is already integrated in the ByteTrack code. The bounding boxes are then separated into low confidence and high confidence bounding boxes. Secondly, new locations are predicted for all trackers from the previous frame, i.e. the persons that are tracked. Next, the Hungarian Algorithm [14] is adopted to match the high confidence bounding boxes to the predicted locations and the unmatched high confidence detections are used to initialize new trackers. The same algorithm is then used to match the low confidence bounding boxes with the unmatched trackers. Unmatched low confidence detections are discarded and unmatched trackers are considered lost. These lost trackers are then merged with the rest of the trackers while keeping track of how many times in a row was each tracker considered lost. If a tracker was lost for a set number of frames, the tracker is discarded. The pose estimation is done after the tracking since ByteTrack only needs the bounding boxes for tracking.

2.2.2 QDTrack

QDTrack [19] proposes a novel learning approach named *quasi-dense similarity learning*, which is used to train a neural network, that can acquire the feature embeddings for each image. These feature embeddings are then used in object association to calculate the similarity between an object and its matching candidates using a bi-directional softmax method. The object is associated to the candidate with the highest similarity. If an object has been newly detected and has a high detection confidence score, it is assigned to a new track. In contrary to other methods, QDTrack does not drop the detected objects that do not match any track, but cannot create a new track. These objects are kept during object matching, which reduces the number of false positives. [19] In case of duplicate detections, the redundant detections are discarded by non-maximum-suppression.

2.2.3 Greedy approach

A tracking method based on the greedy approach has been implemented in three variants, which all differ by the similarity metric used. Greedy tracking first uses the person detector to estimate the bounding boxes in the frame. Since this thesis is on pose estimation and tracking, multiple person detector have been tested beforehand with the Faster-RCNN [38] detector showing the best detection accuracy while achieving reasonable speed. Then, the poses are estimated followed by the tracking. For each of the results from pose estimation, a similarity metric is calculated between it and all the tracked people from the previous frame. If the metric is lower or higher than a given threshold, the new pose and the tracked pose are considered the same person and the tracked ID is assigned to the new pose.

In this thesis, three different metrics are used for the pose matching. First one is Intersection over Union (IoU). For bounding boxes A and B , the metric is calculated as follows:

$$\text{IoU}(A, B) = \frac{S(A \cap B)}{S(A \cup B)}, \quad (2.1)$$

where $S(A \cap B)$ and $S(A \cup B)$ denotes the area of the intersection of A and B and area the union of A and B , respectively. If the maximal IoU between a pose from the current frame and all tracked poses from the previous is higher than the given threshold, the new pose gets assigned the track ID of the pose which had the highest IoU with. In case the IoU is lower than the threshold, the pose gets assigned a new track ID if it consists of more than 3 keypoints.

The other two metrics are keypoint-based, meaning they are calculated directly from the keypoints themselves unlike IoU. One of them is the OKS which has been described in Chapter 1 and is calculated by Eq. 2.2.

$$\text{OKS} = \frac{\sum_i \exp\left(\frac{-d_i}{2s^2k_i^2}\right) \cdot \delta(v_i > 0)}{\sum_i \delta(v_i > 0)} \quad (2.2)$$

The last used metric is the Euclidean distance which is calculated for all pairs of keypoints between the new pose and the pose from the previous frame.

$$d(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \quad (2.3)$$

Here, $\vec{x} = (x_1, x_2)$ and $\vec{y} = (y_1, y_2)$ denotes two keypoints with their corresponding coordinates in the image. The distances between all pairs of the pose are then added together and if the overall distance is lower than the given threshold, the new pose gets assigned the track ID of the pose which has the lowest overall distance. Same as with the other two metrics, if the overall distance is higher than the threshold, the pose gets assigned a new track ID if it consists of more than 3 keypoints.

2.2.4 OC-SORT

In 2023, a tracking method named Observation-Centric SORT (OC-SORT) [15] was introduced which builds upon the previously proposed Simple Online and Real-time Tracking (SORT) [1] method. The original SORT algorithm was introduced in 2016 and works similarly to tracking by the greedy approach with the exception, that SORT does not calculate IoU between the tested bounding box on the current frame with the bounding box from the previous frame. Rather, it predicts where the bounding box from the previous frame should be and calculates IoU between the tested and the predicted bounding boxes. The prediction is based on the velocity of the bounding box calculated from several previous frames. When new object is detected, a new track is created with zero velocity and when an object is lost, the track remains active for a number of frames before being discarded in case of the object reappearing. With every frame that the track is not matched to an object, it deviates from the correct position and when the track is matched again, it is likely to be lost again due to the temporal error accumulated from these predictions. OC-SORT aims to negate this problem by backtracking through the untracked period and updating the prediction parameters, which then provides more accurate bounding box predictions and lowers the probability of the track being lost again. OC-SORT further improves the accuracy by implementing a second attempt on associating the detections to the lost tracks after the initial association. This can handle an object that is stopping or being occluded for a short amount of time [15].

2.3 Evaluation metrics

In the long history of multi-object tracking, many evaluation metrics for human pose estimation have been introduced. Currently, the most widely adopted metric is the CLEAR MOT [21] which was proposed in 2008. CLEAR MOT uses multiple object tracking accuracy (MOTA) as the main metric and a number of support metrics with multiple object tracking precision (MOTP) being the most important of them. MOTA was also widely criticized as it has many flaws and problems which are described below. In order to mitigate these problems, a metric called higher order tracking accuracy (HOTA) [18] was introduced in 2020 and was quickly adopted by many of the widely used benchmarks. There are other metrics which are currently used by some benchmarks, but the majority now uses MOTA and HOTA. For example, IDF1 [11] was originally proposed for use in multi-camera multi-object tracking and thanks to its emphasis on more accurate association over detection, it has been used to evaluate a number of single-camera tracking methods. Another example is Track mAP [27], but it differs from the other mentioned metrics by requiring the confidence score of all trackers which most methods are not providing.

2.3.1 MOTA

The calculation of the MOTA metric is done by using only the detected bounding boxes. Each detected bounding box is paired with its corresponding ground truth bounding box. The pairing is determined by their spatial similarity \mathcal{S} , e.g. IoU, and two bounding boxes are allowed to be paired if \mathcal{S} is higher than threshold α . If a detected bounding box becomes paired with a ground truth bounding box, then it is correctly detected and a true positive (TP). If a detected bounding box is unpaired, it is a wrong detection and a false positive (FP). Similarly, if ground truth bounding box is unpaired, then it is a missing detection and a false negative (FN). The association part of MOTA is handled by the so-called identity switch (IDSW). An IDSW is a TP which has a different prediction ID than in the previous frame, but has the same ground truth ID on both frames. This means, that an IDSW happens when the tracking method wrongly swaps person IDs or when a person is lost and reappeared again but with different IDs. The final value of the MOTA metric is calculated as follows:

$$\text{MOTA} = \frac{|\text{FN}| + |\text{FP}| + |\text{IDSW}|}{|\text{grDet}|}, \quad (2.4)$$

where $|\text{FN}|$, $|\text{FP}|$ and $|\text{IDSW}|$ denotes the number of false negatives, false positives and identity switches, respectively. $|\text{grDet}|$ denotes the total number of ground truth detections. The values appearing in Eq. 2.4 only have the information about correct and incorrect detections which is why MOTA works in conjunction with a secondary metric MOTP to provide a measure of the localization error. MOTP is acquired by:

$$\text{MOTP} = \frac{1}{|\text{TP}|} \sum_{TP} \mathcal{S}. \quad (2.5)$$

As previously mentioned, MOTA has been the main metric for evaluating multi-object tracking methods since 2006, but it has several problems which can impact the research going forward. The main drawback of MOTA is the bias towards detection. This is because the detection errors are measured in false positives and false negatives while association errors are measured only in identity switches. Therefore, the ratio of the effect that detection and association has on the final score is:

$$\frac{|FN| + |FP|}{|IDSW|}.$$

According to [18], this ratio is 98.6 on average for the top 10 tracking methods on the MOT17 benchmark. This means, that on average the accuracy of the detection has a 100 times larger impact on the final score than the accuracy of the association.

Another big issue is that MOTA ignores ID transfers. ID transfers occur when a TP has a different ground truth ID than on the previous frame but the detection ID is the same both frames. This is illustrated in Figure 2.5 on a two frame video. Figure 2.5a shows one object on both frames, but on the second frame it is incorrectly tracked, therefore an IDSW occurs and MOTA reports the value of 0.5. In Figure 2.5b, there are two objects, each one visible on a different frame, and the tracking method predicts that they are the same object. Since the predicted ID remains the same, an IDSW did not occur and MOTA reports a perfect 1.0 despite the incorrectly tracked object on the second frame.

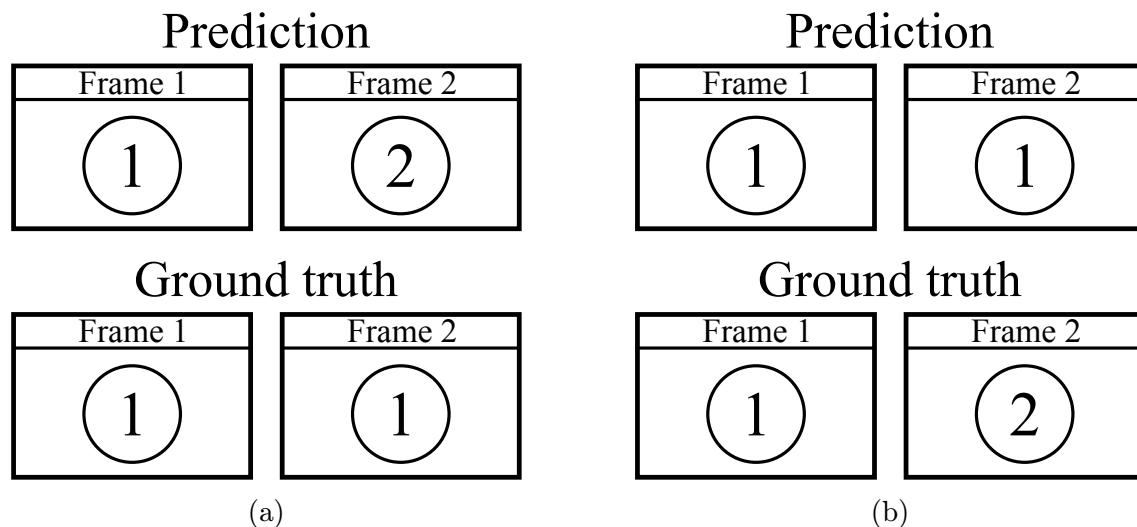


Figure 2.5: Illustration of an ID transfer, circle represents an object, number inside of the circle represents the object’s ID.

MOTA also does not reward mistake corrections which is another problem. This is illustrated in Figure 2.6. In scenario A, the tracking method made a wrong change in tracking but corrected it’s mistake. In B, the mistake is not corrected and in C, the mistake is made worse by wrongly changing the ID again. The MOTA scores for these scenarios should logically be $A > B > C$, yet MOTA shows very different results.

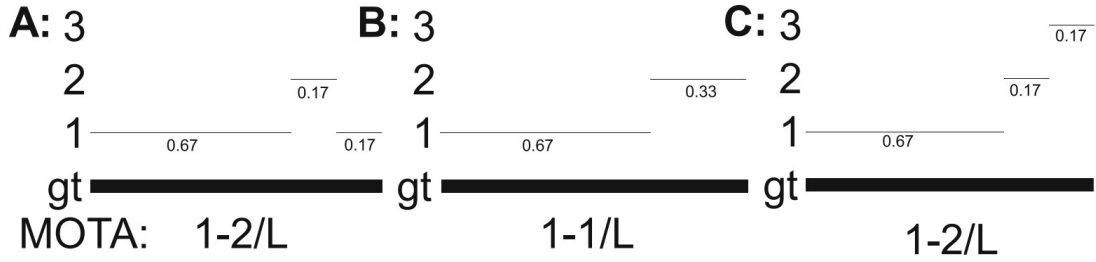


Figure 2.6: Illustration of the mistake correction problem with MOTA, L denotes the number of ground truth detections. Figure from [18].

Similarly to mistake correction, MOTA also does not reward greater alignment with ground truth. An illustration of this problem is shown in Figure 2.7. Here, the scenarios A, B and C are correctly tracked for 50%, 67% and 83% of the time, respectively. However, MOTA reports the same value for all scenarios despite C being the best one.

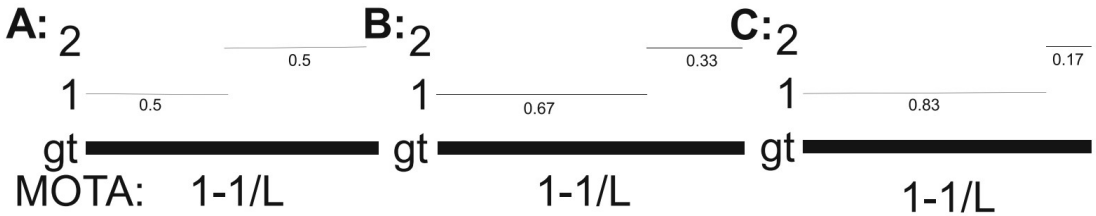


Figure 2.7: Illustration of the ground truth alignment problem with MOTA, L denotes the number of ground truth detections. Figure from [18].

Lastly, there are a few minor issues that could cause problem in some applications. First one is that MOTA is highly dependent on the frame rate of the video. For example, if one IDSW happens on a 2.5 second video running at 40 fps, then the MOTA value would be 0.99. If the same video would be processed at 4 fps with one IDSW, then MOTA would be only 0.9. Next is the need for an additional metric MOTP to handle the localization error and not being able to measure detection, association and localization in a single metric. Last one is the fact that the value of MOTA can be negative which leads to the problem of negative MOTA interpretation.

2.3.2 HOTA

To combat all of these problems, the HOTA [18] metric was proposed and quickly adopted by many of the popular benchmarks. The process of matching the predictions with the ground truth is the same as with MOTA, but where HOTA differs from MOTA is in the handling of the association. HOTA introduced the concept of the true positive associations (TPA), false positive associations (FPA) and false negative associations (FNA). These are defined for a given TP (denoted as c) as follows:

- $\text{TPA}(c)$ is a set of other TPs with the same prediction ID and ground truth ID as c .
- $\text{FNA}(c)$ is a set of ground truth bounding boxes with the same ID as c , but with either different or missing prediction ID.
- $\text{FPA}(c)$ is a set of predicted bounding boxes with the same ID as c , but with either different or missing ground truth ID.

Next, the HOTA_α value is calculated as follows:

$$\text{HOTA}_\alpha = \sqrt{\frac{\sum_{c \in \{\text{TP}\}} \mathcal{A}(c)}{|\text{TP}| + |\text{FN}| + |\text{FP}|}}, \quad (2.6)$$

where α is the threshold used for prediction-ground truth matching and $\mathcal{A}(c)$ is defined as:

$$\mathcal{A}(c) = \frac{|\text{TPA}(c)|}{|\text{TPA}(c)| + |\text{FNA}(c)| + |\text{FPA}(c)|}. \quad (2.7)$$

The final HOTA score is then defined as an integral of HOTA_α over all valid values of α :

$$\text{HOTA} = \int_0^1 \text{HOTA}_\alpha \, d\alpha \quad (2.8)$$

In real applications, this value is approximated by averaging HOTA_α over α values between 0.05 and 0.95 with 0.05 intervals.

HOTA provides a single metric that equally measures detection, association and localization in one value, but it may be beneficial to have access to the individual scores for detection, association and localization. Detection and association metrics are first defined with the dependency on α and their final scores are calculated as an average over α , same as HOTA.

$$\text{DetA}_\alpha = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}| + |\text{FP}|}. \quad (2.9)$$

$$\text{AssA}_\alpha = \frac{1}{|\text{TP}|} \sum_{c \in \{\text{TP}\}} \mathcal{A}(c). \quad (2.10)$$

The localization metric can be measured separately as:

$$\text{LocA} = \int_0^1 \frac{1}{|\text{TP}|_\alpha} \sum_{c \in \{\text{TP}_\alpha\}} \mathcal{S}(c) \, d\alpha, \quad (2.11)$$

where $\mathcal{S}(c)$ is the spatial similarity between the predicted bounding box and the ground truth box of the TP c .

Chapter 3

Experiments

In this chapter, results of the introduced methods are presented in the aforementioned metrics along with their comparison and the discussion about their performance.

Pose estimation methods were evaluated on the MS COCO and CrowdPose datasets which were described in Chapter 1.1. For comparison between methods, the evaluation metrics described in Chapter 1.3, i.e., AP, AP⁵⁰, AP⁷⁵, AP^{medium} and AP^{large}, are used. The speed of pose estimation methods was measured in iterations per second ($\frac{\text{it}}{\text{s}}$).

Pose tracking method evaluation was done on the MOT17 and MOT20 datasets. These datasets were described in Chapter 2.1 along with the created dataset which is unannotated and is used for visual evaluation only. For each method, the HOTA, DetA, AssA, MOTA and MOTP tracking metrics, described in Chapter 2.3, are calculated and frames per second (fps) are used for speed comparison.

In order to minimize the influence of external factors on the results, all methods were tested on the same system and under the same conditions, e.g. other programs running in the background. Hardware used for all tests is shown in Table 3.1. Lastly, all methods were used with the pretrained models that are freely available for download by the authors of their corresponding method.

Processor	AMD Ryzen 3600
Graphics card	NVIDIA RTX 2080
System memory	16 GB at 3200 MHz
Storage	Samsung 970 EVO

Table 3.1: Computer hardware used for the calculations.

3.1 Pose estimation

All top-down pose estimation methods were used with the YOLOv3 [20] person detector implemented in the package MMDetection [22]. It should be noted, that other detector were tested and some resulted in better pose estimation accuracy. YOLOv3 was chosen to in order to offer a fair comparison between the methods as AlphaPose has YOLOv3 integrated in itself. The only exception is MediaPipe that uses their own proprietary person detector. For implementations of the methods, a package MMPose [34] was used. This package contains the implementations of over 20 different pose estimation methods such as HRNet, Stacked hourglass, DEKR or HigherHRNet which were all tested. AlphaPose and LitePose were downloaded and installed from their respective official repositories.

3.1.1 MS COCO

Since the MS COCO dataset was originally intended for object detection, some of the images do not contain people and the images that do mostly involve only a few people with little to no occlusion. For this reason, MediaPipe was tested even though it is a method designed only for single-person pose estimation. Moreover, MediaPipe also differs from the other methods by running only on the CPU.

The results for each of the methods on the MS COCO dataset are shown in Table 3.2.

Method	AP	AP ^{.50}	AP ^{.75}	AP ^{medium}	AP ^{large}	Speed [$\frac{it}{s}$]
HRNet*	0.666	0.796	0.729	0.629	0.731	16.10
AlphaPose	0.722	0.886	0.799	0.682	0.785	16.28
Hourglass*	0.648	0.795	0.708	0.612	0.708	14.83
MediaPipe	0.452	0.777	0.482	0.386	0.476	14.98
DEKR*	0.709	0.876	0.773	0.666	0.783	4.14
HigherHRNet*	0.672	0.864	0.731	0.617	0.762	2.07
LitePose	0.624	0.825	0.679	0.548	0.737	10.58

Table 3.2: Human pose estimation results on COCO val2017 set. * denotes methods implemented in MMPose [34].

AlphaPose shows the overall best performance across all measured metrics with DEKR achieving slightly lower but still comparable results. The rest of the top-down methods, HRNet and Stacked hourglass, perform marginally worse than their tested bottom-up counterparts with only LitePose and MediaPipe bellow them.

However, this was to be expected as LitePose is designed to be less hardware demanding which in turn reduces the accuracy of the estimation. This compromise

has evidently paid off as LitePose achieves more than double the speed of DEKR and more than five times the speed of HigherHRNet. The lower scores of MediaPipe were also expected as it functions only as a single-person pose estimator. But when only the loose metric AP^{.50} is considered, MediaPipe is performing notably better and achieves scores just 0.02 AP^{.50} lower than HRNet or Stacked hourglass.

In terms of inference speed, the top-down methods achieve significantly better performance than most tested bottom-up methods with the exception of LitePose as mentioned above. The highest speed performance was achieved by AlphaPose followed closely by HRNet which both perform 4-8 times faster than DEKR and HigherHRNet. This massive speed difference is present in spite of the fact, that work in sequence by detecting all persons and then estimating the poses one at a time. The parallel approach of bottom-up methods, where the keypoints are detected all at once and then grouped, should in theory be faster, yet the current state-of-the-art bottom-up methods are not able to match the speed of top-down methods without sacrificing too much accuracy.

Based on these results, the HRNet method was chosen to handle the pose estimation in the pose tracking task, thanks to it’s speed to accuracy ratio and the implementation in MMPose which made the integration with MMTracking easier.

3.1.2 CrowdPose

As stated before, MediaPipe is designed for single-person estimation only. This issue was not that severe on MS COCO, but since CrowdPose places high emphasis on multi-person pose estimation and high occlusion scenarios, MediaPipe was not tested. Because of the high amounts of occlusion in the images, many of the detected people do not fall under the area restrictions for AP^{medium} and AP^{large} metrics, which is why the CrowdPose dataset does not measure these metrics. The results for the tested methods on the CrowdPose dataset are shown in Table 3.3.

Method	AP	AP ^{.50}	AP ^{.75}	Speed [$\frac{\text{it}}{\text{s}}$]
HRNet*	0.651	0.795	0.701	6.24
AlphaPose	0.695	0.853	0.749	11.79
Hourglass*	0.464	0.671	0.492	7.52
DEKR*	0.682	0.869	0.736	4.24
HigherHRNet*	0.653	0.860	0.700	1.21
LitePose	0.605	0.831	0.645	8.93

Table 3.3: Human pose estimation results on CrowdPose test set. * denotes methods implemented in MMPose [34].

Results on CrowdPose present a similar trend as on MS COCO with AlphaPose achieving the highest accuracy of all tested methods, followed by the two bottom-up

methods DEKR and HigherHRNet. Important to note, that all methods suffered a performance decrease compared to MS COCO, both in accuracy and speed. This was expected as CrowdPose contains a higher number of people in general as well as on average in a given image. The accuracy decrease with this transition to more multi-person scenarios is relatively small at roughly 0.02 for all methods except Stacked hourglass, which suffered the biggest performance decrease by far at $\sim 29\%$ lower than its AP on MS COCO.

The speed measurements also provide an interesting conclusion that the speed of bottom-up methods is less dependent on the number of persons in the image. For example, HigherHRNet suffered a $\sim 42\%$ speed decrease, LitePose only $\sim 16\%$ decrease and DEKR even achieved marginally faster performance while top-down methods HRNet, AlphaPose and Stacked hourglass suffered a speed decrease of $\sim 61\%$, $\sim 38\%$ and $\sim 50\%$, respectively.

These results confirmed the decision to use HRNet for pose tracking tests as it had the smallest accuracy decrease compared to results on MS COCO while still achieving reasonable speed.

3.2 Pose tracking

The implementations of ByteTrack, QDTrack and OC-SORT were used from a Python package MMTracking [30] and they come already integrated with person detectors. ByteTrack and OC-SORT use the YOLOX [51] person detector, while QDTrack uses the Faster-RCNN [38] detector. The greedy tracking methods were also used in conjunction with the Faster-RCNN person detector implemented in MMDetection [22]. Implementation of the greedy tracking by IoU and keypoint distance was done by the author of this work while greedy tracking by OKS was used from MMPose [34]. All methods were run with pose estimation enabled and the HRNet [24] from MMPose was used.

3.2.1 MOT17 and MOT20

The results for each of the tracking methods on the MOT17 and MOT20 datasets are shown in Table 3.4 and Table 3.5, respectively.

As can be seen for Tables 3.4 and 3.5, ByteTrack and OC-SORT achieve very similar performance with OC-SORT having very minor accuracy lead. The greedy approaches showed the worst accuracy on both datasets, which can be explained by the absence of a re-identification feature to recover people which momentarily disappear from the camera’s vision due to occlusion or environment obstacles. This problem becomes even more apparent on the MOT20 dataset where the videos contain larger amounts of people with higher levels of occlusion than any video in the MOT17 dataset. On the other hand, the achieved inference speed of greedy tracking approaches was higher than ByteTrack and OC-SORT. QDTrack falls in the middle

of the tested methods, achieving comparable or higher inference speed than greedy tracking while performing significantly more accurate.

Method	HOTA	DetA	AssA	MOTA	MOTP	Speed [fps]
ByteTrack	72.220	75.816	69.189	85.925	88.166	2.89
OC-SORT	72.609	75.615	70.116	85.247	88.210	2.81
QDTrack	63.912	67.230	61.145	78.500	84.799	3.84
Greedy by IoU	37.424	40.833	35.036	39.214	78.627	3.76
Greedy by distance	31.017	40.888	24.304	36.469	78.657	3.69
Greedy by OKS	33.976	40.928	28.879	37.803	78.635	3.87

Table 3.4: Results for pose tracking on the MOT17 dataset.

Method	HOTA	DetA	AssA	MOTA	MOTP	Speed [fps]
ByteTrack	53.374	56.784	50.317	68.722	80.187	1.15
OC-SORT	53.981	58.011	50.410	69.251	79.874	1.03
QDTrack	29.738	34.564	25.845	43.512	72.741	1.60
Greedy by IoU	17.167	22.870	13.405	26.377	72.105	2.00
Greedy by distance	12.800	22.892	7.627	24.241	72.169	1.78
Greedy by OKS	15.785	22.916	11.418	25.933	72.131	1.95

Table 3.5: Results for pose tracking on the MOT20 dataset.

Moreover, the person detector Faster-RCNN also attributes to the lower scores of QDTrack and the greedy approach which can be seen by the DetA scores. These scores for Faster-RCNN are much lower than the scores that YOLOX managed to achieve. An effort was made to use the standalone version of YOLOX person detector in MMDetection, but the implementation failed to work even after several attempts to fix the errors. The official implementation of YOLOX was tried and was able to produce results, however, the result for IoU greedy tracking was 37.6 DetA which is worse than the 40.8 DetA achieved by Faster-RCNN. The reason for this anomaly remains unknown.

Numerous attempts were also made to alter the code of ByteTrack and OC-SORT in order to use a different person detector, like Faster-RCNN, for a fair comparison but to no avail. Thus, the results were published in this state, with ByteTrack and OC-SORT using YOLOX and the greedy tracking and QDTrack using Faster-RCNN. It is not a fair comparison of the methods by any means, but even from these results it can be concluded, that greedy approach is unusable for multi-person heavy scenarios, like the ones in MOT20.

The results for MOT17 and MOT20 datasets also show one of the biggest problems with MOTA which is the bias towards the detection score. For example, greedy tracking by IoU and by distance both have detection score of ~ 22.88 DetA on MOT20 with the association scores being 13.405 AssA and 7.627 AssA, respectively. HOTA penalized the tracking by distance for the worse association accuracy by roughly the same value as was the difference between them while MOTA penalized the distance tracking only by less than a half of the difference between the scores. This illustrates that MOTA is biased towards detection which could lead to researchers unknowingly optimizing their methods for better detection instead of association.

3.2.2 Created dataset

The tracking methods were also tested on the dataset which was created for this thesis. Because the dataset is unannotated, only a handful of frames and notable details will be included here as examples. All results on the created dataset are reproducible using the application from Chapter 4. Figures 3.3, 3.4 and 3.5 show some exemplary frame pairs for each of the methods in 3 of the 18 videos.

As previously discussed, ByteTrack, OC-SORT and QDTrack perform significantly better than the greedy tracking methods with a large portion of the performance uplift being the YOLOX person detector. For this reason, this section will more focused on the comparison between the individual greedy approaches.

From the three metrics used for pose matching in the greedy approach, the tracking by distance is performing the worst. This can be seen in Tables 3.4 and 3.5 where it achieves the lowest HOTA and MOTA scores with the poor association performance being highly noticeable on the AssA metric. The problem with this pose matching metric is, that the sum of the distances between keypoints of the same type in the current and previous frame is extremely dependent on the video. For instance, a person in a video with a low frame rate will have their keypoints further between frames when compared to a video with a higher frame rate, which leads to the need for a higher threshold for pose matching. Another example of this problem can be a video where one person is slowly walking and another is running. In this video, the person running would need a higher pose matching threshold than the person walking in order to be tracked. This would lead to the need for a pose matching threshold specifically tuned for each video and in some cases for each person.

The greedy tracking with pose matching based on the Object Keypoint Similarity (OKS) metric performed better than the distance based. The usage of this metric negates the problem with the tracking threshold being highly dependent on the properties of the video or the people in it. Reason for this is, that OKS is percent-based with a value between 0 and 1 indicating how similar are the poses in the current and previous frame. However, the usage of this metric also has a significant flaw which applies to the tracking by distance as well. The fact that the tracking is reliant on the actual keypoint locations can have a major impact on the association accuracy, since the pose estimator may have problems estimating certain poses or people in certain perspectives, see Figure 3.1. Here, a person is shown on two different

frames with a wrongly estimated pose due to the difficult perspective. The positions of the keypoints are vastly different, meaning the OKS between the two poses is lower than the threshold and the person on the newer frame is considered a new person and assigned a new ID, denoted by the change in the color. On the other hand, in Figure 3.2 the same frames are shown, but this time tracked by the IoU which tracks the person correctly despite the wrongly estimated pose.



(a) Frame 61

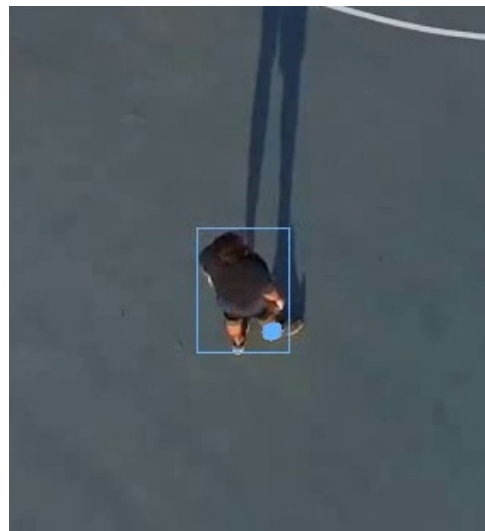


(b) Frame 62

Figure 3.1: Example of an identity switch of the OKS tracking due to wrongly estimated pose in a difficult perspective.



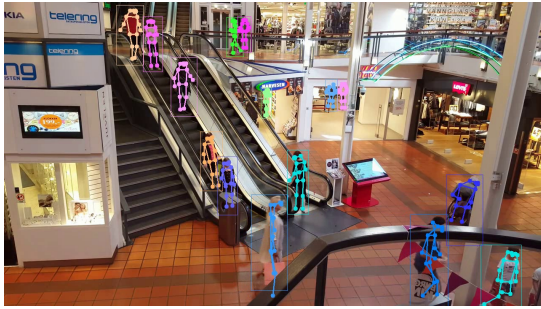
(a) Frame 61



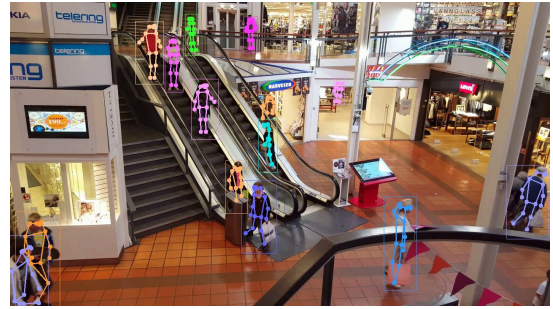
(b) Frame 62

Figure 3.2: Same frames as in Fig. 3.1, but tracked by IoU tracking.

As mentioned above, the usage of the IoU metric for greedy tracking eliminates the problem of inaccurate association with incorrectly estimated poses. This is because the IoU metric is calculated directly from the bounding boxes of the detections. The results in Tables 3.4 and 3.5 show, that using IoU as the similarity metric in greedy tracking yields the best association accuracy of the tested similarity metrics. It has also been observed, that on the videos with lower levels of crowding, the IoU greedy tracking provides fairly stable person tracking with most of the wrong ID changes being caused by the person detector or occlusion. With the integration of a better person detector and the implementation of a re-ID feature, the IoU greedy tracking could be used in scenarios where the crowding level remains low for the majority of the time, like for example a warehouse security camera.



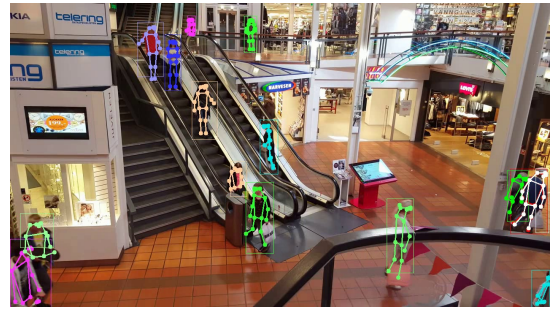
(a) Frame 90, ByteTrack



(b) Frame 150, ByteTrack



(c) Frame 90, IoU tracking



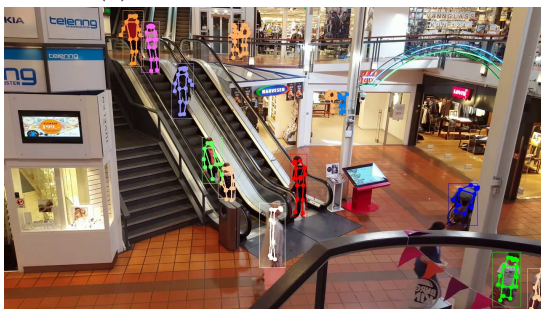
(d) Frame 150, IoU tracking



(e) Frame 90, OKS tracking



(f) Frame 150, OKS tracking



(g) Frame 90, distance tracking

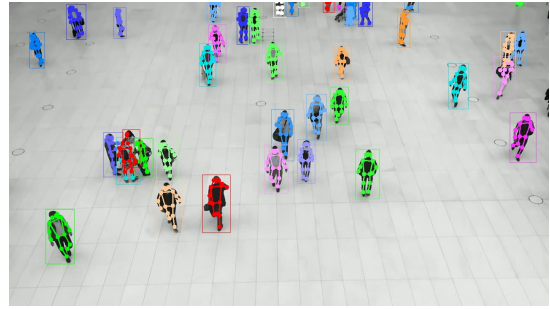


(h) Frame 90, distance tracking

Figure 3.3: Exemplary frame pairs from a video in the created dataset with different tracking methods used.



(a) Frame 90, ByteTrack



(b) Frame 150, ByteTrack



(c) Frame 90, IoU tracking



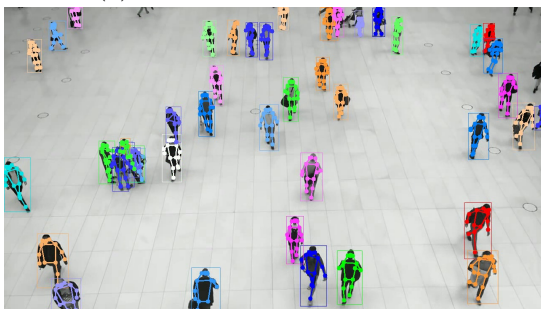
(d) Frame 150, IoU tracking



(e) Frame 90, OKS tracking



(f) Frame 150, OKS tracking



(g) Frame 90, distance tracking

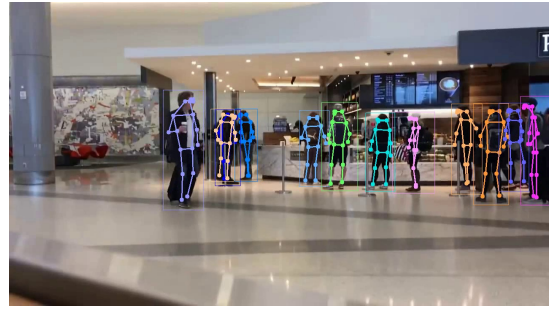


(h) Frame 150, distance tracking

Figure 3.4: Exemplary frame pairs from a video in the created dataset with different tracking methods used.



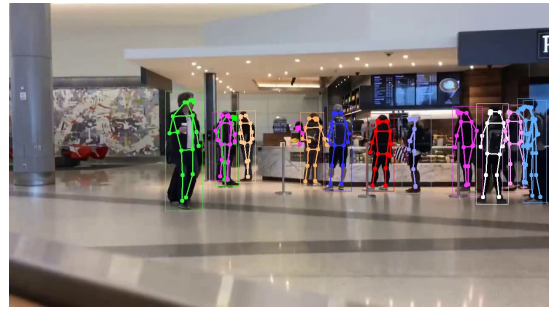
(a) Frame 90, ByteTrack



(b) Frame 150, ByteTrack



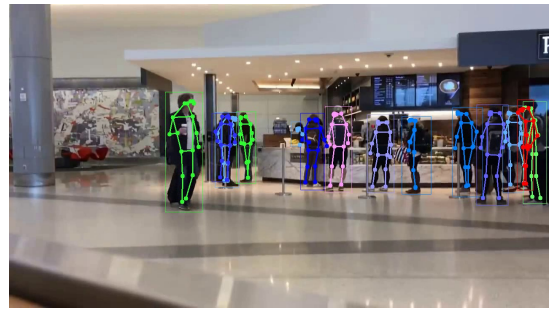
(c) Frame 90, IoU tracking



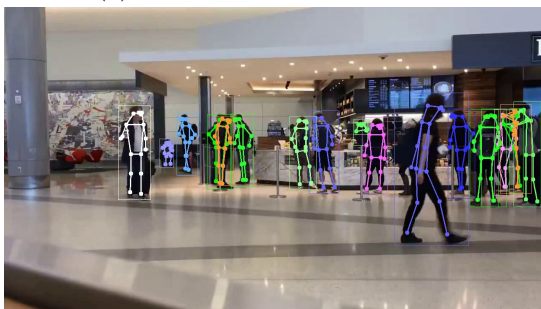
(d) Frame 150, IoU tracking



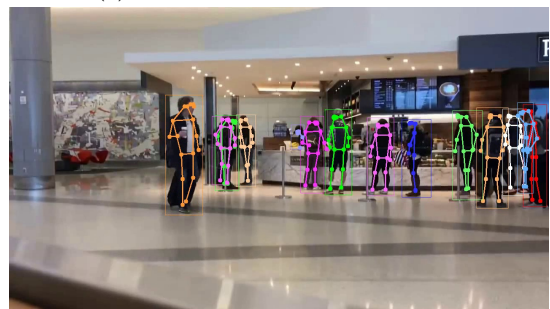
(e) Frame 90, OKS tracking



(f) Frame 150, OKS tracking



(g) Frame 90, distance tracking



(h) Frame 150, distance tracking

Figure 3.5: Exemplary frame pairs from a video in the created dataset with different tracking methods used.

Chapter 4

Application

As a part of this work, an application with a graphical user interface was designed with the aim to provide a more user friendly way to visualize some of the pose estimation and tracking methods. This chapter is focused on the implementation details and the usage of the application. The complete implementation is publicly available at <https://github.com/cechantonin/PoseEstimationApp> along with the instructions for installation.

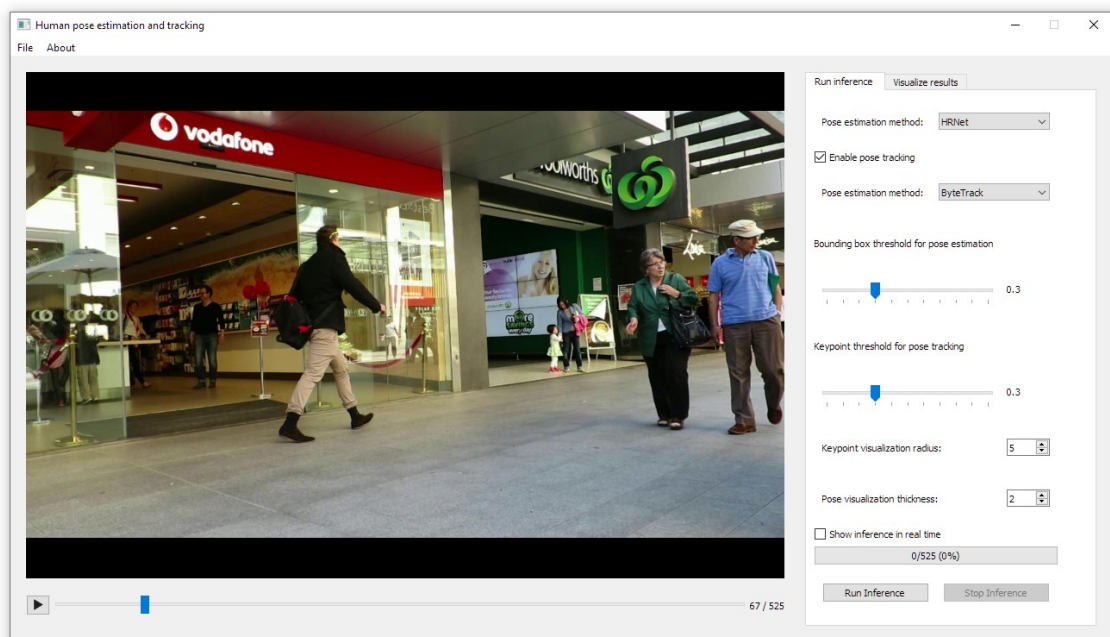


Figure 4.1: The user interface of the application.

4.1 Implementation

The application is implemented in Python, specifically Python 3.10, and the user interface is built upon the PyQt5 package. The opening of a video is handled by

the MMCV package which provides tools for image and video processing, image and annotation visualization, data transformation and convolutional neural network building. This package is also required and used by MMPose, MMDetection and MMTracking.

When a video file is selected, it is opened by MMCV and then stored in an array which is a property of the main window. This array has 4 dimensions in total. First one holds all the loaded videos with the original video at index 0 and the inference result at index 1. All other videos generated by the visualization of a result JSON file are stored in the subsequent indexes. The second dimension stores the individual frames of the selected video, next dimension stores all the columns of the frame and the last dimension stores the individual RGB values for each of the pixels in the column. This approach allows possibility to view the pose estimation in real time in the video player along with the ability to scroll through the video while the estimation is running. For result visualization, this allows the user to load multiple results estimated with different settings, have them visualized on the video and then freely switch between them without the need for any processing. However, the storage of all visualized videos comes with a heavy performance hit when it comes to memory usage. For this reason, it is highly recommended to use lower resolution videos, e.g. 720p, and not visualizing too many results at one. The maximum number of visualizations of course depends on the total amount of free memory in the system the app is running at.

The video player itself is not using the QMediaPlayer widget included in PyQt5, because it only takes the file URL as an input and it does not allow the usage of an array to play the video from. Instead, the video player is implemented from scratch with a QLabel being the main component where the frames are displayed. As mentioned before, the videos are stored in an array as individual frames and when the video is played, a timer running at the frame rate of the video is started. On every tick of the timer, a frame is converted to from the array of pixels to a QImage which then assigned to the QPixmap of the QLabel.

When the pose estimation is started, an instance of the Worker class is created and then run inside a QThreadPool. This ensures, that the user interface stays responsive during the estimation process and, as mentioned before, enables the ability to scroll through the video while running the estimation. The option to stop the estimation is implemented using a global variable defined in a separate python file which is imported into the main script and the estimation script.

Lastly, pose tracking is disabled for the two bottom-up methods DEKR and HigherHRNet due to the bottom-up methods not generating bounding boxes while ByteTrack and IoU greedy tracking need them in order to perform the tracking. Additionally, from the speed testing on CrowdPose and the achieved fps on the pose tracking datasets, it can be assumed that DEKR or HigherHRNet would achieve less than 1 fps when used for pose tracking. Thus, DEKR and HigherHRNet are implemented in the application only for pose estimation visualization. Also, it is important to mention that despite having the best results of all tested methods, AlphaPose is not implemented in the application due to package version incompatibility with MMPose.

4.2 Usage

The user interface layout can be seen in Figure 4.1. It is composed of the video player and a QTabWidget which used for swapping between the ability to run pose estimation on the video or visualize the results from previously generated JSON files.

In the QTabWidget on the right of the window are the inference settings, Figure 4.2. The pose estimation method of choice can be selected here as well as the tracking method, if pose tracking is enabled. The bounding box threshold can be set using the slider to prevent pose estimation on low confidence bounding boxes. Similarly, the keypoint tracking threshold can be set to influence the similarity pose matching. Note, that higher number means more strict tracking for ByteTrack, IoU and OKS, whereas higher number means less strict tracking for distance-based tracking. Next, the keypoint radius and line thickness can be changed in order to be more visible in the video. This visualization setting is not retroactively applicable and must be set before the start of the estimation to achieve the desired effect. Lastly, a checkbox can be checked in order to view the processed frames in real time. The user can pause the video or scroll through it while the estimation is running and the frames will be dynamically updated as the video is playing. The real time showcase of the processed frames can be enabled once again by the checkbox.

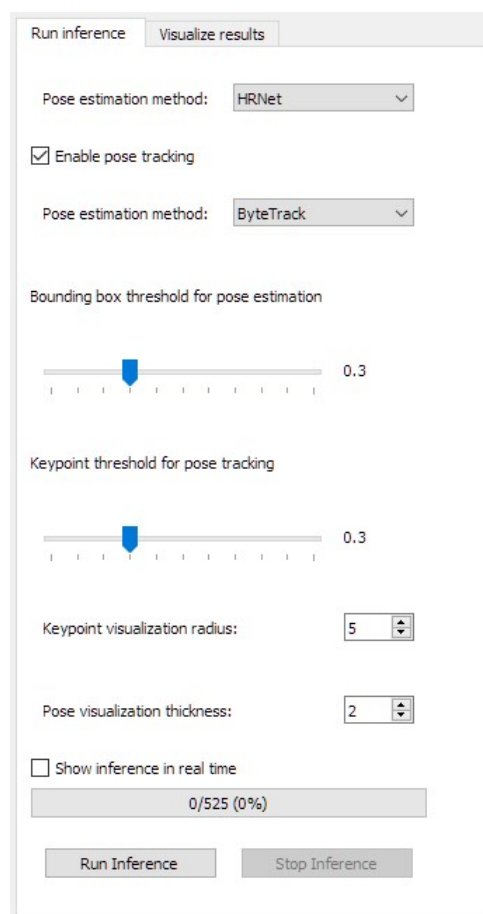


Figure 4.2: Detail of the pose estimation settings panel.

The second tab in the QTabWidget is used for the visualization of previously generated JSON result files, Figure 4.3. The main component of this tab is the QListWidget which holds all the visualized videos and is used to swap between them. Two positions in this list are reserved for a specific purposes that are always the same. First position is always reserved for the original video which remains loaded in the memory all the time so that the estimation or visualization have the unprocessed video to work with. The second position is always occupied by the most recent estimation result which can be a partially processed video if the estimation was terminated by the user or a fully processed video if the estimation finished. The displayed video is determined by the active item in this list widget. The information about the selected video, such as the pose estimation method, tracking method and their corresponding thresholds, is shown under the video list.

A result in a JSON file can be added to the list by the button at the bottom of the tab. Upon selecting a JSON file with the results, the video is processed and the bounding boxes and poses are visualized in the video. Similarly to the estimation, the keypoint radius and line thickness can be adjusted and do not apply retroactively. When a result is visualized, the video can be exported and saved. Additionally, when the inference result is selected, the result can be exported to a JSON file.

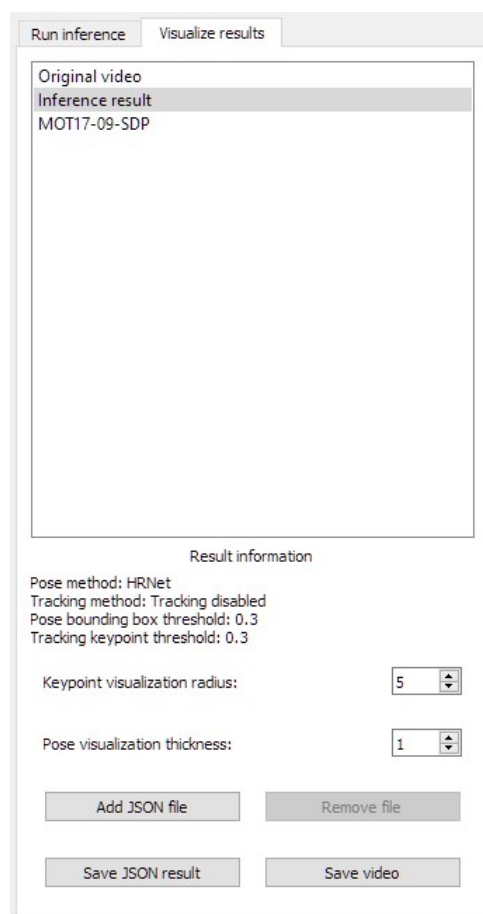


Figure 4.3: Detail of the result visualization panel.

Conclusion

Several objectives were defined in the thesis assignment. First objective was to select several methods for pose estimation based on the preliminary research of published methods and then the selected methods were to be tested and compared with each other. Next, a few methods for pose tracking were to be selected, tested and compared. Additionally, a new unannotated video dataset was to be created and used to test the methods. The last objective was to create an application in which pose estimation and pose tracking can be visualized in a user friendly interface.

As per the first goal, seven pose estimation methods were selected. These include four top-down methods and three bottom-up methods to provide a comparison between the two paradigms. The methods were introduced along with the two most popular datasets for pose estimation on which the methods were compared. The comparison metric AP and it's sub-metrics were also described. From these methods, AlphaPose achieved the best accuracy for all measured metrics with DEKR performing fairly similar. In terms of estimation speed, AlphaPose and HRNet achieved comparable results thanks to their top-down approach. Important to note, that on CrowdPose with a higher number of persons per frame, AlphaPose achieved significantly higher inference speed than the rest with LitePose being the second fastest.

The second objective was to compare a few pose tracking methods and compare the performance between them. Three methods were selected and tested on two different datasets. ByteTrack and OC-SORT performed very similar accuracy on the less crowded MOT17 dataset with QDTrack achieved slightly worse but still respectable results. On the very highly crowded MOT20 dataset, OC-SORT outperformed ByteTrack in all measured accuracy metrics and QDTrack achieved significantly worse results due to the FasterRCNN detector not handling highly occluded scenes very well. For comparison, three greedy pose tracking methods were also tested and showed poor performance, especially on the MOT20 dataset. Inference speed showed greedy tracking to perform the fastest with QDTrack being only slightly slower while achieving much higher accuracy.

An application was developed as a part of the last objective. It can be used to run pose estimation with pose tracking from a user friendly interface. The results can be saved as a video or to a JSON file. The JSON files from this application can then be re-opened in the application to be visualized on the video.

The conclusion from these results and comparisons is as follows. AlphaPose performed the best of all the tested methods for pose estimation. For pose tracking the overall best performing was OC-SORT in terms of accuracy and QDTrack in terms

of accuracy with decent inference speed. Important to note, that greedy tracking did not perform well on these datasets, but could be adequate for pose tracking on a security camera feed, where there's little to no chance of occlusion, e.g., hallway or warehouse cameras.

Bibliography

- [1] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft, "Simple online and realtime tracking," *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464-3468. <https://doi.org/10.1109/ICIP.2016.7533003>
- [2] A. Doering, U. Iqbal, J. Gall, "Joint Flow: Temporal Flow Fields for Multi Person Tracking," 2018. <https://doi.org/10.48550/arXiv.1805.04596>
- [3] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking," 2016. <https://doi.org/10.48550/arXiv.1603.00831>
- [4] A. Newell, Z. Huang, J. Deng, "Associative embedding: End-to-end learning for joint detection and grouping." In *NeurIPS*, 2017, pp. 2274–2284. <https://doi.org/10.48550/arXiv.1611.05424>
- [5] A. Newell, K. Yang, J. Deng, "Stacked Hourglass Networks for Human Pose Estimation," *2016 European Conference on Computer Vision (ECCV)*, 2016, pp. 483–499. https://doi.org/10.1007/978-3-319-46484-8_29
- [6] A. Toshev, C. Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1653-1660. <https://doi.org/10.1109/CVPR.2014.214>
- [7] Technology Agency of the Czech Republic, "AISEE - Artificial Intelligence based Search Environment for video/photo." <https://starfos.tacr.cz/en/projekty/VJ02010029>
- [8] B. Cheng, B. Xiao, J. Wang, H. Shi, T. S. Huang, L. Zhang, "HigherHRNet: Scale-Aware Representation Learning for Bottom-Up Human Pose Estimation," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 5385-5394. <https://doi.org/10.1109/CVPR42600.2020.00543>
- [9] D. Ganesh, R. R. Teja, C. D. Reddy, D. Swathi, "Human Action Recognition based on Depth maps, Skeleton and Sensor Images using Deep Learning," *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, 2022, pp. 1-8. <https://doi.org/10.1109/GCAT55367.2022.9971982>
- [10] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, B. Schiele, "DeeperCut: A Deeper, Stronger, and Faster Multi-person Pose Estimation Model," *2016 European Conference on Computer Vision (ECCV)*, 2016, pp. 34–50. https://doi.org/10.1007/978-3-319-46466-4_3

- [11] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, C. Tomasi, "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking," *2018 ECCV Workshop on Benchmarking Multi-Target Tracking*, 2018, pp. 17–35. <https://doi.org/10.48550/arXiv.1609.01775>
- [12] G. Ning, J. Pei, H. Huang, "LightTrack: A Generic Framework for Online Top-Down Human Pose Tracking," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 4456-4465. <https://doi.org/10.1109/CVPRW50498.2020.00525>
- [13] H. S. Fang et al., "AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. <https://doi.org/10.1109/TPAMI.2022.3222784>
- [14] H. W. Kuhn, "The hungarian method for the assignment problem." In *Naval research logistics quarterly*, vol. 52, 1955, pp. 83–97.
- [15] J. Cao, J. Pang, X. Weng, R. Khirodkar, K. Kitani, "Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 9686-9696. <https://doi.org/10.1109/CVPR52729.2023.00934>
- [16] J. Carreira, P. Agrawal, K. Fragkiadaki, J. Malik, "Human Pose Estimation with Iterative Error Feedback," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4733-4742. <https://doi.org/10.1109/CVPR.2016.512>
- [17] J. Li, C. Wang, H. Zhu, Y. Mao, H. -S. Fang, C. Lu, "CrowdPose: Efficient Crowded Scenes Pose Estimation and a New Benchmark," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10855-10864. <https://doi.org/10.1109/CVPR.2019.01112>
- [18] J. Luiten, A. Ošep, P. Dendorfer et al., "HOTA: A Higher Order Metric for Evaluating Multi-object Tracking," in *International Journal of Computer Vision*, vol. 129, 2021, pp. 548–578. <https://doi.org/10.1007/s11263-020-01375-2>
- [19] J. Pang et al., "Quasi-Dense Similarity Learning for Multiple Object Tracking," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 164-173. <https://doi.org/10.1109/CVPR46437.2021.00023>
- [20] J. Redmon, A. Farhadi, "YOLOv3: An Incremental Improvement," 2018. <https://doi.org/10.48550/arXiv.1804.02767>
- [21] K. Bernardin, R. Stiefelhagen, "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics," *EURASIP Journal on Image and Video Processing*, 2008, pp. 1-10. <https://doi.org/10.1155/2008/246309>
- [22] K. Chen et al., "MMDetection: Open MMLab Detection Toolbox and Benchmark," 2019. <https://doi.org/10.48550/arXiv.1906.07155>

- [23] K. Chen et al., "Patient-Specific Pose Estimation in Clinical Environments," in *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 6, 2018, pp. 1-11. <https://doi.org/10.1109/JTEHM.2018.2875464>
- [24] K. Sun, B. Xiao, D. Liu, J. Wang, "Deep High-Resolution Representation Learning for Human Pose Estimation," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5686-5696. <https://doi.org/10.1109/CVPR.2019.00584>
- [25] K. Wang, R. Zhao, Q. Ji, "Human Computer Interaction with Head Pose, Eye Gaze and Body Gestures," *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, 2018, pp. 789-789. <https://doi.org/10.1109/FG.2018.00126>
- [26] K. Wang, Q. Ji, "Real Time Eye Gaze Tracking with 3D Deformable Eye-Face Model," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1003-1011. <https://doi.org/10.1109/ICCV.2017.114>
- [27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," in *International Journal of Computer Vision*, vol. 115, 2015, pp. 211-252. <https://doi.org/10.1007/s11263-015-0816-y>
- [28] M. Andriluka, L. Pishchulin, P. Gehler, B. Schiele, "2D Human Pose Estimation: New Benchmark and State of the Art Analysis," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3686-3693. <https://doi.org/10.1109/CVPR.2014.471>
- [29] MediaPipe. <https://developers.google.com/mediapipe>
- [30] MMTracking Contributors, "MMTracking: OpenMMLab video perception toolbox and benchmark." <https://github.com/open-mmlab/mtracking>
- [31] M. J. Islam, J. Mo, J. Sattar , "Robot-to-robot relative pose estimation using humans as markers," in *Autonomous Robots*, vol. 45, 2021, pp. 579-593. <https://doi.org/10.1007/s10514-021-09985-6>
- [32] N. Wojke, A. Bewley, D. Paulus, "Simple online and realtime tracking with a deep association metric," *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3645-3649. <https://doi.org/10.1109/ICIP.2017.8296962>
- [33] P. Dendorfer, H. Rezatofghi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, L. Leal-Taixé, "MOT20: A benchmark for multi object tracking in crowded scenes," 2016. <https://doi.org/10.48550/arXiv.2003.09003>
- [34] OpenMMLab Pose Estimation Toolbox and Benchmark. <https://github.com/open-mmlab/mmpose>

- [35] Q. Bao, W. Liu, Y. Cheng, B. Zhou, T. Mei, "Pose-Guided Tracking-by-Detection: Robust Multi-Person Pose Tracking," in *IEEE Transactions on Multimedia*, vol. 23, 2021, pp. 161-175. <https://doi.org/10.1109/TMM.2020.2980194>
- [36] Q. Dang, J. Yin, B. Wang, W. Zheng, "Deep learning based 2D human pose estimation: A survey," in *Tsinghua Science and Technology*, vol. 24, no. 6, 2019, pp. 663-676. <https://doi.org/10.26599/TST.2018.9010100>
- [37] S. Dubey, M. Dixit, "A comprehensive survey on human pose estimation approaches," in *A comprehensive survey on human pose estimation approaches*, vol. 29, 2023, pp. 167–195. <https://doi.org/10.1007/s00530-022-00980-0>
- [38] S. Ren, K. He, R. Girshick, J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, 2017, pp. 1137-1149. <https://doi.org/10.1109/TPAMI.2016.2577031>
- [39] Y. Zhu, C. Detig, S. Kane, G. Lourie, "Kinematic Motion Analysis with Volumetric Motion Capture," *2022 26th International Conference Information Visualisation (IV)*, 2020, pp. 61-66. <https://doi.org/10.1109/IV56949.2022.00019>
- [40] T. Lin, M. Maire, S.J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, C.L. Zitnick, "Microsoft COCO: common objects in context", *European Conference on Computer Vision, 2014*, pp. 740–755. https://doi.org/10.1007/978-3-319-10602-1_48
- [41] U. Iqbal, A. Milan, J. Gall, "PoseTrack: Joint Multi-person Pose Estimation and Tracking," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4654-4663. <https://doi.org/10.1109/CVPR.2017.495>
- [42] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, M. Grundmann, "BlazePose: On-device Real-time Body Pose tracking," *CVPR Workshop on Computer Vision for Augmented and Virtual Reality*, 2020. <https://doi.org/10.48550/arXiv.2006.10204>
- [43] W. Liu, Q. Bao, Y. Sun, T. Mei, "Recent Advances in Monocular 2D and 3D Human Pose Estimation: A Deep Learning Perspective," *ACM Computing Surveys*, vol. 55, no. 4, 2022, pp. 1-41. <https://doi.org/10.1145/3524497>
- [44] W. Tang, P. Yu, Y. Wu, "Deeply Learned Compositional Models for Human Pose Estimation," *Computer Vision – ECCV 2018*, 2018, pp. 197–214. https://doi.org/10.1007/978-3-030-01219-9_12
- [45] Y. Wang, M. Li, H. Cai, W. Chen, S. Han, "Lite Pose: Efficient Architecture Design for 2D Human Pose Estimation," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 13116-13126. <https://doi.org/10.1109/CVPR52688.2022.01278>

- [46] Y. Xiu, J. Li, H. Wang, Y. Fang, C. Lu, "Pose Flow: Efficient Online Pose Tracking," *British Machine Vision Conference (BMVC)*, 2018. <https://doi.org/10.48550/arXiv.1802.00977>
- [47] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, X. Wang, "ByteTrack: Multi-Object Tracking by Associating Every Detection Box," *2022 European Conference on Computer Vision (ECCV)*, 2022, pp. 1-21. https://doi.org/10.1007/978-3-031-20047-2_1
- [48] MediaPipe. <https://www.youtube.com/>
- [49] Z. Cao, T. Simon, S. -E. Wei, Y. Sheikh, "Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1302-1310. <https://doi.org/10.1109/CVPR.2017.143>
- [50] Z. Geng, K. Sun, B. Xiao, Z. Zhang, J. Wang, "Bottom-Up Human Pose Estimation Via Disentangled Keypoint Regression," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 14671-14681. <https://doi.org/10.1109/CVPR46437.2021.01444>
- [51] Z. Ge, S. Liu, F. Wang, Z. Li, J. Sun, "YOLOX: Exceeding YOLO Series in 2021," 2021. <https://doi.org/10.48550/arXiv.2107.08430>

Appendix A

Pose tracking dataset

Length	Crowding	Description	YouTube source
0:50	1	Street with snow at night	youtu.be/dQTwIQiGRTg
1:56	2	Intersection camera	youtu.be/c-ofL3wVG-Y
0:13	2	Top-down view of a mall	youtu.be/ABL1btIACYo
0:10	2	Airport, shot from an escalator	youtu.be/9pyz3HsS0S8
6:29	3	Basketball, moving drone shot	youtu.be/DVMeGqeMbHA
1:01	3	Escalator in a mall	youtu.be/gnxK4xHG1_0
0:26	3	Basketball, stationary drone shot	youtu.be/UlhR0C3ELog
1:29	4	Wide street	youtu.be/XgAcTEjDHnA
0:13	4	Top-down view of a mall	youtu.be/WvhYuDvH17I
0:21	4	People on a bridge, slow motion	youtu.be/6AuFIA0YHgQ
0:29	5	Airport, mainly stationary people	youtu.be/l64NZjETrps
0:08	6	Crowded street	youtu.be/KCt-Qn37GTg
0:26	6	Crowded street in low light	youtu.be/4j6BB6YEGAo
0:09	6	Crowded street during sunset	youtu.be/XNgU_Lk3VAw
5:26	6	Airport, stationary camera	youtu.be/F7_BwCS6r5M
2:19	7	Crowded street	youtu.be/bwJ-TNu0hGM
1:56	8	Multiple crowded streets	youtu.be/YzcawvDGe4Y
1:15	9	Very crowded street, high occlusion	youtu.be/6NBwbKMyzEE

Table A.1: Videos in the created dataset with their length, crowding level, description and the corresponding YouTube source link. Crowding level is estimated by the eye.

Appendix B

Application source code

B.1 Main event loop and user interface

```
1 import sys
2 import os
3 import traceback
4 import mmcv
5 import cv2
6 import QImage2ndarray
7 import numpy as np
8 import json
9 import globals
10 from PyQt5.QtCore import Qt, pyqtSlot, QRunnable, QObject,
    QThreadPool, pyqtSignal, QDir, QTimer, QSize
11 from PyQt5.QtGui import QPixmap
12 from PyQt5.QtWidgets import QApplication, QFileDialog, QMainWindow,
    QSlider, QStyle, QVBoxLayout, QSpinBox,
13     QHBoxLayout, QPushButton, QWidget,
14     QLabel, QCheckBox, QComboBox, QProgressBar,
    QProgressDialog, QTabWidget,
15     QListWidget, QMessageBox)
16 from MMpose import mmpose_inference
17 from mmpose.apis import init_pose_model, vis_pose_tracking_result
18 from mmpose.datasets import DatasetInfo
19
20 class WorkerSignals(QObject):
21     finished = pyqtSignal()
22     error = pyqtSignal(tuple)
23     result = pyqtSignal(list)
24     progress = pyqtSignal(dict)
25
26
27 class Worker(QRunnable):
28     def __init__(self, in_file, tracking_type, pose, bbox_thr,
29         tracking_thr, kpt_radius, line_thickness):
30         super(Worker, self).__init__()
31         # Store constructor arguments (re-used for processing)
```

```

32     self.in_file = in_file
33     self.tracking_type = tracking_type
34     self.pose = pose
35     self.bbox_thr = bbox_thr
36     self.tracking_thr = tracking_thr
37     self.kptradius = kpt_radius
38     self.linethickness = line_thickness
39     self.signals = WorkerSignals()
40
41     # Add the callback to our kwargs
42     self.progress_callback = self.signals.progress
43
44     @pyqtSlot()
45     def run(self):
46         '''
47         Initialise the runner function with passed args, kwargs.
48         '''
49
50         # Retrieve args/kwargs here; and fire processing using them
51         try:
52             result = mmpose_inference(self.progress_callback, self.
in_file, self.tracking_type, self.pose, self.bbox_thr,
53                                     self.tracking_thr, self.
kptradius, self.linethickness)
54         except:
55             traceback.print_exc()
56             exctype, value = sys.exc_info()[:2]
57             self.signals.error.emit((exctype, value, traceback.
format_exc()))
58         else:
59             self.signals.result.emit(result) # Return the result
of the processing
60         finally:
61             self.signals.finished.emit() # Done
62
63
64     class MainWindow(QMainWindow):
65
66         def __init__(self):
67             super().__init__()
68             self.setWindowTitle('Human pose estimation and tracking')
69
70             widget = QWidget()
71
72             self.video = [[], []]
73             self.temp_video = []
74             self.running = False
75             self.loaded_jsons = [{'pose': '--',
76                                 'track': '--',
77                                 'bbox_thr': 0,
78                                 'kpt_thr': 0},
79                                 {'pose': '--',
80                                 'track': '--',
81                                 'bbox_thr': 0,
82                                 'kpt_thr': 0}]
83             self.frame_label = QLabel()

```

```

84     self.frame_timer = QTimer()
85     self.frame_timer.timeout.connect(self.display_video_stream)
86     self.pause = True
87     self.fps = 0
88     self.length = 0
89     self.frameindex = -1
90     self.url = ""
91     self.originalsize = QSize()
92
93     self.inferenceresults = {"pose": "",
94                             "track": "",
95                             "video_path": "",
96                             "bbox_thr": 0,
97                             "kpt_thr": 0,
98                             "results": []}
99
100    file_menu = self.menuBar().addMenu("&File")
101    self.open_action = file_menu.addAction("Open video")
102    self.open_action.triggered.connect(self.open_file)
103    self.exit_action = file_menu.addAction("Exit")
104    self.exit_action.triggered.connect(self.exit_call)
105    self.about_action = self.menuBar().addAction("About")
106    self.about_action.triggered.connect(self.about)
107
108    # INFERENCE WIDGET CREATION
109    inferencewidget = QWidget()
110    inferencewidget.layout = QVBoxLayout(inferencewidget)
111    inferencewidget.setMaximumWidth(300)
112    inferencewidget.setMinimumWidth(300)
113    # Pose Method Combobox
114    posemethod = QWidget()
115    posemethod.layout = QHBoxLayout(posemethod)
116    self._poselabel = QLabel("Pose estimation method:")
117    self._posecombo = QComboBox()
118    self._posecombo.addItems(["Hourglass", "HRNet", "DEKR", "
HigherHRNet"])
119    self._posecombo.insertSeparator(2)
120    self._posecombo.setEnabled(False)
121    posemethod.layout.addWidget(self._poselabel)
122    posemethod.layout.addWidget(self._posecombo)
123    # Tracking Radio Button
124    self._track = QCheckBox()
125    self._track.setText("Enable pose tracking")
126    self._track.stateChanged.connect(self.enable_tracking)
127    self._track.setEnabled(False)
128    # Tracking Method Combobox
129    trackingmethod = QWidget()
130    trackingmethod.layout = QHBoxLayout(trackingmethod)
131    self._tracklabel = QLabel("Pose estimation method:")
132    self._trackcombo = QComboBox()
133    self._trackcombo.addItems(["Greedy by IoU", "Greedy by
distance", "Greedy by OKS", "ByteTrack", "OC-SORT",
"QDTrack"])
134    self._trackcombo.insertSeparator(3)
135    self._trackcombo.setEnabled(False)
136    trackingmethod.layout.addWidget(self._tracklabel)
137

```



```

138     trackingmethod.layout.addWidget(self._trackcombo)
139     # bbox_thr Slider
140     bboxthrslider = QWidget()
141     bboxthrslider.layout = QHBoxLayout(bboxthrslider)
142     self._bboxslider = QSlider()
143     self._bboxslider.setMaximum(100)
144     self._bboxslider.setMinimum(0)
145     self._bboxslider.setTickPosition(QSlider.TicksBelow)
146     self._bboxslider.setValue(30)
147     self._bboxslider.setOrientation(Qt.Horizontal)
148     self._bboxslider.valueChanged.connect(self.
bboxsliderchanged)
149     self._bboxslider.setEnabled(False)
150     self._bboxline = QLabel()
151     self._bboxline.setFixedWidth(50)
152     self._bboxline.setText(str(self._bboxslider.value() / 100)
+ "\n")
153     bboxthrslider.layout.addWidget(self._bboxslider)
154     bboxthrslider.layout.addSpacing(10)
155     bboxthrslider.layout.addWidget(self._bboxline)
156     self._bboxlabel = QLabel("Bounding box threshold for pose
estimation")
157     # kpt_thr Slider
158     kptthrslider = QWidget()
159     kptthrslider.layout = QHBoxLayout(kptthrslider)
160     self._kptslider = QSlider()
161     self._kptslider.setMaximum(100)
162     self._kptslider.setMinimum(0)
163     self._kptslider.setTickPosition(QSlider.TicksBelow)
164     self._kptslider.setValue(30)
165     self._kptslider.setOrientation(Qt.Horizontal)
166     self._kptslider.valueChanged.connect(self.kptsliderchanged)
167     self._kptslider.setEnabled(False)
168     self._kptline = QLabel()
169     self._kptline.setFixedWidth(50)
170     self._kptline.setText(str(self._kptslider.value() / 100) +
"\n")
171     kptthrslider.layout.addWidget(self._kptslider)
172     kptthrslider.layout.addSpacing(10)
173     kptthrslider.layout.addWidget(self._kptline)
174     self._kptlabel = QLabel("Keypoint threshold for pose
tracking")
175     # Keypoint radius
176     kptradius = QWidget()
177     kptradius.layout = QHBoxLayout(kptradius)
178     self._kptradius = QSpinBox()
179     self._kptradius.setRange(1, 10)
180     self._kptradius.setValue(4)
181     self._kptradius.setMaximumWidth(50)
182     self._kptradius.setEnabled(False)
183     self._kptradiuslabel = QLabel("Keypoint visualization
radius:")
184     kptradius.layout.addWidget(self._kptradiuslabel)
185     kptradius.layout.addWidget(self._kptradius)
186     # Line thickness
187     linethickness = QWidget()

```

```

188     linethickness.layout = QHBoxLayout(linethickness)
189     self._linethickness = QSpinBox()
190     self._linethickness.setRange(1, 5)
191     self._linethickness.setValue(1)
192     self._linethickness.setMaximumWidth(50)
193     self._linethickness.setEnabled(False)
194     self._linethicknesslabel = QLabel("Pose visualization
thickness:")
195     linethickness.layout.addWidget(self._linethicknesslabel)
196     linethickness.layout.addWidget(self._linethickness)
197     # Show Radio Button
198     self._show = QCheckBox()
199     self._show.setText("Show inference in real time")
200     self._show.setEnabled(False)
201     # Progress Bar
202     self._progressbar = QProgressBar()
203     self._progressbar.setFormat("%v/%m (%p%)")
204     self._progressbar.setValue(0)
205     self._progressbar.setAlignment(Qt.AlignCenter)
206     self._progressbar.setTextVisible(False)
207     # Start Stop Buttons
208     startstop = QWidget()
209     startstop.layout = QHBoxLayout(startstop)
210     self._buttonstart = QPushButton("Run Inference")
211     self._buttonstart.clicked.connect(self.start_inference)
212     self._buttonstart.setEnabled(False)
213     self._buttonstop = QPushButton("Stop Inference")
214     self._buttonstop.setEnabled(False)
215     self._buttonstop.clicked.connect(self.stop_inference)
216     startstop.layout.addWidget(self._buttonstart)
217     startstop.layout.addSpacing(10)
218     startstop.layout.addWidget(self._buttonstop)
219     # Put It All Together
220     inferencewidget.layout.addWidget(posemethod)
221     inferencewidget.layout.addWidget(self._track)
222     inferencewidget.layout.addWidget(trackingmethod)
223     inferencewidget.layout.addWidget(self._bboxlabel)
224     inferencewidget.layout.addWidget(bboxthrslider)
225     inferencewidget.layout.addWidget(self._kptlabel)
226     inferencewidget.layout.addWidget(kpthrslider)
227     inferencewidget.layout.addWidget(kprradius)
228     inferencewidget.layout.addWidget(linethickness)
229     inferencewidget.layout.addWidget(self._show)
230     inferencewidget.layout.addWidget(self._progressbar)
231     inferencewidget.layout.addWidget(startstop)
232
233     # Visualization widget
234     visualizationwidget = QWidget()
235     visualizationwidget.layout = QVBoxLayout(
visualizationwidget)
236     self._resultlist = QListWidget()
237     self._resultlist.itemClicked.connect(self.change_json)
238     visualizationwidget.layout.addWidget(self._resultlist)
239     self._infolabel1 = QLabel()
240     self._infolabel1.setText('Result information')
241     self._infolabel1.setAlignment(Qt.AlignCenter)

```

```

242     self._infolabel2 = QLabel()
243     self._infolabel2.setText('Pose method: --\n'
244                             'Tracking method: --\n'
245                             'Pose bounding box threshold: 0\n'
246                             'Tracking keypoint threshold: 0')
247     visualizationwidget.layout.addWidget(self._infolabel1)
248     visualizationwidget.layout.addWidget(self._infolabel2)
249     # Keypoint radius
250     kpradius1 = QWidget()
251     kpradius1.layout = QHBoxLayout(kpradius1)
252     self._kpradius1 = QSpinBox()
253     self._kpradius1.setRange(1, 10)
254     self._kpradius1.setValue(4)
255     self._kpradius1.setMaximumWidth(50)
256     self._kpradius1.setEnabled(False)
257     self._kpradiuslabel1 = QLabel("Keypoint visualization
radius:")
258     kpradius1.layout.addWidget(self._kpradiuslabel1)
259     kpradius1.layout.addWidget(self._kpradius1)
260     # Line thickness
261     linethickness1 = QWidget()
262     linethickness1.layout = QHBoxLayout(linethickness1)
263     self._linethickness1 = QSpinBox()
264     self._linethickness1.setRange(1, 5)
265     self._linethickness1.setValue(1)
266     self._linethickness1.setMaximumWidth(50)
267     self._linethickness1.setEnabled(False)
268     self._linethicknesslabel1 = QLabel("Pose visualization
thickness:")
269     linethickness1.layout.addWidget(self._linethicknesslabel1)
270     linethickness1.layout.addWidget(self._linethickness1)
271     visualizationwidget.layout.addWidget(kpradius1)
272     visualizationwidget.layout.addWidget(linethickness1)
273     # JSON Load and Save Buttons
274     jsonbuttons = QWidget()
275     jsonbuttons.layout = QHBoxLayout(jsonbuttons)
276     self._buttonopenjson = QPushButton("Add JSON file")
277     self._buttonopenjson.setEnabled(False)
278     self._buttonopenjson.clicked.connect(self.open_json)
279     self._buttonremovejson = QPushButton("Remove file")
280     self._buttonremovejson.setEnabled(False)
281     self._buttonremovejson.clicked.connect(self.remove_json)
282     jsonbuttons.layout.addWidget(self._buttonopenjson)
283     jsonbuttons.layout.addSpacing(10)
284     jsonbuttons.layout.addWidget(self._buttonremovejson)
285     visualizationwidget.layout.addWidget(jsonbuttons)
286     # Save Buttons
287     savebuttons = QWidget()
288     savebuttons.layout = QHBoxLayout(savebuttons)
289     self._buttonsavejson = QPushButton("Save JSON result")
290     self._buttonsavejson.setEnabled(False)
291     self._buttonsavejson.clicked.connect(self.save_json)
292     self._buttonsavevideo = QPushButton("Save video")
293     self._buttonsavevideo.setEnabled(False)
294     self._buttonsavevideo.clicked.connect(self.save_video)
295     savebuttons.layout.addWidget(self._buttonsavejson)

```

```

296     savebuttons.layout.addSpacing(10)
297     savebuttons.layout.addWidget(self._buttonsavevideo)
298     visualizationwidget.layout.addWidget(savebuttons)
299
300     # Create tab widget
301     self.tabwidget = QTabWidget()
302     self.tabwidget.addTab(inferencewidget, 'Run inference')
303     self.tabwidget.addTab(visualizationwidget, 'Visualize
results')
304
305     rightside = QWidget()
306     rightside.layout = QHBoxLayout(rightside)
307     rightside.layout.addWidget(self.tabwidget)
308
309     videowidget = QWidget()
310     self._playbutton = QPushButton()
311     self._playbutton.setEnabled(False)
312     self._playbutton.setIcon(self.style().standardIcon(QStyle.
SP_MediaPlay))
313     self._playbutton.clicked.connect(self.play)
314     self._slider = QSlider(Qt.Horizontal)
315     self._slider.setRange(0, 0)
316     self._slider.sliderMoved.connect(self.set_position)
317     self._slider.setEnabled(False)
318     self._durationlabel = QLabel()
319     self._durationlabel.setText("-- / --")
320     self._durationlabel.setFixedHeight(50)
321
322     controlwidget = QWidget()
323     controlwidget.layout = QHBoxLayout()
324     controlwidget.layout.addWidget(self._playbutton)
325     controlwidget.layout.addWidget(self._slider)
326     controlwidget.layout.addWidget(self._durationlabel)
327     controlwidget.setFixedHeight(50)
328
329     videowidget.layout = QVBoxLayout(videowidget)
330     videowidget.layout.addWidget(self.frame_label)
331     videowidget.layout.addLayout(controlwidget.layout)
332
333
334     # FINAL WIDGET CREATION
335     widget.layout = QHBoxLayout(widget)
336     widget.layout.addWidget(videowidget)
337     widget.layout.addWidget(rightside)
338     self.setCentralWidget(widget)
339
340     available_geometry = self.screen().availableGeometry()
341     self.setFixedSize(round(available_geometry.width() / 2),
round(available_geometry.height() / 2))
342     self.video_size = QSize(round(available_geometry.width() /
2) - 400,
343                             round(available_geometry.height() /
2) - 60)
344     image = np.zeros((self.video_size.height(), self.video_size
.width(), 3), np.uint8)
345     self.frame_label.setPixmap(QPixmap.fromImage(qimage2ndarray

```

```

.array2qimage(image)))
346
347     self.threadpool = QThreadPool()
348
349     def set_position(self, position):
350         if self._show.isChecked():
351             self._show.setChecked(False)
352         if not position == 0:
353             self.frameindex = position - 1
354         else:
355             self.frameindex = -1
356         self.display_video_stream()
357
358     def progress_fn(self, res):
359         frame_id = res["frame_id"]
360         image = res["image"]
361         self.video[1][frame_id] = image
362         self._progressbar.setValue(frame_id)
363         if self._show.isChecked():
364             self.frameindex = frame_id - 1
365             self.display_video_stream()
366
367     def handle_result(self, s):
368         self.inferenceresults["results"] = s
369
370     def thread_complete(self):
371         self._progressbar.setValue(self._progressbar.value() + 1)
372         self._buttonstop.setEnabled(False)
373         self._buttonstart.setEnabled(True)
374         self._resultlist.item(1).setHidden(False)
375         self.running = False
376         self._resultlist.setCurrentRow(1)
377         self.change_json()
378
379     def about(self):
380         QMessageBox.about(self, 'About', "
Application made by Bc. Antonin Cech\n"
381                                     "as a part of a master's
thesis on human pose estimation and tracking  \n"
382                                     "                                     at FNSPE
CTU in Prague, Czech Republic")
383
384     def bboxsliderchanged(self):
385         self._bboxline.setText(str(self._bboxslider.value() / 100))
386
387     def kptsliderchanged(self):
388         self._kptline.setText(str(self._kptslider.value() / 100))
389
390     def enable_tracking(self):
391         if self._trackcombo.isEnabled():
392             self._trackcombo.setEnabled(False)
393             self._kptslider.setEnabled(False)
394         else:
395             self._trackcombo.setEnabled(True)
396             self._kptslider.setEnabled(True)
397

```

```

398     def start_inference(self):
399         self._buttonstart.setEnabled(False)
400         globals.kill_thread = False
401         tracking = self._trackcombo.currentText() if self._track.
isChecked() else "Tracking disabled"
402         self.inferenceresults["pose"] = self._posecombo.currentText
()
403         self.inferenceresults["track"] = tracking
404         self.inferenceresults["video_path"] = self.url
405         self.inferenceresults["bbox_thr"] = self._bboxslider.value
() / 100
406         self.inferenceresults["kpt_thr"] = self._kptslider.value()
/ 100
407         self.loaded_jsons[1]['pose'] = self.inferenceresults["pose"
]
408         self.loaded_jsons[1]['track'] = self.inferenceresults["
track"]
409         self.loaded_jsons[1]['bbox_thr'] = self.inferenceresults["
bbox_thr"]
410         self.loaded_jsons[1]['kpt_thr'] = self.inferenceresults["
kpt_thr"]
411         self.video[1] = self.video[0].copy()
412         self.worker = Worker(self.url, tracking, self._posecombo.
currentText(),
413                               self._bboxslider.value() / 100, self.
_kptslider.value() / 100, self._kptradius.value(),
414                               self._linethickness.value())
415         self.worker.signals.result.connect(self.handle_result)
416         self.worker.signals.finished.connect(self.thread_complete)
417         self.worker.signals.progress.connect(self.progress_fn)
418         self._buttonstop.setEnabled(True)
419         self.running = True
420
421         # Execute
422         self.threadpool.start(self.worker)
423
424     def stop_inference(self):
425         globals.kill_thread = True
426
427     def save_json(self):
428         videoname, _ = os.path.splitext(os.path.basename(self.url))
429         fileName, _ = QFileDialog.getSaveFileName(self, "Save JSON"
, os.path.dirname(self.url) + '/' + videoname
430                                                     + "_result.json",
431                                                     "JSON Files (*.
json));;All Files")
432
433         if fileName != '':
434             with open(fileName, "w") as outfile:
435                 json.dump(self.inferenceresults, outfile)
436
437     def save_video(self):
438         videoname, _ = os.path.splitext(os.path.basename(self.url))
439         fileName, _ = QFileDialog.getSaveFileName(self, "Save video
", os.path.dirname(self.url) + '/' + videoname
440                                                     + "_result.avi",

```

```

441                                                                 "Video Files (*.
avi);;All Files")
442     if fileName != '':
443         fourcc = cv2.VideoWriter_fourcc(*'mp4v')
444         videoWriter = cv2.VideoWriter(
445             fileName, fourcc, self.fps, (self.originalsize.
width(), self.originalsize.height()))
446         item = self._resultlist.currentRow()
447         self.progress = QProgressDialog("Saving video...", "
Cancel", 0, self.length, self)
448         self.progress.setWindowModality(Qt.WindowModal)
449         self.progress.show()
450         for i in range(self.length):
451             self.progress.setValue(i + 1)
452             if self.progress.wasCanceled():
453                 break
454             videoWriter.write(self.video[item][i])
455             videoWriter.release()
456
457     def open_file(self):
458         self.ensure_stopped()
459         self.video = [[], []]
460         self.fps = 0
461         self.length = 0
462         self.frameindex = -1
463         self._resultlist.clear()
464         fileName, _ = QFileDialog.getOpenFileName(self, "Open video
", QDir.homePath())
465
466         if fileName != '':
467             video = mmcv.VideoReader(fileName)
468             self.progress = QProgressDialog("Loading video...", "
Cancel", 0, video.frame_cnt, self)
469             self.progress.setWindowModality(Qt.WindowModal)
470             self.progress.show()
471             for i in range(video.frame_cnt):
472                 self.progress.setValue(i + 1)
473                 if self.progress.wasCanceled():
474                     break
475                 self.video[0].append(video[i])
476             self.fps = video.fps
477             self.length = len(self.video[0])
478             self.url = fileName
479             self.originalsize = QSize(video.width, video.height)
480             self._progressbar.setRange(0, self.length)
481             self._progressbar.setTextVisible(True)
482             self._playbutton.setEnabled(True)
483             self._posecombo.setEnabled(True)
484             self._track.setEnabled(True)
485             self._bboxslider.setEnabled(True)
486             self._kpctradius.setEnabled(True)
487             self._linethickness.setEnabled(True)
488             self._kpctradius1.setEnabled(True)
489             self._linethickness1.setEnabled(True)
490             self._show.setEnabled(True)
491             self._buttonstart.setEnabled(True)

```

```

492         self._buttonopenjson.setEnabled(True)
493         self._resultlist.addItem("Original video")
494         self._resultlist.addItem("Inference result")
495         self._resultlist.item(1).setHidden(True)
496         self._resultlist.setCurrentRow(0)
497         self._slider.setEnabled(True)
498         self._slider.setRange(0, self.length - 1)
499         self._progressbar.setValue(0)
500         self.frameindex = -1
501         self.display_video_stream()
502
503     def open_json(self):
504         fileName, _ = QFileDialog.getOpenFileName(self, "Open JSON
result", QDir.homePath())
505
506         if fileName != '':
507             with open(fileName, "r") as infile:
508                 file_contents = json.loads(infile.read())
509                 self.tabwidget.setCurrentIndex(1)
510                 videoname, _ = os.path.splitext(os.path.basename(
file_contents["video_path"]))
511                 self._resultlist.addItem(videoname)
512                 num = self._resultlist.count()
513                 file_contents["num"] = num
514                 self.loaded_jsons.append(file_contents)
515                 if file_contents["pose"] == "Hourglass":
516                     pose_config = './configs/hourglass.py'
517                     pose_checkpoint = './checkpoints/hourglass.pth'
518                 elif file_contents["pose"] == 'HRNet':
519                     pose_config = './configs/hrnet.py'
520                     pose_checkpoint = './checkpoints/hrnet.pth'
521                 elif file_contents["pose"] == "DEKR":
522                     pose_config = './configs/dekr.py'
523                     pose_checkpoint = './checkpoints/dekr.pth'
524                 elif file_contents["pose"] == "HigherHRNet":
525                     pose_config = './configs/higher_hrnet.py'
526                     pose_checkpoint = './checkpoints/higher_hrnet.pth'
527
528                 self.progress = QProgressDialog("Visualizing results...
", "Cancel", 0, len(file_contents["results"]), self)
529                 self.progress.setWindowModality(Qt.WindowModal)
530                 self.progress.show()
531
532                 pose_model = init_pose_model(
533                     pose_config, pose_checkpoint, device='cuda:0')
534
535                 dataset = pose_model.cfg.data['test']['type']
536                 dataset_info = pose_model.cfg.data['test'].get('
dataset_info', None)
537                 dataset_info = DatasetInfo(dataset_info)
538
539                 vid = self.video[0].copy()
540                 self.video.append(vid)
541
542                 for i in range(len(file_contents["results"])):
543                     self.progress.setValue(i + 1)

```



```

544         if self.progress.wasCanceled():
545             break
546         self.video[num - 1][i] = vis_pose_tracking_result(
547             pose_model,
548             self.video[0][
i],
549             file_contents[
"results"][i],
550             radius=self.
_kptradius1.value(),
551             thickness=self
._linethickness1.value(),
552             dataset=
dataset,
553             dataset_info=
dataset_info,
554             kpt_score_thr
=0.3,
555             show=False)
556         self._resultlist.setCurrentRow(num - 1)
557         self._infolabel2.setText(f'Pose method: {file_contents
["pose"]}\n'
558                                 f'Tracking method: {
file_contents["track"]}\n'
559                                 f'Pose bounding box threshold:
{file_contents["bbox_thr"]}\n'
560                                 f'Tracking keypoint threshold:
{file_contents["kpt_thr"]}')
561         self.frameindex = self.frameindex - 1
562         self.display_video_stream()
563
564     def remove_json(self):
565         item = self._resultlist.currentRow()
566         self._resultlist.takeItem(item)
567         del self.loaded_jsons[item]
568         del self.video[item]
569         self._resultlist.setCurrentRow(0)
570         self._buttonremovejson.setEnabled(False)
571         self._infolabel2.setText('Pose method: --\n'
572                                 'Tracking method: --\n'
573                                 'Pose bounding box threshold: --\n'
574                                 'Tracking keypoint threshold: --')
575
576     def change_json(self):
577         item = self._resultlist.currentRow()
578         self._infolabel2.setText(f'Pose method: {self.loaded_jsons[
item]["pose"]}\n'
579                                 f'Tracking method: {self.
loaded_jsons[item]["track"]}\n'
580                                 f'Pose bounding box threshold: {
self.loaded_jsons[item]["bbox_thr"]}\n'
581                                 f'Tracking keypoint threshold: {
self.loaded_jsons[item]["kpt_thr"]}')
582         self.frameindex = self.frameindex - 1
583         self.display_video_stream()

```

```

584     self._linethickness1.setValue(self._linethickness.value())
585     self._kpctradius1.setValue(self._kpctradius.value())
586     if item == 0:
587         self._buttonremovejson.setEnabled(False)
588         self._buttonsavejson.setEnabled(False)
589         self._buttonsavevideo.setEnabled(False)
590     elif item == 1:
591         self._buttonremovejson.setEnabled(False)
592         self._buttonsavejson.setEnabled(True)
593         self._buttonsavevideo.setEnabled(True)
594     else:
595         self._buttonremovejson.setEnabled(True)
596         self._buttonsavejson.setEnabled(False)
597         self._buttonsavevideo.setEnabled(True)
598
599     def exit_call(self):
600         self.ensure_stopped()
601         self.close()
602
603     def play(self):
604         if self._show.isChecked():
605             self._show.setChecked(False)
606         if self.frameindex == self.length - 1:
607             self.frameindex = -1
608             self.display_video_stream()
609
610         if not self.pause:
611             self.frame_timer.stop()
612             self._playbutton.setIcon(
613                 self.style().standardIcon(QStyle.SP_MediaPlay))
614         else:
615             self.frame_timer.start(int(1000 // self.fps))
616             self._playbutton.setIcon(
617                 self.style().standardIcon(QStyle.SP_MediaPause))
618
619         self.pause = not self.pause
620
621     def display_video_stream(self):
622         if self.frameindex == self.length - 1:
623             self.frame_timer.stop()
624             self._playbutton.setIcon(
625                 self.style().standardIcon(QStyle.SP_MediaPlay))
626             self.pause = True
627         else:
628             self.frameindex += 1
629             self._slider.setValue(self.frameindex)
630             self._durationlabel.setText(str(self.frameindex + 1) +
631 " / " + str(self.length))
632
633             if self.running:
634                 frame = cv2.cvtColor(self.video[1][self.frameindex
635 ], cv2.COLOR_BGR2RGB)
636             else:
637                 item = self._resultlist.currentRow()
638                 frame = cv2.cvtColor(self.video[item][self.
639 frameindex], cv2.COLOR_BGR2RGB)

```

```

637
638         h, w = frame.shape[:2]
639         sw, sh = self.video_size.width(), self.video_size.
height()
640         padColor = 0
641
642         # interpolation method
643         if h > sh or w > sw: # shrinking image
644             interp = cv2.INTER_AREA
645         else: # stretching image
646             interp = cv2.INTER_CUBIC
647
648         # aspect ratio of image
649         aspect = w / h # if on Python 2, you might need to
cast as a float: float(w)/h
650
651         # compute scaling and pad sizing
652         if aspect > 1: # horizontal image
653             new_w = sw
654             new_h = np.round(new_w / aspect).astype(int)
655             pad_vert = (sh - new_h) / 2
656             pad_top, pad_bot = np.floor(pad_vert).astype(int),
np.ceil(pad_vert).astype(int)
657             pad_left, pad_right = 0, 0
658         elif aspect < 1: # vertical image
659             new_h = sh
660             new_w = np.round(new_h * aspect).astype(int)
661             pad_horz = (sw - new_w) / 2
662             pad_left, pad_right = np.floor(pad_horz).astype(int
), np.ceil(pad_horz).astype(int)
663             pad_top, pad_bot = 0, 0
664         else: # square image
665             new_h, new_w = sh, sw
666             pad_left, pad_right, pad_top, pad_bot = 0, 0, 0, 0
667
668         # set pad color
669         if len(frame.shape) == 3 and not isinstance(padColor, (
670             list, tuple, np.ndarray)): # color image but only one
color provided
671             padColor = [padColor] * 3
672
673         # scale and pad
674         scaled_img = cv2.resize(frame, (new_w, new_h),
interpolation=interp)
675         scaled_img = cv2.copyMakeBorder(scaled_img, pad_top,
pad_bot, pad_left, pad_right,
676                                         borderType=cv2.
BORDER_CONSTANT, value=padColor)
677
678         image = QImage2ndarray.array2qimage(scaled_img)
679         self.frame_label.setPixmap(QPixmap.fromImage(image))
680
681     def ensure_stopped(self):
682         if not self.pause:
683             self.frame_timer.stop()
684             self.play_pause_button.setText("Play")

```

```

685         self._playbutton.setIcon(
686             self.style().standardIcon(QStyle.SP_MediaPlay))
687
688
689 if __name__ == '__main__':
690     globals.initialize()
691     app = QApplication(sys.argv)
692     main_win = MainWindow()
693     main_win.show()
694     sys.exit(app.exec())

```

B.2 Pose estimation and tracking functions

```

1     import warnings
2     import numpy as np
3     import mmcv
4     import globals
5
6     from mmpose.apis import (get_track_id,
7                             inference_top_down_pose_model,
8                             init_pose_model,
9                             process_mmdet_results,
10                            vis_pose_tracking_result,
11                            inference_bottom_up_pose_model)
12    from mmpose.core import Smoother
13    from mmpose.datasets import DatasetInfo
14    from mmdet.apis import inference_detector, init_detector
15    from mmtrack.apis import inference_mot
16    from mmtrack.apis import init_model as init_tracking_model
17
18    def get_dist(pose, pose_last):
19        dist = 0
20        for i in range(17):
21            dist += np.sqrt((pose[i, 1] - pose_last[i, 1])**2 + (pose
22                [i, 2] - pose_last[i, 2])**2)
23        return dist
24
25    def get_iou(bbox, bbox_last):
26        x1 = max(bbox[0], bbox_last[0])
27        y1 = max(bbox[1], bbox_last[1])
28        x2 = min(bbox[2], bbox_last[2])
29        y2 = min(bbox[3], bbox_last[3])
30
31        area = (bbox[2] - bbox[0]) * (bbox[3] - bbox[1])
32        area_last = (bbox_last[2] - bbox_last[0]) * (bbox_last[3] -
33            bbox_last[1])
34
35        intersection = max(0, x2 - x1) * max(0, y2 - y1)
36        union = float(area + area_last - intersection)
37
38        if union == 0:
39            union = 1e-5
40            warnings.warn('Union is 0, setting union to 1e-5.')

```

```

38
39     iou = intersection / union
40
41     return iou
42
43
44 def track_iou(results, results_last, next_id, threshold,
45 min_keypoints=3):
46     if results_last is None:
47         results_last = []
48
49     for result in results:
50         if len(results_last) == 0:
51             track_id = -1
52         else:
53             max_iou = -1
54             max_index = -1
55
56             for index, res_last in enumerate(results_last):
57                 iou_score = get_iou(result['bbox'], res_last['bbox',
58 ])
59
60                 if iou_score > max_iou:
61                     max_iou = iou_score
62                     max_index = index
63
64                 if max_iou > threshold:
65                     track_id = results_last[max_index]['track_id']
66                     del results_last[max_index]
67                 else:
68                     track_id = -1
69
70     if track_id == -1:
71         if np.count_nonzero(result['keypoints'][:, 1]) >=
72 min_keypoints:
73             result['track_id'] = next_id
74             next_id += 1
75         else:
76             # If the number of keypoints detected is small,
77             # delete that person instance.
78             result['keypoints'][:, 1] = -10
79             result['bbox'] *= 0
80             result['track_id'] = -1
81         else:
82             result['track_id'] = track_id
83
84     return results, next_id
85
86
87 def track_dist(results, results_last, next_id, threshold,
88 min_keypoints=3):
89     if results_last is None:
90         results_last = []
91
92     for result in results:
93         if len(results_last) == 0:
94             track_id = -1

```

```

90         else:
91             min_dist = np.inf
92             min_index = -1
93
94             # CALCULATE MINIMUM DISTANCE BETWEEN POSES
95             for index, res_last in enumerate(results_last):
96                 dist = get_dist(result['keypoints'], res_last['
keypoints'])
97                 if dist < min_dist:
98                     min_dist = dist
99                     min_index = index
100
101                 if min_dist < threshold:
102                     track_id = results_last[min_index]['track_id']
103                     del results_last[min_index]
104                 else:
105                     track_id = -1
106
107                 if track_id == -1:
108                     if np.count_nonzero(result['keypoints'][:, 1]) >=
min_keypoints:
109                         result['track_id'] = next_id
110                         next_id += 1
111                     else:
112                         # If the number of keypoints detected is small,
113                         # delete that person instance.
114                         result['keypoints'][:, 1] = -10
115                         result['bbox'] *= 0
116                         result['track_id'] = -1
117                 else:
118                     result['track_id'] = track_id
119
120             return results, next_id
121
122
123 def process_mmtracking_results(mmtracking_results):
124     person_results = []
125     if 'track_bboxes' in mmtracking_results:
126         tracking_results = mmtracking_results['track_bboxes'][0]
127     elif 'track_results' in mmtracking_results:
128         tracking_results = mmtracking_results['track_results'][0]
129
130     for track in tracking_results:
131         person = {}
132         person['track_id'] = int(track[0])
133         person['bbox'] = track[1:]
134         person_results.append(person)
135     return person_results
136
137
138 def mmpose_inference(progress_callback, in_file, track='iou', pose=
'hrnet', bbox_thr=0.3, tracking_thr=0.3,
139                       kpt_radius=4, line_thickness=1):
140     # Detection
141     det_config = './configs/faster_rcnn.py'
142     det_checkpoint = './checkpoints/faster_rcnn.pth'

```

```

143
144 # Tracking "Greedyly by IoU", "Greedyly by distance", "Greedyly
145 by OKS", "ByteTrack", "OC-SORT", "QDTrack"
146 tracking_type = ''
147 if track == "Greedyly by IoU":
148     tracking_type = 'iou'
149 elif track == "Greedyly by distance":
150     tracking_type = 'dist'
151 elif track == "Greedyly by OKS":
152     tracking_type = 'oks'
153 elif track == "ByteTrack":
154     tracking_type = 'bytetrack'
155     track_config = './configs/bytetrack.py'
156     track_checkpoint = './checkpoints/bytetrack.pth'
157 elif track == "OC-SORT":
158     tracking_type = 'ocsort'
159     track_config = './configs/ocsort.py'
160     track_checkpoint = './checkpoints/ocsort.pth'
161 elif track == "QDTrack":
162     tracking_type = 'qdtrack'
163     track_config = './configs/qdtrack.py'
164     track_checkpoint = './checkpoints/qdtrack.pth'
165
166 # Pose "Hourglass", "HRNet", "DEKR", "HigherHRNet"
167 if pose == "Hourglass":
168     pose_config = './configs/hourglass.py'
169     pose_checkpoint = './checkpoints/hourglass.pth'
170 elif pose == 'HRNet':
171     pose_config = './configs/hrnet.py'
172     pose_checkpoint = './checkpoints/hrnet.pth'
173 elif pose == "DEKR":
174     pose_config = './configs/dekr.py'
175     pose_checkpoint = './checkpoints/dekr.pth'
176 elif pose == "HigherHRNet":
177     pose_config = './configs/higher_hrnet.py'
178     pose_checkpoint = './checkpoints/higher_hrnet.pth'
179
180 print('Initializing model...')
181 if tracking_type != 'bytetrack' and tracking_type != 'ocsort'
182 and tracking_type != 'qdtrack':
183     det_model = init_detector(
184         det_config, det_checkpoint, device='cuda:0')
185 elif tracking_type != "Tracking disabled":
186     tracking_model = init_tracking_model(
187         track_config, track_checkpoint, device='cuda:0')
188
189 pose_model = init_pose_model(
190     pose_config, pose_checkpoint, device='cuda:0')
191
192 dataset = pose_model.cfg.data['test']['type']
193 dataset_info = pose_model.cfg.data['test'].get('dataset_info',
194 None)
195 if dataset_info is not None:
196     dataset_info = DatasetInfo(dataset_info)
197
198 # read video

```

```

196 video = mmcv.VideoReader(in_file)
197 assert video.opened, f'Failed to load video file {in_file}'
198
199 smoother = Smoother(filter_cfg='configs/_base_/filters/one_euro
.py', keypoint_dim=2)
200
201 next_id = 1
202 pose_results = []
203 pose_json = []
204 print('Running inference...')
205 for frame_id, cur_frame in enumerate(mmcv.track_iter_progress(
video)):
206     pose_results_last = pose_results
207
208     # get the detection results of current frame
209     # the resulting box is (x1, y1, x2, y2)
210     if tracking_type != 'bytetrack' and tracking_type != '
ocsort' and tracking_type != 'qdtrack':
211         mmdet_results = inference_detector(det_model, cur_frame
)
212
213         # keep the person class bounding boxes.
214         person_results = process_mmdet_results(mmdet_results,
1)
215     else:
216         mmtracking_results = inference_mot(
217             tracking_model, cur_frame, frame_id=frame_id)
218
219         # keep the person class bounding boxes.
220         person_results = process_mmtracking_results(
mmtracking_results)
221
222         # test a single image, with a list of bboxes.
223         if pose == 'DEKR' or pose == 'HigherHRNet':
224             pose_results, _ = inference_bottom_up_pose_model(
225                 pose_model,
226                 cur_frame,
227                 dataset=dataset,
228                 dataset_info=dataset_info,
229                 pose_nms_thr=0.9,
230                 return_heatmap=False,
231                 outputs=None)
232         else:
233             pose_results, _ = inference_top_down_pose_model(
234                 pose_model,
235                 cur_frame,
236                 person_results,
237                 bbox_thr=bbox_thr,
238                 format='xyxy',
239                 dataset=dataset,
240                 dataset_info=dataset_info,
241                 return_heatmap=False,
242                 outputs=None)
243
244         if tracking_type != 'bytetrack' and tracking_type != '
ocsort' and tracking_type != 'qdtrack':

```



```

245         if tracking_type == 'oks':
246             pose_results, next_id = get_track_id(
247                 pose_results,
248                 pose_results_last,
249                 next_id,
250                 use_oks=True,
251                 tracking_thr=tracking_thr)
252         elif tracking_type == 'dist':
253             pose_results, next_id = track_dist(pose_results,
254                                               pose_results_last, next_id,
255                                               threshold=
256                                               tracking_thr * 100)
257         elif tracking_type == 'iou':
258             pose_results, next_id = track_iou(pose_results,
259                                               pose_results_last, next_id, threshold=tracking_thr)
260         else:
261             for i in range(len(pose_results)):
262                 pose_results[i]["track_id"] = i
263
264         # post-process the pose results with smoother
265         pose_results = smoother.smooth(pose_results)
266
267         poselist = []
268         if pose == 'DEKR' or pose == 'HigherHRNet':
269             for i in range(len(pose_results)):
270                 keypoints = pose_results[i]["keypoints"]
271                 kptlist = keypoints.tolist()
272                 poselist.append({"keypoints": kptlist, "track_id":
273                                pose_results[i]["track_id"]})
274             pose_json.append(poselist)
275         else:
276             for i in range(len(pose_results)):
277                 bbox = pose_results[i]["bbox"]
278                 bboxlist = bbox.tolist()
279                 keypoints = pose_results[i]["keypoints"]
280                 kptlist = keypoints.tolist()
281                 poselist.append({"bbox": bboxlist, "keypoints":
282                                kptlist, "track_id": pose_results[i]["track_id"]})
283             pose_json.append(poselist)
284
285         # show the results
286         vis_frame = vis_pose_tracking_result(
287             pose_model,
288             cur_frame,
289             pose_results,
290             radius=kpt_radius,
291             thickness=line_thickness,
292             dataset=dataset,
293             dataset_info=dataset_info,
294             kpt_score_thr=0.3,
295             show=False)
296
297         progress = {"frame_id": frame_id, "image": vis_frame}
298         progress_callback.emit(progress)
299
300         if globals.kill_thread:

```

```
296         return pose_json
297
298     return pose_json
```